

ROBOTICS

Non-Invasive Brain-Actuated Control of a Mobile Robot

José del R. Millán^{1,2}, Frédéric Renkens², Josep Mouriño³, Wulfram Gerstner²

¹Dalle Molle Institute for Perceptual Artificial Intelligence (IDIAP)
CH-1920 Martigny, Switzerland. jose.millan@idiap.ch

²Laboratory of Computational Neuroscience, Swiss Federal Institute of Technology
CH-1015 Lausanne EPFL, Switzerland

³Centre de Recerca en Enginyeria Biomèdica, Universitat Politècnica de Catalunya
E-08028 Barcelona, Spain

Abstract

Recent experiments have indicated the possibility to use the brain electrical activity to directly control the movement of robotics or prosthetic devices. In this paper we report results with a portable non-invasive brain-computer interface that makes possible the continuous control of a mobile robot in a house-like environment. The interface uses 8 surface electrodes to measure electroencephalogram (EEG) signals from which a statistical classifier recognizes 3 different mental states. Until now, brain-actuated control of robots has relied on invasive approaches—requiring surgical implantation of electrodes—since EEG-based systems have been considered too slow for controlling rapid and complex sequences of movements. Here we show that, after a few days of training, two human subjects successfully moved a robot between several rooms by mental control only. Furthermore, mental control was only marginally worse than manual control on the same task.

1 Introduction

There is a growing interest in the use of physiological signals for communication and operation of devices for physically-disabled as well as able-bodied people. Over the last years evidence has accumulated to show the possibility to analyze brainwaves on-line in order to determine the subjects' mental state that is then mapped into actions such as selecting a letter from a virtual keyboard or moving a robotics device [Birbaumer *et al.*, 1999; Kennedy *et al.*, 2000; Millán, 2002; Millán *et al.*, 2002; Pfurtscheller and Neuper, 2001; Roberts and Penny, 2000; Serruya *et al.*, 2002; Taylor *et al.*, 2002; Wolpaw and McFarland, 1994; Wolpaw *et al.*, 2002]. This alternative communication and control channel, which does not require the user to perform any physical action, is called a *brain-computer interface (BCI)*.

A BCI may monitor a variety of brainwave phenomena. Most BCIs use electroencephalogram (EEG) signals;

i.e., the brain electrical activity recorded from electrodes placed onto the scalp. The main source of the EEG is the synchronous activity of thousands of cortical neurons. Measuring the EEG is a simple noninvasive way to monitor brain electrical activity, but it does not provide detailed information on the activity of single neurons (or small clusters of neurons) that could be recorded from microelectrodes surgically implanted in the cortex.

Some groups exploit evoked potentials—the automatic responses of the brain to external stimuli—recorded from either scalp or intracranial electrodes (for a review, see [Wolpaw *et al.*, 2002]). Evoked potentials are, in principle, easy to pick up but constrain the subject to synchronize themselves to the external machinery. A more natural and suitable alternative for controlling devices is to analyze components associated with spontaneous mental activity. Thus, some researchers measure slow cortical potentials—whose negative amplitudes are related to the overall preparatory excitation level of a given cortical network—over the top of the scalp [Birbaumer *et al.*, 1999]. Other groups look at local variations of EEG rhythms. The most used of such rhythms are related to the imagination of body movements and are recorded from the central region of the scalp overlying the sensorimotor cortex [Pfurtscheller and Neuper, 2001; Wolpaw and McFarland, 1994]. But, in addition to motor-related rhythms, other cognitive mental tasks are being explored [Millán *et al.*, 2002; Roberts and Penny, 2000] as a number of neurocognitive studies have found that different mental tasks—such as imagination of movements, arithmetic operations, or language—activate local cortical areas at different extents. In this case, rather than looking for predefined EEG phenomena as when using slow cortical potentials or movement rhythms, the approach aims at discovering mental-specific EEG patterns embedded in the continuous EEG signals. Yet, another kind of spontaneous signals is the direct activity of neurons in the motor cortex measured with implanted electrodes [Kennedy *et al.*, 2000; Serruya *et al.*, 2002; Taylor *et al.*, 2002; Wessberg *et al.*, 2000].

Recent experiments have shown the near possibility to use the brain electrical activity to directly control the

movement of robotics or prosthetic devices. In these experiments, several monkeys have been implanted with microelectrodes recording the activity of single neurons (their spiking rate) in the motor and premotor areas of the cortex. Then, the monkey's hand trajectory was predicted (and replicated online with a robot arm) from the activity of the neural populations [Wessberg *et al.*, 2000]. Also, after appropriate training, monkeys were able to move a computer cursor to desired targets using only their brain activity [Serruya *et al.*, 2002; Taylor *et al.*, 2002]. Until now, brain-actuated control of robots has been only tried with this kind of invasive approaches—requiring surgical implantation of electrodes—since EEG-based systems have been considered too slow for controlling rapid and complex sequences of movements. In this paper we show that two human subjects could, within a few days, learn to master a portable EEG-based brain-computer interface that recognized three mental states. Subjects successfully moved a robot between several rooms by mental control only. Furthermore, mental control was only marginally worse than manual control on the same task.

2 Brain Interface Protocol

EEG-based brain-computer interfaces are limited by a low channel capacity. Most of the current systems have a channel capacity below 0.5 bits/second [Wolpaw *et al.*, 2002]. One of the main reasons for such a low bandwidth is that they are based on synchronous protocols where EEG is time-locked to externally paced cues repeated every 4-10 s and the response of the BCI is the average decision over this period [Birbaumer *et al.*, 1999; Pfurtscheller and Neuper, 2001; Wolpaw and McFarland, 1994]. In contrast, our approach uses an asynchronous protocol that analyzes the ongoing EEG to determine the subject's mental state, which they can voluntarily change at any moment. The rapid responses of the BCI, together with its performance (see Section 3), give a theoretical channel capacity in between 1 and 1.5 bits/second.

Two volunteer healthy subjects "A" and "B" wore a commercial EEG cap with integrated electrodes (white spots in Figure 1). EEG potentials were recorded at the 8 standard fronto-centro-parietal locations F3, F4, C3, Cz, C4, P3, Pz, and P4. The sampling rate was 128 Hz. The raw EEG potentials were first transformed by means of a surface Laplacian (SL) computed globally by means of a spherical spline of order 2 [Perrin *et al.*, 1989, 1990]. This spatial filtering yields new potentials that should represent better the cortical activity due only to local sources below the electrodes. Then, we used the Welch periodogram algorithm to estimate the power spectrum of each channel over the last second. We averaged 3 segments of 0.5 second with 50% overlap, what yields a frequency resolution of 2 Hz. The values in the frequency band 8-30 Hz were normalized according to the total energy in that band. Thus an EEG sample has 96 features (8 channels times 12 components each). EEG samples were computed every 62.5 ms (i.e., 16 times per second).

During an initial training period of a few days, the two subjects learned to control 3 mental tasks of their choice. The subjects tried the following mental tasks: "relax", imagination of "left" and "right" hand (or arm) movements, "cube rotation", "subtraction", and "word association". The tasks consisted of getting relaxed, imagining repetitive self-paced movements of the limb, visualizing a spinning cube, performing successive elementary subtractions by a fixed number (e.g., $64-3=61$, $61-3=58$, etc.), and concatenating related words¹. After a short evaluation, the experimental subjects "A" and "B" chose to work with the combination of 3 tasks relax-left-cube and relax-left-right, respectively. In the sequel, we will refer to these mental tasks as #1, #2 and #3 (i.e., relax is #1, left is #2, and cube or right is #3). Neither subject had previous experience with BCIs or mental training.

Each day, subjects participated in four consecutive training sessions of about 5 min, separated by breaks of 5-10 min. During each training session subjects switched randomly every 10-15 s between the three tasks. Subjects received feedback online through three colored buttons on a computer screen. Each button is associated to one of the mental tasks to be recognized. A button flashed when an EEG sample is classified as belonging to the corresponding mental task. After each training session the statistical classifier was optimized offline. After this initial training, subjects learned to control mentally the mobile robot for 2 days. The results reported here were obtained at the end of the second day of work with the robot. During this training period, the user and the BCI engaged in a mutual learning process where they were coupled and adapted to each other.

3 Statistical Classifier

The mental tasks (or classes) are recognized by a Gaussian classifier trained to classify EEG samples as state #1, #2, #3 or "unknown". In this statistical classifier, every unit represents a prototype of one of the classes to be recognized. Its output gives an estimation of the posterior class probability distribution for an EEG sample. The challenge is to find the appropriate position, and receptive field, of the prototypes in the high-dimensional input space described above to differentiate the desired classes.

Although Gaussian classifiers are well known, our implementation differs from classical ones in a few respects. We assume that the class-conditional density function of class C_k is a superposition of N_k Gaussians (or prototypes) and that classes have equal prior probabilities. In our case, all the classes have the same number of prototypes, namely 4. In addition, we assume that all four prototypes have an equal weight of 1/4. Then, dropping constant terms, the posterior probability y_k of class C_k for sample x is

¹ Relax is done with eyes closed, whereas the other tasks are performed with eyes opened. But the recognition of the task "relax" is *not* based on the detection of eye movements.

$$y_k(x) = \frac{\sum_{i=1}^{N_k} a_k^i(x)}{\sum_{j=1}^{N_c} \sum_{i=1}^{N_j} a_j^i(x)} \quad (1)$$

where N_c is the number of classes and a_k^i is the activation level of the i th prototype of the class C_k

$$a_k^i(x) = |\Sigma_k|^{-1/2} \exp\left(-1/2(x - \mu_k^i)^T \Sigma_k^{-1} (x - \mu_k^i)\right) \quad (2)$$

where μ_k^i corresponds to the center of the i th prototype of class C_k , Σ_k is the covariance matrix of class C_k , and $|\Sigma_k|$ is the determinant of that matrix. In our case, Σ_k is diagonal and common to all the prototypes of the class. In this way, we reduce the number of parameters and pool data to increase the accuracy of their estimation.

The response of the network for sample x is the class C_k with the highest posterior probability provided that is greater than a given probability threshold of 0.85; otherwise the response is “unknown.” This rejection criterion keeps the number of errors (false positives) low, because recovering from erroneous actions (e.g., robot turning in the wrong direction) has a high cost. The choice of this probability threshold was guided by a previous ROC study where different subjects only carried out the initial training described before [Hauser *et al.*, 2002], and the actual value was selected based on the performance of the two subjects during the initial period of training.

To initialize the center of the prototypes and the covariance matrix of the class C_k we run a clustering algorithm (typically, a self-organizing map [Kohonen, 1997]) to compute the position of the desired number of prototypes. Then, the covariance matrix is

$$\Sigma_k = \frac{1}{S_k} \sum_{n=1}^{S_k} (x^n - \mu_k^n)(x^n - \mu_k^n)^T \quad (3)$$

where S_k denotes the number of training samples belonging to the class C_k and μ_k^n is the nearest prototype of this class to the sample x^n .

We then improve these initial estimations iteratively by stochastic gradient descent so as to minimize the mean square error. For every sample x in the training set, the update rule for all the prototypes of all the classes is

$$\Delta \mu_k^i(x) = \alpha(t_k(x) - y_k(x)) \Sigma_k^{-1} (x - \mu_k^i) \frac{a_k^i(x)}{A(x)} \sum_{j \neq k}^{N_c} y_j(x) \quad (4)$$

where α is the learning rate, t_k is the k th component of the target vector in the form *1-of-c* and A is the total activity of the network—i.e., the denominator in (1). Intuitively, during training, units are pulled towards the EEG samples of the mental task they represent and are pushed away from EEG samples of other tasks.

Finally, after every iteration over the training set, we estimate again the new value of Σ_k using expression (3).

It is possible to estimate the covariance matrices in more elaborated ways, including through gradient descent in order to minimize their contribution to the error function.

The brain-computer interface responds every 0.5 s. Firstly, it computes the class-conditioned probability for each class—i.e., the mixture of Gaussians in the numerator of Eq. (1). Secondly, it averages the class-conditioned probabilities over 8 consecutive samples. Thirdly, it estimates the posterior probability based on the average class-conditioned probability of each class using Bayes’ formula; cf. Eq. (1). Finally, it compares the posterior probability with a threshold value of 0.85. At the end of training, errors and “unknown” responses are below 5% and 30%, respectively. The theoretical channel capacity of the interface is hence above 1bit/second (operation mode I). In addition, the interface could also operate in another mode (operation mode II) where classification errors are further reduced by requiring that two consecutive periods of 0.5 s give the same classification response. In this mode II errors and “unknown” responses are below 2% and 40%, respectively, and the theoretical channel capacity is about 1 bit/second.

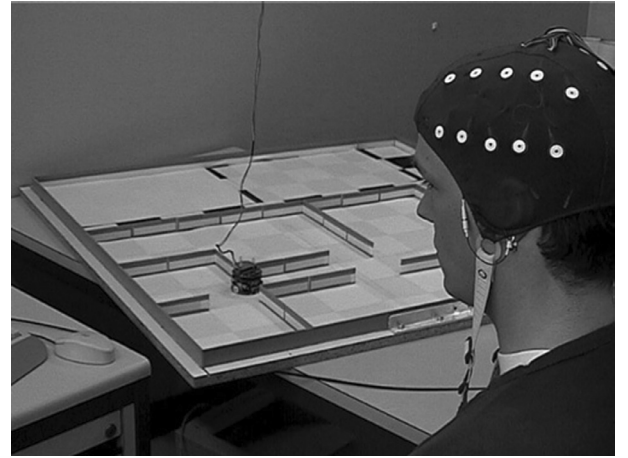


Figure 1. One of the experimental subjects while driving mentally the robot through the different rooms of the environment during the first experiment.

4 Robot Setup and Control

The task was to drive the robot through different rooms in a house-like environment (Figure 1). The robot was a small Khepera (5.7 cm diameter) that closely mimics a motorized wheelchair. The robot moved at a constant speed of one third of its diameter per second, similar to the speed of a wheelchair in an office building.

To make the robot move along a desired trajectory it is necessary to determine the speed of the motors controlling the wheels at each time step. Obviously, this is impossible by means of just three mental commands. A key idea is that the user’s mental states are associated to high-level commands (e.g., “turn right at the next occasion”) that the robot executes autonomously using the

readings of its on-board sensors. Another critical aspect for the continuous control of the robot is that subjects can issue high-level commands at any moment. This is possible because the operation of the BCI is asynchronous and does not require waiting for external cues, unlike synchronous approaches. The robot will continue executing a high-level command until the next is received.

The robot relies on a behavior-based controller [Arkin, 1998] to implement the high-level commands that guarantees obstacle avoidance and smooth turns. In this kind of controller, on-board sensors are read constantly and determine the next action to take. The mapping from the user's mental states (or commands) to the robot's behaviors is not simply one-to-one, but, in order to achieve a more flexible control of the robot, the mental states are just one of the inputs for a finite state automaton with 6 states (or behaviors). The transitions between behaviors are determined by the 3 mental states (#1, #2, #3), 6 perceptual states (as described by the robot's sensory readings: left wall, right wall, wall or obstacle in front, left obstacle, right obstacle, and free space) and a few internal memory variables. Figure 2 shows a simplified version of the finite state automaton. The memory variables were required to implement correctly the different behaviors. Thus, if the robot is performing the behavior "forward" and perceives a wall to the left, it switches automatically to the behavior "follow left wall". The actual transitions between the behaviors "forward" and "follow left/right wall" are not exactly as indicated in the figure, otherwise the robot would stay following walls forever. The transition to the behavior "forward" is necessary, for example, in the case the robot is approaching an open door and the user wants the robot not to enter into the room. On the other hand, the robot "stops" whenever it perceived an obstacle in front to avoid collisions (not all the transitions to the behavior "stop" appear in the figure for the sake of simplicity). Briefly, the interpretation of a mental state depends on the perceptual state of the robot. Thus, in an open space the mental state #2 means "left turn" while the same mental state is interpreted as "follow left wall" if a wall is detected on the left-hand side. Similarly, mental state #3 means "right turn" or "follow right wall"; mental state #1 always implied "move forward". Altogether experimental subjects felt that our control schema was simple and intuitive to use.

The Khepera robot is a two-wheeled vehicle. It has 8 infrared sensors around its diameter to detect obstacles. The sensors have a limited perception range, what makes difficult the recognition of the different perceptual states from the raw readings. To overcome this limitation, the robot uses a multilayer perceptron that maps the 8 raw infrared sensory readings into the current perceptual state.

A final element is the use of an appropriate feedback indicating the current mental state recognized by the embedded classifier. This is done by means of three lights on top of the robot, with the same colors as the buttons

used during the training phase. The front light is green and is on when the robot receives the mental command #1. The left light is blue and is associated to the mental command #2, whereas the right light is red and is associated to the mental command #3. Thus, if the robot is following the left wall and is approaching an open door, a blue feedback indicates that the robot will turn left to continue following the left wall (and, so, it will enter into the room). On the contrary, a green feedback indicates that robot will move forward along the corridor when facing the doorway and will not enter into the room. This simple feedback allows users to correct rapidly the robot's trajectory in case of errors in the recognition of the mental states or errors in the execution of the desired behavior (due to the limitations of the robot's sensors).

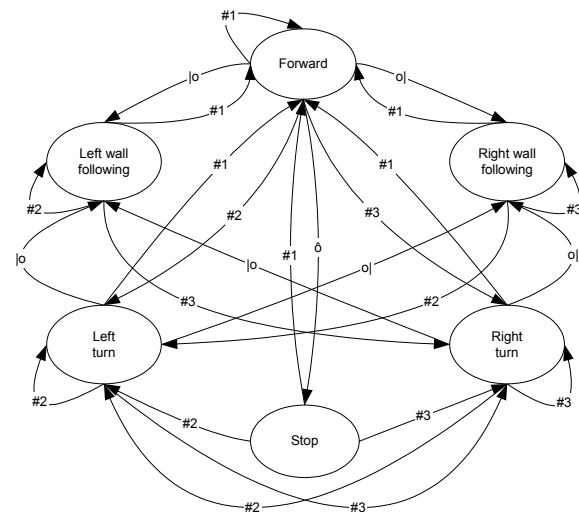


Figure 2. Finite state automaton used for the control of the robot. Transitions between the 6 behaviors were determined by 3 mental states (#1, #2, #3), 6 perceptual states (|o: left wall, |l: right wall, o: wall or obstacle in front), and some memory variables. The memory variables and some of the perceptual states are not shown for the sake of simplicity.

5 Experimental Results

After 5 and 3 days of initial training with the interface operating in mode I, subjects "A" and "B", respectively, achieved a satisfactory level of performance (correct recognition was above 60% while errors were below 5%). At this moment, subjects started to learn to control mentally the robot with the interface operating in mode II. During this second period of training subjects had to drive the robot mentally from a starting position to a first target room; once the robot arrived, a second target room was selected and so on. The starting position and the target rooms were drawn at random.

Figure 3 shows a trajectory generated by subject "A" after two days of training. The robot had to visit 3 different rooms, drawn randomly, starting from location "S".

Although the figure does not show the details of the trajectory inside the rooms, the robot made a short exploration in each of them. During the experiment, the subject was driving the robot for about 10 minutes continuously. Although the subject brought the robot to the desired room each time, there were a few occasions where the robot did not follow the optimal trajectory. This was mainly because the brain interface took a longer time than usual to recognize the subject's mental state. For instance, in one case the robot missed a turn because the brain interface did not recognize the appropriate mental state until the robot had passed the doorway of the desired room, and so the subject needed to maneuver mentally the robot to bring it back. In other situations, the robot's sensors perceived a wall or corner too close, thus making the robot stop automatically to avoid collisions. In these situations, the subject needed to turn (by mental control) the robot away from the phantom obstacle and then resume the trajectory.

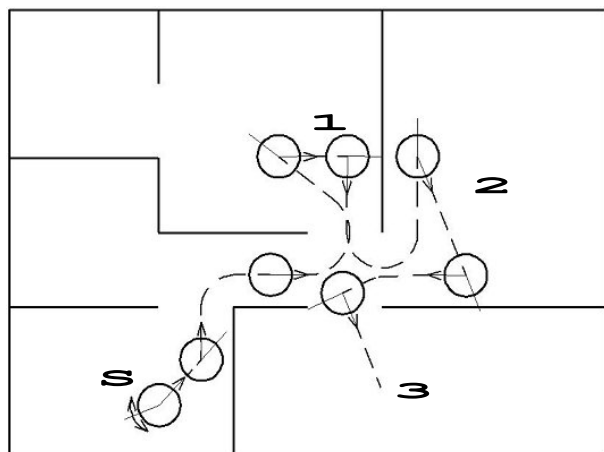


Figure 3. Trajectory followed by the robot under the mental control of subject “A” during one of the trials of the first experiment. The robot started in the bottom left room and then visited 3 other rooms, top center, top right and bottom right, sequentially. The figure does not show the details of the trajectory inside the rooms.

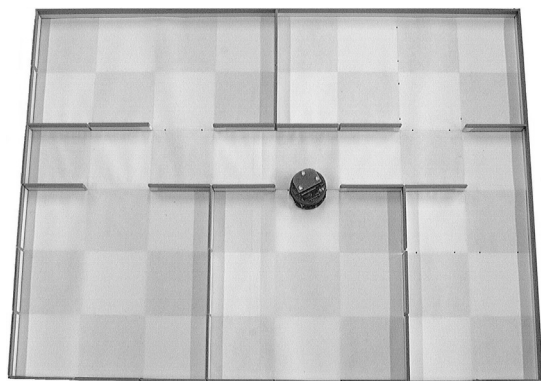


Figure 4. Environment used for the second set of experiments.

Qualitatively, the trajectory is rather good as the robot visited the 4 rooms in the desired order and it was never necessary to make significant corrections to the robot's active behaviors. But in order to evaluate *quantitatively* the performance of the brain-actuated robot, subjects “A” and “B” also carried out a second set of experiments in a slightly different arrangement of the rooms that were now located along the two sides of a corridor (Figure 4).

In a given trial, the robot must travel from a starting room to a target room as well as also visiting an intermediate room. The rooms and their order were selected at random. First, the subject made the robot visit the desired sequence of rooms by mental control. In a later session, the subject drove the robot along the same sequence of rooms by manual control. In this case, the subject used the same controller described above but, instead of sending mental commands to the robot, he simply pressed one of three keys. This procedure allowed us to compare mental and manual control for a system that is identical in all other aspects. In addition, the manual trajectory should be quite close to the optimal path that can be generated with the current controller. It is worth noting that the reason why the subject controls the robot mentally first and only afterwards manually is to avoid any learning process that could facilitate mental control.

Table 1 gives the time in seconds necessary to generate the desired trajectory for three different trials for the two subjects. For each trial, the table indicates the time required for mental control and manual control. Surprisingly, we can see that mental control was only marginally worse than manual control. On average, brain-actuated control of the robot is only 35% longer than manual control for both subjects.

Table 1. Time in seconds for three different trials where subjects “A” and “B” controlled the robot first mentally and then manually.

Subject	Trial	Mental	Manual
“A”	1	149	124
	2	183	135
	3	191	129
	<i>Average</i>	<i>174</i>	<i>129</i>
“B”	1	219	156
	2	189	155
	3	175	117
	<i>Average</i>	<i>194</i>	<i>143</i>

6 Discussion

In this paper we have reported first results of a brain-actuated mobile robot by means of a portable non-invasive BCI. Although the quality and resolution of the brain signals measured with our EEG system are not comparable to those recorded by implanted electrodes, they are sufficient to operate robots in indoor environments. This is possible because of the combination of advanced robotics, an asynchronous protocol for the

analysis of online EEG signal, and machine learning techniques.

The work described in this paper suggests that it could be possible for human subjects to mentally operate a wheelchair. But porting the current results to the wheelchair is not straightforward for, at least, two reasons. First, the performance of the BCI will suffer once the subject is seated on a mobile platform. This will require longer training times for the subject. Second, the current finite state automaton only allows for simple control actions, and so the resulting wheelchair could be too constrained for practical use in cluttered environments. In this respect, recent progress in EEG analysis [Michel *et al.*, 2001] suggests that a sufficient number of mental states can be recognized to control robotics and prosthetic devices and in a more natural and flexible way. In this approach we will transform scalp potentials—recorded with a sufficiently high number of electrodes (32, 64 or more)—to brain maps to get detailed information on the activity of small cortical areas. The Gaussian classifier embedded in the BCI would work upon selected parts of these brain maps instead of using EEG features.

Acknowledgments

JRM is supported by the Swiss National Science Foundation through the National Centre of Competence in Research on “Interactive Multimodal Information Management (IM2).”

References

- [Arkin, 1998] R. C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, Massachusetts, 1998.
- [Birbaumer *et al.*, 1999] N. Birbaumer, N. Ghanayim, T. Hinterberger, I. Iversen, B. Kotchoubey, A. Kübler, J. Perelmouter, E. Taub, and H. Flor. A spelling device for the paralysed. *Nature*, 398:297–298, 1999.
- [Hauser *et al.*, 2002] A. Hauser, P.-E. Sottas, and J. del R. Millán. Temporal processing of brain activity for the recognition of EEG patterns. In *Proc International Conference on Artificial Neural Networks*, pages 1125–1130, Madrid, Spain, August 2002. Springer-Verlag.
- [Kennedy *et al.*, 2000] P. R. Kennedy, R. Bakay, M. M. Moore, K. Adams, and J. Goldwaithe. Direct control of a computer from the human central nervous system. *IEEE Trans. on Rehabilitation Engineering*, 8:198–202, 2000.
- [Kohonen, 1997] T. Kohonen. *Self-Organizing Maps*, 2nd ed. Springer-Verlag, Berlin, 1997.
- [Michel *et al.*, 2001] C. M. Michel, G. Thut, S. Morand, A. Khateb, A. Pegna, R. Grave de Peralta, S. Gonzalez, M. Seeck, and T. Landis. Electric source imaging of human brain functions. *Brain Research Reviews*, 36:108–118, 2001.
- [Millán, 2002] J. del R. Millán. Brain-computer interfaces. In M. A. Arbib (ed.), *Handbook of Brain Theory and Neural Networks*, pp. 178–181. MIT Press, Cambridge, Massachusetts, 2002.
- [Millán *et al.*, 2002] J. del R. Millán, J. Mouriño, M. Franzé, F. Cincotti, M. Varsta, J. Heikkonen, and F. Babiloni. A local neural classifier for the recognition of EEG patterns associated to mental tasks. *IEEE Trans. on Neural Networks*, 13:678–686, 2002.
- [Perrin *et al.*, 1989] F. Perrin, J. Pernier, O. Bertrand, and J. Echallier. Spherical spline for potential and current density mapping. *Electroencephalography and Clinical Neurophysiology*, 72:184–187, 1989.
- [Perrin *et al.*, 1990] F. Perrin, J. Pernier, O. Bertrand, and J. Echallier. Corrigendum EEG 02274. *Electroencephalography and Clinical Neurophysiology*, 76:565, 1990.
- [Pfurtscheller and Neuper, 2001] G. Pfurtscheller and C. Neuper. Motor imagery and direct brain-computer communication. *Proceedings of the IEEE*, 89:1123–1134, 2001.
- [Roberts and Penny, 2000] S. J. Roberts and W. D. Penny. Real-time brain-computer interfacing: A preliminary study using Bayesian learning. *Medical & Biological Engineering & Computing*, 38:56–61, 2000.
- [Serruya *et al.*, 2002] M. D. Serruya, N. G. Hatsopoulos, L. Paninski, M. R. Fellows, and J. Donoghue. Instant neural control of a movement signal. *Nature*, 416:141–142, 2002.
- [Taylor *et al.*, 2002] D. M. Taylor, S. I. Helms Tillery, and A. B. Schwartz. Direct cortical control of 3D neuroprosthetic devices. *Science*, 296:1829–1832, 2002.
- [Wessberg *et al.*, 2000] J. Wessberg, C. R. Stambaugh, J. D. Kralik, P. D. Beck, M. Laubach, J. K. Chapin, J. Kim, S. J. Biggs, M. A. Srinivasan, and M. A. L. Nicolelis. Real-time prediction of hand trajectory by ensembles of cortical neurons in primates. *Nature*, 408:361–365, 2000.
- [Wolpaw and McFarland, 1994] J. R. Wolpaw and D. J. McFarland. Multichannel EEG-based brain-computer communication. *Electroencephalography and Clinical Neurophysiology*, 90:444–449, 1994.
- [Wolpaw *et al.*, 2002] J. R. Wolpaw, N. Birbaumer, D. J. McFarland, G. Pfurtscheller, and T. M. Vaughan. Brain-computer interfaces for communication and control. *Clinical Neurophysiology*, 113:767–791, 2002.

Exploring Unknown Environments with Mobile Robots using Coverage Maps

Cyrill Stachniss Wolfram Burgard

University of Freiburg
Department of Computer Science
D-79110 Freiburg, Germany *

Abstract

In this paper we introduce coverage maps as a new way of representing the environment of a mobile robot. Coverage maps store for each cell of a given grid a posterior about the amount the corresponding cell is covered by an obstacle. Using this representation a mobile robot can more accurately reason about its uncertainty in the map of the environment than with standard occupancy grids. We present a model for proximity sensors designed to update coverage maps upon sensory input. We also describe how coverage maps can be used to formulate a decision-theoretic approach for mobile robot exploration. We present experiments carried out with real robots in which accurate maps are build from noisy ultrasound data. Finally, we present a comparison of different view-point selection strategies for mobile robot exploration.

1 Introduction

Generating maps is one of the fundamental tasks of mobile robots and many researchers have focused on the problem of how to represent the environment as well as how to acquire models using this representation [5, 9, 10, 14, 17]. The mapping problem itself has several aspects that have been studied intensively in the past. Some of the most important aspects are the localization of the vehicle during mapping, appropriate models of the environment and the sensors, as well as strategies for guiding the vehicle. In literature, the localization problem plays an important role, since the quality of the resulting map depends strongly on the accuracy of the pose estimates during the mapping process. However, the accuracy of the map also depends on the choice of view-points during exploration. Especially if noisy sensors are used, the map will be quite inaccurate in areas which have been sensed a few times only or maybe even from disadvantageous view-points.

Exploration is the task of guiding a vehicle in such a way that it covers the environment with its sensors. Efficient exploration strategies are also relevant for surface inspection,

mine sweeping, or surveillance [3, 12]. In the past, several strategies for exploration have been developed. One group of approaches deals with the problem of simultaneous localization and mapping [1, 4], an aspect that we do not address in this paper. A common technique for exploration strategies is to extract frontiers between known and unknown areas [2, 7, 20] and to visit the nearest unexplored place. These approaches only distinguish between scanned and un-scanned areas and do not take into account the actual information gathered at each view-point. To overcome this limitation, Gonzales et al. [8] determine the amount of unseen area that might be visible to the robot from possible view-points. To incorporate the uncertainty of the robot about the state of the environment Moorehead et al. [13] as well as Bourgault et al. [1] use occupancy grids [14] and compute the entropy of each cell in the grid to determine the utility of scanning from a certain location. Whaithe and Ferrie [19] present an approach that also uses the entropy to measure the uncertainty in the geometric structure of objects that are scanned with a laser range sensor. In contrast to the work described here they use a parametric representation of the objects to be scanned. Edlinger and Puttkamer [7] developed a hierarchical exploration strategy for office environments. Their approach first explores rooms and then traverses through doorways to explore other parts of the environment. Taylor and Kriegman [16] describe a system for visiting all landmarks in the environment of the robot. Their robot maintains a list of unvisited landmarks that are approached and mapped by the robot. Dudek et al. [6] propose a strategy for exploring an unknown graph-like environment. Their algorithm does not consider distance metrics and is designed for robots with very limited perceptual capabilities. Recently Koenig has shown, that a strategy, which guides the vehicle to the closest point that has not been covered yet, keeps the traveled distance reasonably small [11]. However, as experiments reported in this paper illustrate, such techniques can lead to a serious increase of measurements necessary to build an accurate map if the robot is not able to incorporate measurements on-the-fly while it is moving. This might be the case, for example, for robots extracting distance information from camera images.

In this paper we introduce coverage maps as a new probabilistic way to represent the belief of the robot about the state of the environment. In contrast to occupancy grids [14], in which each cell is considered as either occupied or free, cov-

*This work has partly been supported by the German Science Foundation (DFG) under contract number SFB/TR8-03.



Figure 1: Typical occupancy map obtained in situations in which cells are only partly occupied (left image) and a coverage map containing the corresponding coverage values (right image).

erage maps represent in each cell of a given discretization a posterior about the amount this cell is covered by an object. As an example consider the situation depicted in Figure 1 in which a cell is partly covered by an obstacle. With the standard occupancy algorithm the probability that this cell is occupied will converge to 1 if the sensors of the robot repeatedly detect the obstacle. The left picture of this figure shows the resulting occupancy probabilities (black represents high likelihood). Since the object does only cover 20% of this cell, a coverage value of .2 (as shown in the right image of Figure 1) would be a better approximation of the true situation. In addition to the representation aspect, we also present a sensor model that allows the robot to appropriately update a coverage map upon sensory input and describe how coverage maps can be used to realize a decision-theoretic approach to exploration of unknown environments.

This paper is organized as follows. In the next section we introduce coverage maps. In Section 3 we present a probabilistic technique to update a given coverage map upon sensory input. In Section 4 we describe a decision-theoretic approach to exploration based on coverage maps. In Section 5 we present experiments illustrating that our approach allows a mobile robot can learn accurate maps from noisy range sensors. Additionally, we present experiments comparing different view-point selection strategies for exploration.

2 Coverage Maps

As already mentioned above, occupancy grids rest on the assumption that the environment has binary structure, i.e. that each grid cell is either occupied or free. This assumption, however, is not always justified. For example, if the environment contains a wall that is not parallel to the x- or y-axis of the grid there must be grid cells which are only partly covered. In occupancy grids the probability that such cells are occupied will inevitably converge to one (see Figure 1). Coverage maps overcome this limitation by storing a posterior about its coverage for each cell. Coverage values range from 0 to 1. Whereas a coverage of 1 means that the cell is fully occupied, an empty cell has a coverage of 0. Since the robot usually does not know the true coverage of a grid cell, it maintains a probabilistic belief $p(c_l)$ about the coverage of the cell c_l . In principle, there are different ways of representing $p(c_l)$. They range from parametric distributions such as (mixtures of) Gaussians or non-parametric variants such as histograms. Throughout this paper we assume that each $p(c_l)$ is given by a histogram over possible coverage values. More precisely, we store a histogram for each grid cell, in which each bin contains the probability that the corresponding grid cell has the particular coverage. A typical example is depicted in Figure 2. It shows the posterior for the cell containing the obstacle in the situation illustrated in Figure 1 obtained after 30 measurements.

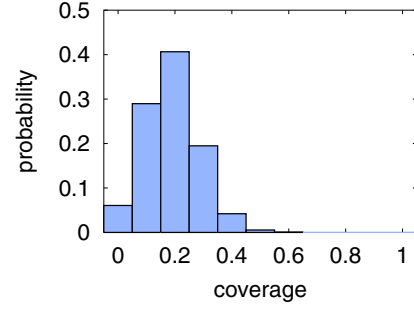


Figure 2: Coverage posterior for the cell containing the obstacle in the situation depicted in Figure 1.

3 Updating Coverage Maps

To update a coverage map based on sensory input, we apply a Bayesian update scheme similar to that of occupancy grids. Throughout this paper we assume that our sensor provides distance information. Accordingly, we have to convert the distance information to coverage values. What we need to know is the coverage map c that has the highest likelihood under all distance measurements $d_0, \dots, d_T$. If we use Bayes rule and assume that consecutive measurements are independent given that we know the map c , we obtain:

$$p(c \mid d_0, \dots, d_T) \stackrel{\text{Bayes}}{=} \frac{p(d_0, \dots, d_T \mid c) \cdot p(c)}{p(d_0, \dots, d_T)} \quad (1)$$

$$= \beta \cdot p(c) \cdot p(d_0, \dots, d_T \mid c) \quad (2)$$

$$= \beta \cdot p(c) \cdot \prod_{t=0}^T p(d_t \mid c) \quad (3)$$

Next we need to know how to determine the likelihood $p(d_t \mid c)$ of measuring d_t given the map c . Again we apply Bayes rule and obtain:

$$p(c \mid d_0, \dots, d_T) \stackrel{\text{Bayes}}{=} \beta \cdot p(c) \cdot \prod_{t=0}^T \frac{p(c \mid d_t) \cdot p(d_t)}{p(c)} \quad (4)$$

$$= \beta \cdot \frac{\xi^{T+1}}{p(c)^T} \cdot \prod_{t=0}^T p(c \mid d_t) \quad (5)$$

$$= \beta' \cdot \prod_{t=0}^T p(c \mid d_t) \quad (6)$$

Equation (5) is obtained from Equation (4) by assuming that $p(c)$ is constant and that $\xi = p(d_t)$ is constant for every t . The variables β and β' represent normalization constants ensuring that the left-hand side sums up to one over all c . Under the strong assumption that the individual cells of a coverage map are independent¹ we finally obtain:

$$p(c \mid d_0, \dots, d_T) = \beta' \cdot \prod_{t=0}^T \prod_{l=0}^L p(c_l \mid d_t) \quad (7)$$

$$= \beta' \cdot \prod_{l=0}^L \prod_{t=0}^T p(c_l \mid d_t) \quad (8)$$

¹This independence is frequently assumed in the context of occupancy maps. We would like to refer to a recent work by Thrun [18] on how to better deal with the dependency between cells.

Thus, to update a map given a measurement d_t we simply have to multiply the current belief about the coverage of each cell c_l by the belief about the coverage of this cell resulting from d_t . Additionally the maximum likelihood coverage map is obtained by choosing the mode of the coverage histogram for each cell c_l .

It remains to describe how we actually compute $p(c_l | d_t)$, i.e. how we determine the distribution about the potential coverage values of a cell c_l with distance d^l to the sensor given a measurement d_t . In our current system, we use a mixture of a Gaussian $\mathcal{N}(\mu, \sigma)$ and a uniform distribution γ to compute the probability $p(c_l = x | d_t)$ that the coverage of c_l is x :

$$p(c_l = x | d_t) = \gamma(d^l, d_t) + \mathcal{N}(\mu(d^l - d_t), \sigma(d^l, d_t), x). \quad (9)$$

The value of the uniform distribution is computed using the function $\gamma(d^l, d_t)$ which increases monotonously in d^l and d . It reflects a typical behavior of proximity sensors like sonars, because the accuracy of a measurement decreases with the distance to the obstacle.

The mean $\mu(d^l - d)$ of the Gaussian is computed in the following way:

$$\mu(d^l - d) = \begin{cases} 0, & (d^l - d) < -\frac{r}{2} \\ \frac{r}{2} + d^l - d, & |d^l - d| < \frac{r}{2} \\ 1, & (d^l - d) > \frac{r}{2} \end{cases} \quad (10)$$

where r is the grid resolution of the map. Note that we distinguish three situations, depending on whether the measurement ends in c_l or not. Suppose that the measurement does not end in c_l and the distance d^l is shorter than d . In this case we have $d^l - d < -\frac{r}{2}$. In such a situation, the mean of the Gaussian is zero, since it is more likely that a cell covered by a range measurement that does not end in it is completely empty. The second line of Equation (10) represents the situation in which d ends within c_l . In this case the mean is inverse proportional to the amount the cell is covered by d . Finally, cells lying up to 20cm behind a cell, in which the measurement ends, are most likely completely occupied so that the mean is 1.

The value of the standard deviation $\sigma(d^l, d)$ of the Gaussian also is a function that is monotonously increasing in d^l and d except when $|d^l - d| < \frac{r}{2}$. In this range $\sigma(d^l, d)$ has a constant value that exceeds all values outside of this interval.

To obtain the optimal parameters for the functions in our sensor model we apply the maximum likelihood principle. We first apply a highly accurate scan-alignment procedure [10] on laser range information. Next we manually extract geometric objects from the corresponding range data. Given these geometric primitives we use straightforward geometric projections to compute the ground truth information, i.e., the exact coverage of each cell of a given discretization. We evaluate a particular setting for the parameters of our model by determining the likelihood of the ground truth given a coverage map obtained using this setting. To maximize the likelihood we apply local search techniques in the parameter space. Figure 3 plots $p(c_l = x | d)$ for a measured distance of $d = 1\text{m}$ obtained for ultrasound data recorded with our B21r robot Albert, which is depicted in picture (a) of Figure 4. As the plot illustrates, for a measured distance of 1m , cells close

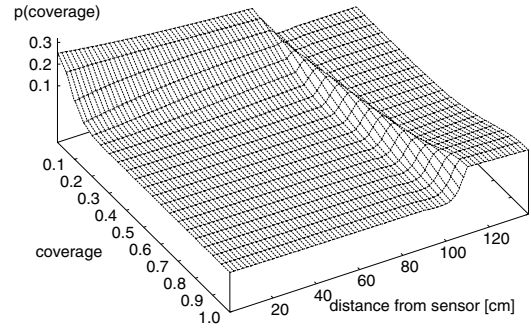


Figure 3: Sensor model $p(c_l = x | d)$ for a measured distance $d = 100\text{cm}$.

to the robot are unoccupied with high likelihood. However, cells close to the measured distance are covered with high probability. Figure 4 shows two coverage maps build from real sonar data obtained by a real robot. Image (b) depicts a map of the Sieg Hall at the University of Washington and (c) a map of the office environment of our laboratory.

4 Strategies for Choosing the Next View-Point

One of the key problems during exploration is to choose appropriate vantage points. At the selected location the robot will perform the next measurement to retrieve new information about its environment. In this section we will present four methods to choose an appropriate position. In general there are two different aspects that are relevant for the view-point selection. On the one hand, the uncertainty of the robot in the map should be as small as possible, and on the other hand, the number of measurements to be incorporated as well as the distance traveled should be minimized.

To determine the uncertainty in the state of a particular cell we consider the entropy of the posterior for that cell. Entropy is a general measure for the uncertainty of a belief. The entropy H of a histogram h consisting of n bins h_i ($i = 1, \dots, n$) is defined as:

$$H(h) = - \sum_{i=1}^n p(h_i) \cdot \log p(h_i). \quad (11)$$

H is maximal in case of a uniform distribution. The minimal value zero is obtained if the system is absolutely certain about the state of the corresponding cell. Thus, if we want to minimize the uncertainty in the current map, all we need to do is to reduce the entropy of the histograms in the coverage map. Furthermore, we can specify, at which moment the exploration task has been completed. Suppose the size of the environment is limited. Then the goal of the exploration process for a coverage map c has been achieved if $H(h(c_l)) < \epsilon$ for all cells $c_l \in c$ that can be reached by the robot. Here $h(c_l)$ corresponds to the histogram representing the coverage of c_l . Additionally, the system has to detect a situation in which the robot is unable to reduce the entropy of a cell below ϵ to ensure the termination of the exploration task. In our system this is achieved by monitoring the change of entropy. If this change is below .001 for five consecutive measurements, the cell is regarded as explored enough.

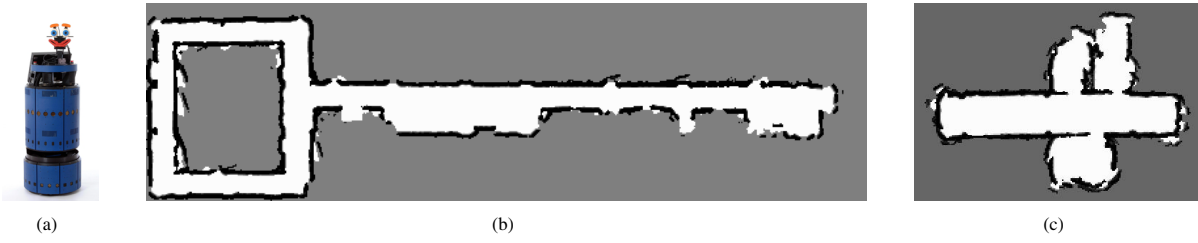


Figure 4: Picture (a) shows our B21r robot Albert equipped with a ring of 24 sonar sensors. Image (b) depicts a coverage map generated for data collected in the Sieg Hall building of the University of Washington. Picture (c) displays the coverage map of our laboratory environment.

4.1 Closest Location (CL)

A simple and frequently used strategy is to drive to the closest location at which the robot can gather information about a cell that has not been explored well enough. This way the question whether a cell has been explored well enough depends on the actual measure used to quantify the uncertainty about individual grid cells. In our approach this measure is the entropy. A cell is regarded as been sensed accurately enough if the entropy of the coverage belief does not exceed ϵ or if it does not change any longer. This strategy CL does not take into account how much information will be obtained at a particular view-point. Rather it seeks to minimize the distance to that location:

$$l_{next} = \operatorname{argmin}_{l \in L(c)} d_c(l, x), \quad (12)$$

where $L(c)$ is the set of cells which have a grid cell with high entropy in its visible range and $d_c(l, x)$ is the distance between the locations l and x given the current map c of the environment.

4.2 Maximum Information Gain (IG) and (IG_WIN)

The second strategy is solely governed by the information gain that can be obtained about the environment at a specific view-point. The information gain is defined as the change of entropy introduced by incorporating the measurement obtained at that location into the map. If we integrate a single measurement d into a cell c_l , the information gain is defined as:

$$I(h(c_l) | d) = H(h'_d(c_l)) - H(h(c_l)) \quad (13)$$

where $h'_d(c_l)$ is the histogram of cell c_l after integrating measurement d according to our sensor model. The information gain of a measurement is then computed as the sum of the information gains for all cells covered by that measurement. Since we do not know which measurement we will receive if the robot measures at a certain position l , we have to integrate over all possible measurements to compute the expected information gain for that view-point:

$$E[I(l)] = \sum_d p(d | c) \cdot \sum_{c_i \in C(l, d)} I(h(c_i) | d) \quad (14)$$

Here $C(l, d)$ is the set of cells covered by measurement d . To efficiently compute the likelihood of an observation $p(d | c)$ we apply a ray-tracing technique similar to Moravec and Elfes [14] using the current maximum likelihood coverage map.

Since the complexity of Equation (14) depends exponentially on the number of dimensions of the measurement, we consider all measurements independently. For example, for our robot equipped with 24 ultrasound sensors we compute the average information gain over all 24 sensors.

The next view-point is then defined as:

$$l_{next} = \operatorname{argmax}_{l \in L(c)} E[I(l)] \quad (15)$$

One of the disadvantages of this strategy is that it does not take into account the distance to be traveled by the robot. To deal with this problem we also consider the strategy IG_WIN which restricts the search for potential vantage points to a local window until this has been explored. Once this has been done, there is no need for the robot to return to this area again.

4.3 Combination of IG and CL (IG_CL)

The final strategy discussed in this paper tries to combine the properties of the strategies CL and IG. The goal is to find an optimal tradeoff between the evaluation functions (12) and (15):

$$l_{next} = \operatorname{argmax}_{l \in L(c)} \left[\alpha \cdot \frac{E[I(l)]}{\max_{l' \in L(c)} E[I(l')]} - \frac{d_c(l, x)}{\max_{l' \in L(c)} d_c(l', x)} \right] \quad (16)$$

By adapting the weight α the user can easily influence the behavior of a robot and optimize its performance for a special task. A value close to zero results in a behavior similar to the strategy CL. For high values of α the strategy converges to the strategy IG. Please note that functions of this type have successfully been applied in the past for coordinating multiple robots during exploration [2, 15].

5 Experiments

We implemented the models and exploration strategies described above and performed a series of exploration runs in different environments. The goal of the experiments presented in this section is to illustrate that a robot can build accurate maps using our models. Additionally we describe experiments in which we analyze the properties of the four view-point selection techniques described above.

5.1 Advantage over Scan Counting

The first experiment is designed to illustrate that an approach which considers the uncertainty in the belief about the coverage of a cell to select view-points yields more accurate maps than techniques relying on scan counting approaches. Scan

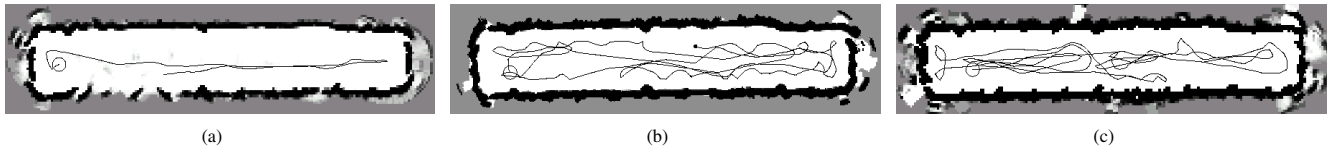


Figure 5: These images depict maps and trajectories of the robot obtained in the corridor of our office environment. The left map (a) is a typical occupancy grid map obtained when every cell needs to be covered just once. Image (b) shows a coverage map obtained with the strategy CL. The rightmost image (c) shows an occupancy map obtained using scan counting with a threshold of $n = 50$.

technique	path length	$\{c_i \mid H(h(c_i)) > 0.65\}$
coverage maps	89.1m	0%
counting	26.6m	21%
ext. counting	90.6m	1.5%

Table 1: Data from the experiments shown in Figure 5. To compute the entropy for the scan counting results we generated coverage maps from the recorded sonar data.

counting techniques count the number of times a cell is intercepted by a measurement. Several exploration techniques [2, 7, 20] assume that a place is explored if it has been scanned once. This is problematic especially when the underlying sensors are noisy. Figure 5 (a) shows a typical occupancy grid map of our laboratory environment obtained from real sonar data when using the scan counting technique. Since the exploration process is stopped as soon as all reachable locations were covered by a measurement, many cells of the resulting occupancy map have a high uncertainty (see Table 1). This is due to the high amount of sensor noise, which sonars often produce. On the other hand, if we use our approach and consider the uncertainty in the coverage of individual cells to select view-points, the resulting maps are more accurate (see Figure 5(b)).

Obviously, a straightforward extension of the scan counting would be to assume that each cell has to be covered n times and not only once. An occupancy grid obtained by this extension can be seen in Figure 5 (c). In this experiment we additionally discounted longer beams in order to account for the fact that range sensors provide fewer information for distant places. Whereas this map looks similar as the corresponding coverage map, the robot’s uncertainty about the state of the environment is higher. This is illustrated by the values given in Table 1. The right column of this table contains the percentage of cells in c for which the entropy exceeds the given threshold. Thus, even extended scan counting does not guarantee that in the end every cell is explored well enough. Typically, some cells will be measured too often, others not often enough.

Figure 6 shows the result of a series of 50 simulation experiments. In these experiments we forced the robot to reach a scan count of n where n varied between 1 and 130.² We counted the number of cells m that would be explored well enough given the entropy criterion for coverage maps and plotted the length of the overall path against m . The resulting graph is shown in Figure 6. The cross on the right side indi-

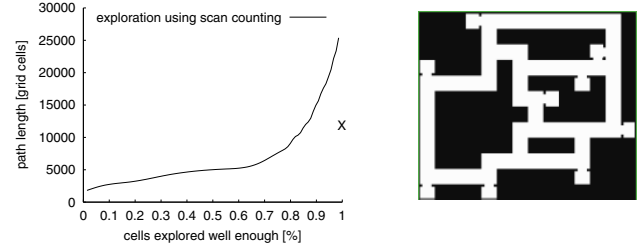


Figure 6: Average path length of the robot depending on the number of sufficiently explored cells (left image) and the simulation environment used during this experiment (right image).

cates the path length obtained when using the strategy CL for coverage maps. As can be seen, if more than 85% of the map should be explored well enough, a decision-theoretic exploration strategy yields shorter trajectories than extended scan counting.

5.2 A Comparison of the View-Point Selection Strategies

Robots performing 2-d exploration tasks with sonars oder laser scanners normally integrate every sensor measurement because the amount of data is reasonably small and easy to integrate. In this section we consider the situation that analyzing a measurement produces high costs. This might be the case if e.g. the distance information needs to be extracted from stereo images. In such a situation the number of measurements needed for the exploration task is a value of interest. As mentioned above, one of the major advantages of our coverage maps is that they allow the integration of the uncertainty into the selection process of the next view-point. The experiments in this section are designed to compare the performance of the different strategies. To carry out the experiments, we varied the size of the local window when using IG_WIN and the weight α in the evaluation function of IG_CL (see Equation (16)). In Figure 7 (a) and (b) the numbers behind IG_CL show the value of the weight α and the numbers behind IG_WIN indicate the radius of a circle which defines the local window. The results have been obtained using 20 runs per strategy in the environment shown in Figure 7 (d). Please note that further experiments carried out in alternative environments showed similar results and are omitted for the sake of brevity. The maximum allowed entropy during all experiments described in this section was set to .6.

Figure 7 (a) shows the average number of measurements necessary to complete the exploration task for each strategy. As can be seen from the figure, the strategy IG needs the minimum amount of measurements. The strategy IG_CL with $\alpha = 1.0$ needs approximately the same number of measure-

²In practice, a good candidate value of n would be the maximum number of measurements necessary to obtain a coverage map that fulfills the entropy threshold criterion

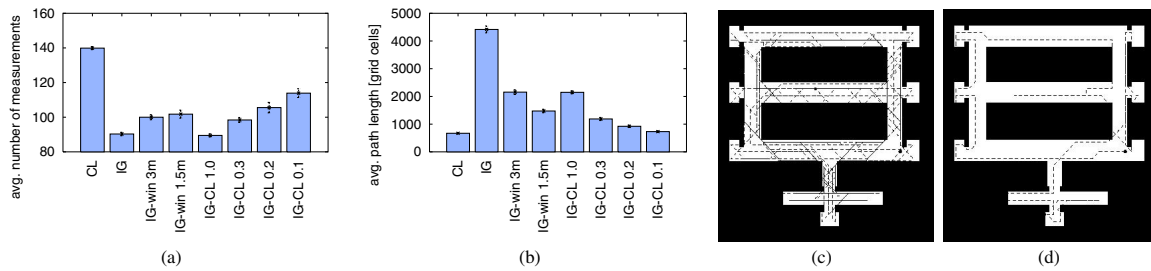


Figure 7: Figure (a) shows the average number of measurements needed by different strategies, whereas in (b) depicts the average path length of the exploration for each strategy. The value behind IG_WIN shows the size of the local window and behind IG_CL the value of the parameter α . The error-bars show the .05 significance interval. The images (c) and (d) show the paths driven during the exploration experiment. Figure (c) shows a sample path of method IG and the (d) a path generated by CL.

ments as IG. The strategy CL requires the maximum number of measurements compared to all other strategies considered here. The reason is that it only seeks to minimize the path length without considering the information gained at particular locations. The error-bars correspond to the 5% confidence interval.

In our experiments we found that a nearest neighbor view-point selection strategy like CL outperforms an approach considering information gain if the robot is allowed to integrate measurements while it is moving (assuming that the acquisition and integration of measurements can be done fast). This can be seen in Figure 7 (b), which plots the average path length driven by the robot during the exploration task for all different strategies. With respect to the path length the strategy CL shows the best behavior as the resulting trajectories are shorter than those of all other techniques. Thus, the CL strategy efficiently covers the terrain. In contrast to that, the IG strategy ignores the distance to be driven and therefore produces an extremely long path which results in the worst behavior of all strategies (see also Figure 7 (c) & (d)). The IG-CL strategy with $\alpha = .3$ appears to yield a good trade-off between number of measurements and overall path length. According to the experiments it slightly outperforms the IG-WIN strategy.

6 Conclusions

In this paper we have introduced a new representation scheme for maps build with mobile robots from sensor data. In contrast to standard occupancy maps our coverage maps store a posterior for each cell about its coverage. This offers the opportunity to more accurately compute the uncertainty of the robot about the corresponding area in the environment. Additionally, we have presented a sensor model designed to update these maps upon sensory input. Finally, we have compared decision-theoretic approaches to guide a vehicle during exploration.

The technique has been implemented and evaluated in extensive simulation runs and real world applications. The experiments illustrate that by using coverage maps it is possible to build accurate maps even if noisy sensors are used. Experiments analyzing different exploration strategies indicate that a technique combining the maximum uncertainty reduction and the distance to be traveled yields the best trade-off between the number of necessary measurements and the length of the resulting paths.

References

- [1] F. Bourgault, A. A. Makarenko, S. B. Williams, B. Grocholsky, and F. Durrant-Whyte. Information based adaptive robotic exploration. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2002.
- [2] W. Burgard, M. Moors, and F. Schneider. Collaborative exploration of unknown environments with teams of mobile robots. In *dagstuhl*. Springer Verlag, 2002.
- [3] H. Choset. Coverage for robotics - a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31, 2001.
- [4] H. Choset. Topological simultaneous localization and mapping (slam): Toward exact localization without explicit localization. *IEEE Transactions on Robotics and Automation*, 17(2), April 2001.
- [5] G. Dissanayake, H. Durrant-Whyte, and T. Bailey. A computationally efficient solution to the simultaneous localisation and map building (SLAM) problem. Working notes of ICRA'2000 Workshop W4: Mobile Robot Navigation and Mapping, April 2000.
- [6] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *IEEE Transactions on Robotics and Automation*, 7(6):859–865, 1991.
- [7] T. Edlinger and E. von Puttkamer. Exploration of an indoor-environment by an autonomous mobile robot. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 1994.
- [8] H.H. González-Baños and J.C. Latombe. Navigation strategies for exploring indoor environments. *International Journal of Robotics Research*, 2001.
- [9] J.-S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *Proc. of the International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, 2000.
- [10] D. Hähnel, D. Schulz, and W. Burgard. Map building with mobile robots in populated environments. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2002.
- [11] S. Koenig, C. Tovey, and W. Halliburton. Greedy mapping of terrain. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2001.
- [12] N. Massios, L. Dorst, and F. Voorbraak. A strategy for robot surveillancs using the geometrical structure of the environment,. In *IJCAI Workshop "Reasoning with Uncertainty in Robotics"*, 2001.
- [13] S.J. Moorehead, R. Simmons, and W.L. Whittaker. Autonomous exploration using multiple sources of information. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2001.
- [14] Hans P. Moravec and A.E. Elfes. High resolution maps from wide angle sonar. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 116–121, 1985.
- [15] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes. Coordination for multi-robot exploration and mapping. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2000.
- [16] C.J. Taylor and D.J. Kriegman. Exploration strategies for mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 248–253, 1993.
- [17] S. Thrun. An online mapping algorithm for teams of mobile robots. *International Journal of Robotics Research*, 2001.
- [18] S. Thrun. Learning occupancy grids with forward sensor models. *Autonomous Robots*, 2002. To appear.
- [19] P. Whaithe and F. P. Ferrie. Autonomous exploration: Driven by uncertainty. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(3):193–205, 1997.
- [20] B. Yamauchi, A. Schultz, and W. Adams. Integrating exploration and localization for mobile robots. *Adaptive Systems*, 7(2), 1999.

ROBOTICS

SIMULTANEOUS LOCALIZATION AND MAPPING

DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks

Austin Eliazar and Ronald Parr

Department of Computer Science

Duke University

{*eliazar, parr*}@cs.duke.edu

Abstract

We present a novel, laser range finder based algorithm for simultaneous localization and mapping (SLAM) for mobile robots. SLAM addresses the problem of constructing an accurate map in real time despite imperfect information about the robot's trajectory through the environment. Unlike other approaches that assume predetermined landmarks (and must deal with a resulting data-association problem) our algorithm is purely laser based. Our algorithm uses a particle filter to represent *both* robot poses and possible map configurations. By using a new map representation, which we call distributed particle (DP) mapping, we are able to maintain and update hundreds of candidate maps and robot poses efficiently. The worst-case complexity of our algorithm per laser sweep is log-quadratic in the number of particles we maintain and linear in the area swept out by the laser. However, in practice our run time is usually much less than that. Our technique contains essentially no assumptions about the environment yet it is accurate enough to close loops of 60m in length with crisp, perpendicular edges on corridors and minimal or no misalignment errors.

1 Introduction

The availability of relatively inexpensive laser range finders and the development of particle filter based algorithms have led to great strides in recent years on the problem of robot localization – determining a robot's position given a known map [Fox *et al.*, 1999]. Initially, the maps used for these methods were constructed by hand. However, the accuracy of the laser suggests its use for map-making as well as localization. Potential applications for accurate map-making would include search and rescue operations, as well as space, underwater and subterranean exploration.

Even with an accurate laser range finder, map-making presents a difficult challenge: A precise position estimate is required to make consistent updates to the the map, but a good map is required for reliable localization. The challenge of simultaneous localization and mapping (SLAM) is that of producing accurate maps in real time, based on a single pass over the sensor data, without an off line correction phase. Straight-forward approaches that localize the robot based upon a partial map and then update the map based upon the maximum likelihood position of the robot tend to produce maps with errors that accumulate over time. When the robot closes a phys-

ical loop in the environment, serious misalignment errors can result.

The EM algorithm provides a very principled approach to the problem, but it involves an expensive off-line alignment phase [Burgard *et al.*, 1999]. There exist heuristic approaches to this problem that fall short of full EM¹, but they are not a complete solution and they require additional passes over the sensor data [Thrun, 2001]. Scan matching can produce good maps [Lu and Milios, 1997; Gutmann and Konolige, 2000] from laser range finder data, but such approaches typically must explicitly look for and require additional effort to close loops. In varying degrees, these approaches can be viewed as partially separating the localization and mapping components of SLAM.

The FastSLAM algorithm [Montemerlo *et al.*, 2002], which does not require explicit loop-closing heuristics, is a recent SLAM approach which has made great progress in the field. FastSLAM follows a proposal by Murphy [Murphy, 1999] using a Rao-Blackwellized particle filter to sample robot poses and track the position of a fixed number of predetermined landmarks using a Kalman filter. (The landmark positions are conditionally independent given the robot pose.) This method mitigates some of the challenges in mapping at the expense of some challenges in landmark selection and identification. The latter can involve a fairly complicated data association problem, although recent progress has been made in addressing this [Montemerlo and Thrun, 2002].

We present a novel, laser range finder based algorithm called DP-SLAM that, like FastSLAM, exploits the conditional independences noted by Murphy. However, our algorithm is purely laser based and makes no landmark assumptions. We avoid the data association problem by storing multiple detailed maps instead of sparse landmarks, thereby subsuming association with localization. Our algorithm uses a particle filter to represent *both* robot poses *and* possible map configurations. Using a new map representation, which we call distributed particle (DP) mapping, we are able to maintain and update hundreds or thousands of candidate maps and robot poses in real time as the robot moves through the environment. The worst-case complexity of our algorithm per laser sweep is log-quadratic in the number of particles we

¹This approach is heuristic because it does not maintain a joint probability distribution over maps and poses.

maintain and linear in the area swept out by the laser. However, in practice our run time is usually much less. Our technique makes essentially no assumptions about the environment yet it is accurate enough to close loops of 60m in length with crisp, perpendicular edges on corridors and minimal or no misalignment errors. This accuracy is achieved throughout the mapping process, without a need for explicit algorithms to correct the loop as temporally distant observations begin to overlap. In fact, DP-SLAM could be further complimented by inclusion of existing algorithms for closing loops, though the addition may be unnecessary if a sufficient number of particles is used.

2 Particle Filters for Localization and Mapping

A particle filter is a simulation-based method of tracking a system with partially observable state. We briefly review particle filters here, but refer the reader to excellent overviews of this topic [Doucet *et al.*, 2001] and its application to robotics [Thrun, 2000] for a more complete discussion.

A particle filter maintains a weighted (and normalized) set of sampled states, $S = \{s_1 \dots s_m\}$, called *particles*. At each step, upon observing an observation o (or vector of observations), the particle filter:

1. Samples m new states $S' = \{s'_1 \dots s'_m\}$ from S with replacement.
2. Propagates each new state through a Markovian transition (or simulation) model: $P(s''|s')$. This entails sampling a new state from the conditional distribution over next states given the sampled previous state.
3. Weighs each new state according to a Markovian observation model: $P(o|s'')$
4. Normalizes the weights for the new set of states

Particle filters are easy to implement have been used to track multimodal distributions for many practical problems [Doucet *et al.*, 2001].

2.1 Particle Filters for Localization

A particle filter is a natural approach to the localization problem, where the robot pose is the hidden state to be tracked. The state transition is the robot's movement and the observations are the robot's sensor readings, all of which are noisy.

The change of the state over time is handled by a motion model. Usually, the motion indicated by the odometry is taken as the basis for the motion model, as it is a reliable measure of the amount that the wheels have turned. However, odometry is a notoriously inaccurate measure of actual robot motion, even in the best of environments. The slip and shift of the robot's wheels, and unevenness in the terrain can combine to give significant errors which will quickly accumulate. A motion model differs across robots and types of terrain, but generally consists of a linear shift, to account for systematic errors and Gaussian noise. Thus, for odometer changes of x , y and θ , a particle filter applies the error model and obtains, for particle i ,

$$x_i = a_x * x + b_x + \mathcal{N}(0, \sigma_x)$$

$$\begin{aligned} y_i &= a_y * y + b_y + \mathcal{N}(0, \sigma_y) \\ \theta_i &= a_\theta * \theta + b_\theta + \mathcal{N}(0, \sigma_\theta) \end{aligned}$$

The a and b terms are linear correction to account for consistent errors in motion. The function $\mathcal{N}(0, \sigma)$ returns random noise from a normal distribution with mean 0 and standard deviation σ , which is derived experimentally and may depend upon the magnitudes of x , y , and θ .

After simulation, we need to weight the particles based on the robot's current observations of the environment. For pure localization, the robot stores a map in memory. The position described by each particle corresponds to a distinct point and orientation in the map. Therefore, it is relatively simple to determine what values the sensors should return, given the pose within the map. The standard assumption is that sensor errors are normally distributed. Thus, if the first obstruction in the map along a line traced by a laser cast is at distance d and the reported distance is d' , the probability density of observing discrepancy $\delta = d' - d$, is normally distributed with mean 0. For our experiments we assumed a standard deviation in laser measurements of 5cm. Given the model and pose, each sensor reading is correctly treated as an independent observation [Murphy, 1999]. The total posterior for particle i is then

$$P_i = \prod_k P(\delta_{ik} | s_i, m),$$

where δ_{ik} is the difference between the expected and perceived distances for sensor (laser cast) k and particle i .

2.2 Particle Filters for SLAM

Some approaches for SLAM using particle filters attempt to maintain a single map with multiple robot poses [Thrun, 2001], an approach that we avoid because it leads to errors that accumulate over time. The basic problem is that the hidden state is actually both the robot pose and the map itself. An important consequence of this problem is that all observations are no longer compared to a single map, which is presumed to be correct. Instead, the observations are compared against an incomplete and possibly incorrect map, identified with the particle in question. The map itself is created by the accumulation of observations of the environment and estimates of robot positions.

In principle, this "solves" the SLAM problem. In practice, it replaces a conceptual problem with an algorithmic one. Particles with errors in robot position estimates will make erroneous additions to their maps. An error in the map will then, in turn, cause another error in localization in the next step, and these inaccuracies can quickly compound. Thus, the number of particles required for SLAM is typically more than that required for localization since the price of accumulated errors is much higher. Note that in pure localization, small errors in position estimation can be absorbed as part of the noise in the motion model.

The algorithmic problem becomes one of efficiently maintaining a large enough set of particles to obtain robust performance, where each particle is not merely a robot pose, but a pose and map. Since maps are not light weight data structures, maintaining the hundreds or thousands of such maps poses a serious challenge. One reasonable approach to taming this problem is to assume that the uncertainty in the map

can be represented in a simple parametric form. This is essentially the approach taken by FastSLAM, for which the map is a Kalman filter over a set of landmark positions. This is certainly the right thing to do if one is given a set of landmarks that can be quickly and unambiguously identified. We will show that this strong assumption is not required: By using raw laser data, combined with an occupancy grid and efficient data structures, we can handle a large number of candidate maps and poses efficiently, achieving robust performance.

3 DP-SLAM

In this section we motivate and present the main technical contribution of the DP-SLAM algorithm. DP-SLAM implements what is essentially a simple particle filter over maps and robot poses. However, it uses a technique called distributed particle mapping (DP-Mapping), which enables it to maintain a large number of maps very efficiently.

3.1 Naive SLAM

When using a particle filter for SLAM, each particle corresponds to a specified trajectory through the environment and has a specific map associated with it. When a particle is resampled, the entire map itself is treated as part of the hidden state that is being tracked and is copied over to the new particle. If the map is an occupancy grid of size M and P particles are maintained by the particle filter, then ignoring the cost of localization, $O(MP)$ operations must be performed merely copying maps. For a number of particles sufficient to achieve precise localization in a reasonably sized environment, the naive approach would require gigabytes worth of data movement per update².

3.2 Distributed Particle Mapping

By now the astute reader has most likely observed that the naive approach is doing too much work. To make this clearer, we will introduce the notion of a particle ancestry. When a particle is sampled at iteration i to produce a successor particle at iteration $i + 1$, we call the generation i particle a *parent* and the generation $i + 1$ particle a *child*. Two children with the same parent are *siblings*. From here, the concept of a particle ancestry extends naturally. Suppose the laser sweeps out an area of size $A \ll M$ and consider two siblings, s_1 and s_2 . Each sibling will correspond to a different robot pose and will make at most A updates to the map it inherits from its parent. Thus, s_1 and s_2 can differ in at most A map positions.

When the problem is presented in this manner, the natural reaction from most computer scientists is to propose recording the “diff” between maps, i.e., recording a list of changes that each particle makes to its parent’s map. While this would solve the problem of making efficient map updates, it would create a bad computational problem for localization: Tracing a line through the map to look for an obstacle would require working through the current particle’s entire ancestry and consulting the stored list of differences for each particle in the ancestry. The complexity of this operation would be

²In addition to the theoretical analysis, anecdotal comments made by researchers in this area reinforce the impracticality of this approach.

linear in the number of iterations of the particle filter. The challenge is, therefore, to provide data structures that permit efficient updates to the map *and* efficient localization queries with time complexity that is independent of the number of iterations of the particle filter. We call our solution to this problem *Distributed Particle Mapping* or *DP-Mapping*, and we explain it in terms of the two data structures that are maintained: the ancestry tree and the map itself.

Maintaining the particle ancestry tree

The basic idea of the particle ancestry tree is fairly straightforward. The tree itself is rooted with an initial particle, of which all other particles are progeny. Each particle maintains a pointer to its parent and is assigned a unique numerical ID. Finally each particle maintains a list of grid squares that it has updated.

The details of how we will use the ancestry tree for localization are described in the subsequent section. In this section we focus on the maintenance of the ancestry tree, specifically on making certain that the tree has bounded size regardless of the number of iterations of the particle filter.

We maintain a bounded size tree by pruning away unnecessary nodes. First, note that certain particles may not have children and can simply be removed from the tree. Of course, the removal of such a particle may leave its parent without children as well, and we can recursively prune away dead branches of the tree. After pruning, it is obvious that the only particles which are stored in our ancestry tree are exactly those particles which are ancestors of the current generation of particles.

This is still somewhat more information than we need to remember. If a particle has only one child in our ancestry tree, we can essentially remove it, by collapsing that branch of the tree. This has the effect of merging the parent’s and child’s updates to the map, a process described in the subsequent section. By applying this process to the entire tree after pruning, we obtain a *minimal* ancestry tree, which has several desirable and easily provable properties:

Proposition 3.1 *Independent of the number of iterations of particle filtering, a minimal ancestry tree of P particles*

1. *has exactly P leaves,*
2. *has branching factor of at least 2, and*
3. *has depth no more than P .*

Map representation

The challenge for our map representation is to devise a data structure that permits efficient updates and efficient localization. The naive approach of a complete map for each particle is inefficient, while the somewhat less naive approach of simply maintaining history of each particle’s updates is also inefficient because it introduces a dependency on the number of iterations of the particle filter.

Our solution to the map representation problem is to associate particles with maps, instead of associating maps with particles. DP-mapping maintains just a single occupancy grid. (The particles are distributed over the map.) Unlike a traditional occupancy grid, each grid square stores a balanced tree, such as a red-black tree. The tree is keyed on the IDs of

the particles that have made changes to the occupancy of the square.

The grid is initialized as a matrix of empty trees. When a particle makes an observation about a grid square it inserts its ID and the observation into the associated tree. Notice that this method of recording maps actually allows each particle to behave as if it has its own map. To check the value of a grid square, the particle checks each of its ancestors to find the most recent one that made an observation for that square. If no ancestor made an entry, then the particle can treat this position as being unknown.

We can now describe the effects of collapsing an ancestor with a single child in the ancestry tree more precisely: First, the set of squares updated by the child is merged into the parent's set. Second, for each square visited by the child, we change the ID key stored in the balanced tree to match that of the parent. (If both the child and parent have made an update to the same square, the parent's update is replaced with the child's.) The child is then removed from the tree and the parent's grandchildren become its direct children. Note that this ensures that the number of items stored in the balanced tree at each grid square is $O(P)$.

3.3 Computational Complexity

The one nice thing about the naive approach of keeping a complete map for each particle is the simplicity: If we ignore the cost of block copying maps, lookups and changes to the map can all be done in constant time. In these areas, distributed particle mapping may initially seem less efficient. However, we can show that DP maps are in fact asymptotically superior to the naive approach.

Lookup on a DP-map requires a comparison between the ancestry of a particle with the balanced tree at that grid square. Let D be the depth of the ancestry tree, and thus is the maximum length of a particle's ancestry. Strictly speaking, as the ancestry tree is not guaranteed to be balanced, D can be $O(P)$. However, in practice, this is almost never the case, and we have found $D \approx O(\lg P)$, as the nature of particle resampling lends to very balanced ancestry trees. (Please see the discussion in the following section for more detail on this point.) Therefore, we can complete our lookup after just D accesses to the balanced tree. Since the balanced tree itself can hold at most P entries, and a single search takes $O(\lg P)$ time. Accessing a specific grid square in the map can therefore be done in $O(D \lg P)$ time.

For localization, each particle will need to make $O(A)$ accesses to the map. As each particle needs to access the entire observed space for its own map, we need $O(AP)$ accesses, giving localization with DP-maps a complexity of $O(AD \lg P)$.

To complete the analysis we must handle two remaining details: The cost of inserting new information into the map, and the cost of maintaining the ancestry tree. Since we use a balanced tree for each grid square, insertions and deletions on our map both take $O(\lg P)$ per entry. Each particle can make at most $O(A)$ new entries, which in turn will only need to be removed once. Thus the procedure of adding new entries can be accomplished in $O(AD \lg P)$ per particle, or $O(AD \lg P)$

total and the cost of deleting childless particles will be amortized as $O(AD \lg P)$.

It remains to be shown that the housekeeping required to maintain the ancestry tree has reasonable cost. Specifically, we need to show that the cost of collapsing childless ancestry tree nodes does not exceed $O(AD \lg P)$. This may not be obvious at first, since successive collapsing operations can make the set of updated squares for a node in the ancestry tree as large as the entire map. We now argue that the amortized cost of these changes will be $O(AD \lg P)$. First, consider the cost of merging the child's list of modified squares into the parent's list. If the child has modified n squares, we must perform $O(n \lg P)$ operations (n balanced tree queries on the parent's key) to check the child's entries against the parent's for duplicates.

The final step that is required consists of updating the ID for all of the child's map entries. This is accomplished by deleting the old ID, and inserting a new copy of it, with the parent's ID. The cost of this is again $O(n \lg P)$. Consider that each map entry stored in the particle ancestry tree has a potential of D steps that it can be collapsed, since D is the total number of nodes between its initial position and the root, and no new nodes will ever be added in between. At each iteration, P particles each create A new map entries with potential D . Thus the total potential at each iteration is $O(AD \lg P)$.

The computational complexity of DP-SLAM can be summarized as follows:

Proposition 3.2 *For a particle filter that maintains P particles, laser that sweeps out A grid squares, and an ancestry tree of depth D , DP-SLAM requires:*

- $O(AD \lg P)$ operations for localization arising from:
 - P particles checking A grid squares
 - A lookup cost of $O(D \lg P)$ per grid square
- $O(AP \lg P)$ operations to insert new data into the tree, arising from:
 - P particles inserting information at A grid squares
 - Insertion cost of $O(\lg P)$ per new piece of information
- Ancestry tree maintenance with amortized cost $O(AD \lg P)$ arising from
 - A cost of $O(\lg P)$ to remove an observation or move it up one level in the ancestry tree
 - A maximum potential of ADP introduced at each iteration.

3.4 Complexity Comparison

Our analysis shows that the amortized complexity of DP-SLAM is $O(AD \lg P)$, which can be as large as $O(AP^2 \lg P)$. For the naive approach, each map is represented explicitly, so lookups can be done in constant time per grid square. The localization step thus takes only $O(AP)$. Without the need for updates to an ancestry tree, map updates can likewise be done in $O(AP)$ time. However, the main bulk of computation lies in the need to copy over an entire map for each particle, which will require $O(MP)$ time, where M is

the size of the map. Since typically $M \gg A$, this obviously is the dominant term in the computation.

DP-SLAM will be advantageous when $ADlgP < M$. Even in the worst case where D approaches P , there will still be a benefit. For a high resolution occupancy grid, M will be quite large since it grows quadratically with the linear measure of the environment and would grow cubically if we consider height. Moreover, A , will be a tiny fraction of M . The size of $DlgP$ will, of course, depend upon the environmental features and the sampling rate of the data. It is important to note that since the time complexity of our algorithm does not depend directly on the size of the map, there will necessarily exist environments for which DP-SLAM is vastly superior to the naive approach since the naive approach will require block copying an amount of data will exceed physical memory.

In our initial experiments, we have observed that P was surprisingly small, suggesting that in practice the advantage of DP-SLAM is much greater than the worst-case analysis suggests. For some problems the point at which it is advantageous to use DP-SLAM may be closer to $Alg^2P < M$. This phenomenon is discussed in more detail in the subsequent section.

4 Empirical Results

We implemented and deployed the DP-SLAM algorithm on a real robot using data collected from the second floor of our computer science building. In our initial implementation our map representation is a binary occupancy grid. When a particle detects an obstruction, the corresponding point in the occupancy grid is marked as fully occupied and it remains occupied for all successors of the particle. (A probabilistic approach that updates the grid is a natural avenue for future work.)

On a fast PC (2.4 GHz Pentium 4), the run time of DP-SLAM is close to that of the data collection time, so the algorithm could have been run in real time for our test domains. In practice, however, we collected our data in a log file using the relatively slow computer on our robot, and processed the data later using a faster machine. To speed things up, we also implemented a novel particle culling technique to avoid fully evaluating the posterior for bad particles that are unlikely to be resampled. This works by dividing the sensor readings for each particle into k disjoint, evenly distributed subsets. The posterior for the particles is computed in k passes, where pass i evaluates the contribution from subset i . At the end of each pass, particles with significantly lower (partially evaluated) posterior in comparison to the others are assigned 0 probability and removed from further consideration. We also set a hard threshold on the number of particles we allowed the algorithm to keep after culling. Typically, this was set to the top 10% of the total number of particles considered. In practice, this cutoff was almost never used since culling effectively removed at least 90% of the particles that were proposed. A value if $k = 6$ gave us a speedup of approximately $6\times$.

For the results we present, it is important to emphasize that our algorithm knows *absolutely nothing* about the environment or the existence of loops. No assumptions about the en-

vironment are made, and no attempt is made to smooth over errors in the map when loops are closed. The precision in our maps results directly from the robustness of maintaining multiple maps.

4.1 Robot and Robot Model

The robot we used for testing DP-SLAM is an iRobot ATRV Jr. equipped with a SICK laser range finder attached to the front of the robot at a height of 7cm. Readings are made across 180° , spaced one degree apart, with an effective distance of up to 8m. The error in distance readings is typically less than 5mm.

Odometric readings from the ATRV Jr.'s shaft encoders are unreliable, particularly when turning. Our motion model assumes errors are distributed in a Gaussian manner, with a standard deviation of 25% in lateral motion and 50% in turns. Turns are quasi-holonomic, performed by skid steering. We obtained our motion model using an automated calibration procedure that worked by positioning the robot against a smooth wall and comparing odometry data with the movement inferred from the laser readings.

4.2 Test Domain(s) and Map(s)

We tested the algorithm on a loop of hallway approximately 16m by 14m. A new set of observations was recorded after each 20cm motion (approximately). The maps were constructed with a resolution of 3cm per grid square, providing a very high resolution cross section of the world at a height of 7cm from the floor.

Figure 1 shows the highest probability map generated after the completion of one such loop using 9000 particles³. In this test, the robot began in the middle of the bottom hallway and continued counterclockwise through the rest of the map, returning to the starting point.

This example demonstrates the accuracy of DP-SLAM when completing a long loop, one of the more difficult tasks for SLAM algorithms. After traveling 60m, the robot is once again able to observe the same section of hallway in which it started. At that point, any accumulated error will readily become apparent, as it will lead to obvious misalignments in the corridor. As the figure shows, the loop was closed perfectly, with no discernible misalignment.

To underscore the advantages of maintaining multiple maps, we have included the results obtained when using a single map and the same number of particles. Figure 2 shows the result of processing the same sensor log file by generating 9000 particles at each time step, keeping the single particle with the highest posterior, and updating the map based upon the robot's pose in this particle. There is a considerable misalignment error where the loop is closed at the top of the map.

Larger, annotated versions of the maps shown here, as well as maps generated from different sensor logs, will be available at <http://www.cs.duke.edu/~parr/dpslam/>.

³In an earlier version of this paper, we presented a map produced with 1000 particles based upon a different sensor log file. The run shown here covers the same area, but has less reliable odometry, probably due to weaker battery, and is more illustrative of the benefits of DP-SLAM.

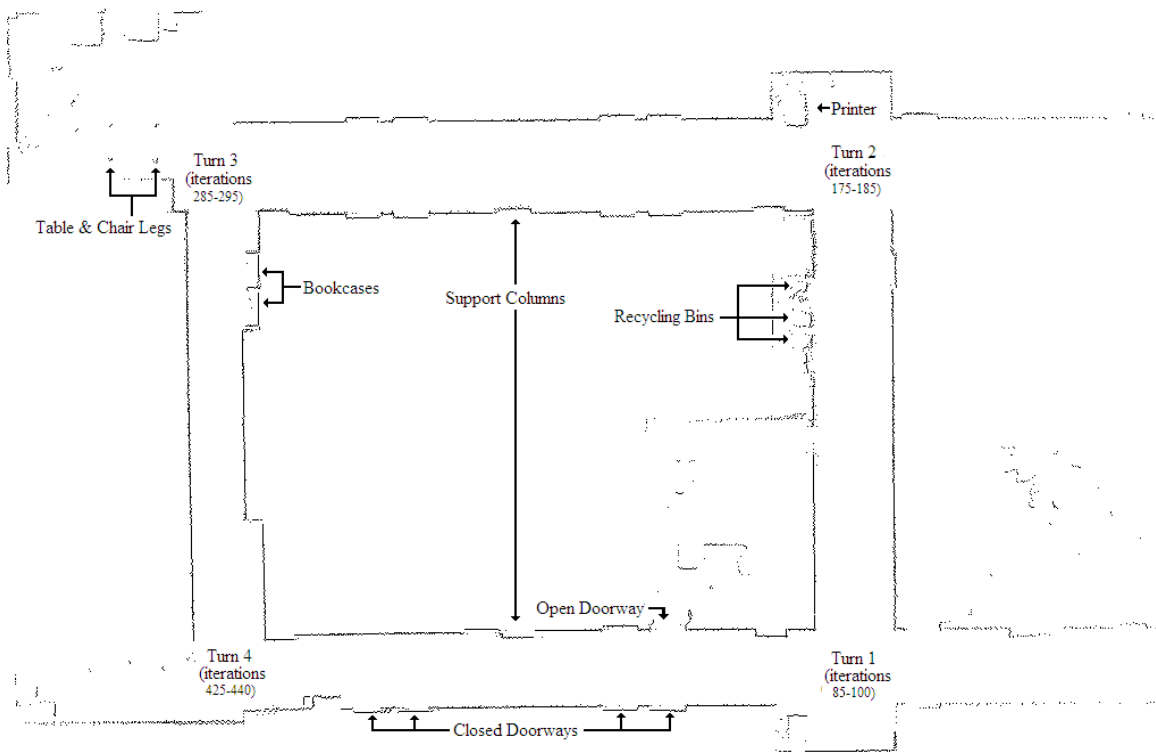


Figure 1: A DP-SLAM map with 9000 particles.

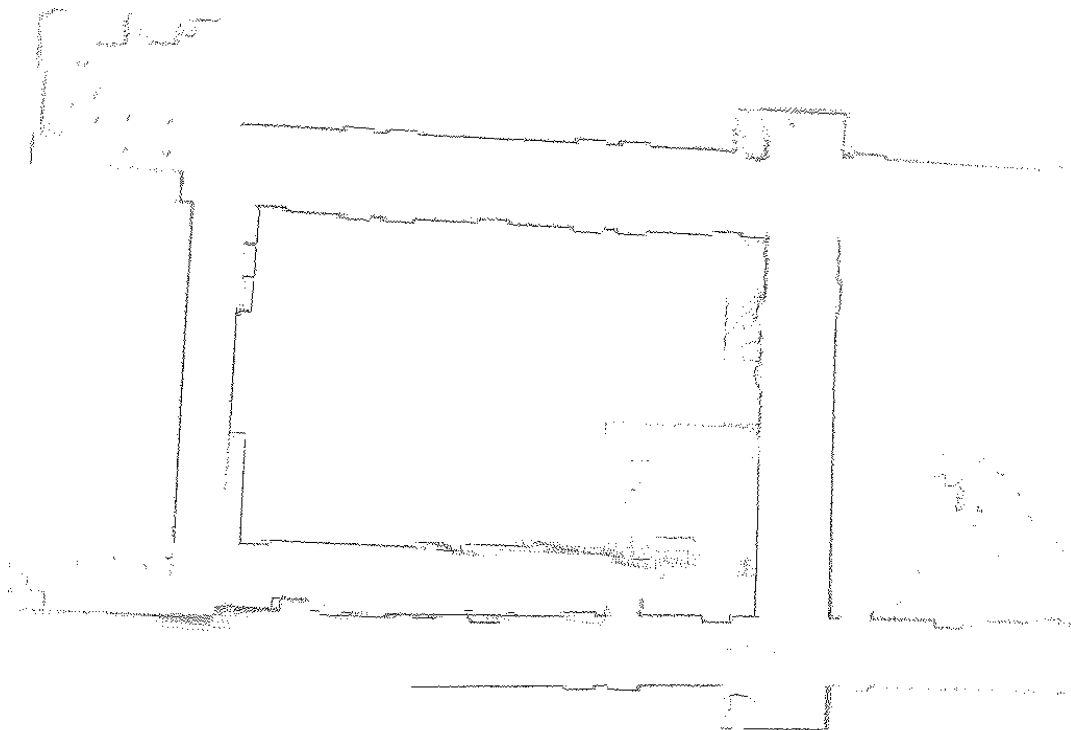


Figure 2: SLAM using a single map.

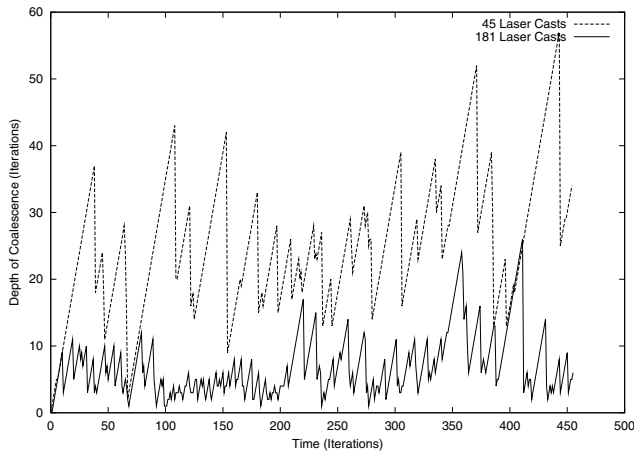


Figure 3: Depth of coalescence as a function of time.

4.3 Coalescence

In our sample domain (the second floor of the Duke University Department of Computer Science), we have found that DP-SLAM produces very shallow ancestry trees due to a phenomenon we refer to as *coalescence*. Over a number iterations of the particle filter, nearly all of the particles will have a common ancestor particle in the not-too-distant past. This point of coalescence, or common ancestry, varies over time, and is sensitive to the environment. Observations made on our empirical results indicate that while this number peaked during events of higher uncertainty, coalescence was often as recent as 20 generations in the past, and never exceeded 90. This means that beyond about 20 steps in the past, every particle has an identical map. The fact that we can close loops so precisely while maintaining relatively shallow coalescence points suggests that we are able to maintain distributions over map regions long enough to resolve ambiguities, but not longer than is needed. While the phenomenon deserves more careful study, our initial impression is that by the time a region has passed outside of the robot’s field of view, it has been scanned enough times by the laser that there is only a single hypothesis that is consistent with the accumulated observations.

Additional experiments were run to show the relationship between coalescence and uncertainty, as well as the ability of DP-SLAM to automatically preserve more information when required. We considered two different scenarios: the original DP-SLAM algorithm with the same sensor data used to generate Figure 1, and a handicapped version of DP-SLAM run on the same sensor log, but with three out of every four laser casts ignored at each iteration. The handicapped version required 30,000 particles to produce good maps.

To study the effects of uncertainty on coalescence, we ran both algorithms with the same number of particles and then compared the coalescence points. The results in Figure 3 show how the two versions of DP-SLAM performed on the same sensor log with the same number of particles. The decrease in observational information for the handicapped version of DP-SLAM makes it more difficult for the robot to be

certain about its position. This is reflected in the higher peaks in the graph, and the less recent point of coalescence overall.

In both cases, strong correlation can be seen between the higher peaks in this graph and the turns that the robot took in the map. (Note the iteration number annotations at the turns in Figure 1.) This is because odometry is particularly unreliable during skid turns and the robot has greater difficulty localizing. The points at which our turns occur in our office environment also tend to produce more ambiguous sensor data since turns are typically in open areas where the laser scans will be more widely spaced. Other peaks can be mapped to areas of clutter in the map, such as the recycling bins.

Figure 4 was created from the same two experiments, and tracks the amount of memory used over time, as measured by the total number of leaves in the balanced trees stored in our occupancy grid. Since no leaves are stored for empty space and our maps are essentially two one-dimensional surfaces, the amount of memory required for leaves should grow roughly linearly over time. The peaks corresponding to the turns in the map are even more pronounced for the handicapped version of DP-SLAM in this graph, indicating that under conditions of greater uncertainty, the ancestry tree grows in width as well as depth.

Of course, shallow coalescence may not occur and may not even be desirable in all environments, but it has significant implications for the efficiency of the algorithm when it does occur. It implies $D \ll P$, making our $O(ADPlgP)$ bound closer to $O(APlg^2P)$ in practice. Moreover, we get maximum benefit from the compactness of our map representation. As the particles coalesce, irrelevant entries are removed. Thus all areas in the map that were observed before the point of coalescence contain only one entry in their list. Similarly, those areas which have not been observed by any particle yet have no entries in their list. If we can assume that the depth of coalescence is bounded by some reasonable constant C , then all map squares but those that have been altered prior the point of coalescence can be thought of as requiring a constant amount of storage. This means that the total space required to store the map tends to be close to $O(M + APC)$ in practice, not the theoretical worst case of $O(MP)$.

A sceptic may wonder if coalescence is indeed a desirable thing since it may imply a loss of information. If the algorithm maintains only a limited window of uncertainty, how can it be expected to close large loops, which would seem to require maintaining uncertainty over map entries for arbitrarily long durations? The simple answer is that coalescence is desirable when it is warranted by the data and undesirable when the accumulation of sensor data is insufficient to resolve ambiguities. In fact, coalescence is a direct product of the amount of uncertainty still present in the observations. Our algorithm is both able to maintain ambiguity for arbitrary time periods and benefit from coalescence when it is warranted, so there is nothing negative about the fact that we have observed this phenomenon in our experiments.

Of course, since we are representing a complicated distribution with a finite number of samples, there is always a danger that the low probability “tails” of the distribution will be lost if too few particles are used. This would result in premature coalescence due to failure to resample a potentially

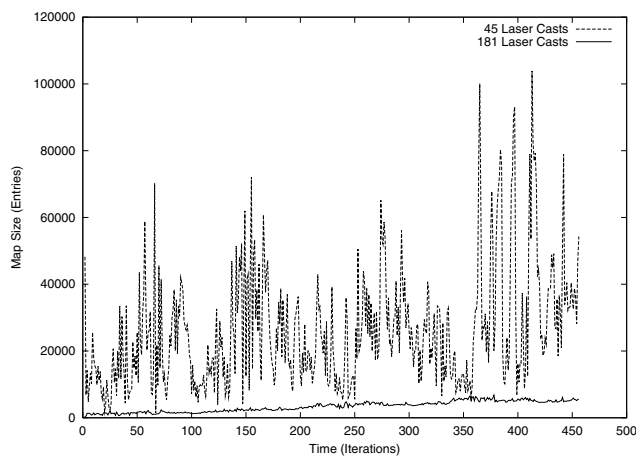


Figure 4: Total map entries used over time.

useful and correct map. There will certainly be some environments, typically those with very sparse sensor data, where it will be difficult to avoid premature coalescence without an excessive number of particles. In situations with sparse data, a parametric representation of uncertainty over landmarks may be more tractable and more appropriate.

5 Conclusion and Future Work

We have presented a novel particle-filter based algorithm for simultaneous localization and mapping. Using very efficient data structures, this algorithm is able to maintain thousands of maps in real time, providing robust and highly accurate mapping. The algorithm does not assume the presence of predetermined landmarks or assume away the data association problem created by the use of landmarks. Our algorithm has been deployed on a real robot, where it has produced highly detailed maps of an office environment.

We believe that this is the first time this level of accuracy has been achieved for the type of data we consider using an algorithm that does not explicitly attempt to close loops and that has no domain specific knowledge. Nevertheless, our algorithm does have some limitations. Most reasonably priced laser range finders scan at a fixed height, giving us an incomplete view of the environment. Our current map representation is very simplistic - a grid where each cell is presumed to be completely clear or completely opaque to the laser. This obviously creates discretization errors at the edges of objects and confusion for very small objects. Our small cell size reduces this problem, but does not eliminate it entirely. For example, power cords hanging off of the edge of desks tend to introduce localization errors and increase the number of particles needed to get robust performance.

There is a logical inconsistency in the way we treat the laser data. When we add new information to the map, we treat the laser like a deterministic device, but when we localize, we treat it like a noisy device with a standard deviation in measurement that can span several grid squares. Exaggerated error models like this are common in many real-world applications of statistical models, where additional noise is added

to one part of the model to compensation for unmodeled noise in other parts. Nevertheless, in future work we would like to develop a more comprehensive error model and a principled scheme for updating the map that handles partially or transiently occupied grid squares. Naturally, we would like to consider other map representation methods too.

The most important future direction for this work is to incorporate less reliable sensors as such video or sonar to construct a more comprehensive, three-dimensional view of the environment.

Acknowledgments

We are very grateful to Sebastian Thrun for critical feedback and encouragement for this line of research. We also thank Tammy Bailey, Dieter Fox, Carlos Guestrin, Dirk Haehnel, Mark Paskin, and Carlo Tomasi for helpful feedback and comments.

References

- [Burgard *et al.*, 1999] W. Burgard, D. Fox, H. Jans, C. Matenar, and S. Thrun. Sonar-based mapping with mobile robots using EM. In *Proc. of the International Conference on Machine Learning*, 1999.
- [Doucet *et al.*, 2001] Arnaud Doucet, Nando de Freitas, and Neil Gordon. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, Berlin, 2001.
- [Fox *et al.*, 1999] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *AAAI-99*, 1999.
- [Gutmann and Konolige, 2000] J. Gutmann and K. Konolige. Incremental mapping of large cyclic environments, 2000.
- [Lu and Milios, 1997] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping, 1997.
- [Montemerlo and Thrun, 2002] M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using FastSLAM. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2002.
- [Montemerlo *et al.*, 2002] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *AAAI-02*, Edmonton, Canada, 2002. AAAI.
- [Murphy, 1999] K. Murphy. Bayesian map learning in dynamic environments. In *Advances in Neural Information Processing Systems 11*. MIT Press, 1999.
- [Thrun, 2000] S. Thrun. Probabilistic algorithms in robotics. *AI Magazine*, 21(4):93–109, 2000.
- [Thrun, 2001] S. Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *International Journal of Robotics Research*, 20(5):335–363, 2001.

Consistent, Convergent, and Constant-Time SLAM*

J. Leonard and P. Newman[†]

Abstract

This paper presents a new efficient algorithm for simultaneous localization and mapping (SLAM), using multiple overlapping submaps, each built with respect to a local frame of reference defined by one of the features in the submap. The global position of each submap is estimated using information from other submaps in an efficient, provably consistent manner. For situations where the mobile robot is able to make repeated visits to all regions of the environment, the method achieves convergence to a near-optimal result with $\mathcal{O}(1)$ time complexity while maintaining consistent error bounds. Simulation results demonstrate the ability of the technique to converge to errors that are only slightly greater than the full solution, while maintaining consistency.

1 Introduction

The capability of simultaneous localization and mapping (SLAM) is considered vital for the creation of long-lived autonomous mobile agents. Because of the difficulties encountered by SLAM algorithms when applied to larger environments, the “map scaling” problem has been identified as one of the key issues for research in this area. In this paper, we adopt the feature-based, state-space formulation of SLAM [Smith *et al.*, 1987]. Recent related work in SLAM includes submap decomposition methods [Leonard and Feder, 2000; Guivant and Nebot, 2001; Julier and Uhlmann, 2001; Williams *et al.*, 2002; Tardós *et al.*, 2002], FastSLAM [Montemerlo *et al.*, 2002], sparse extended information filters (SEIF’s) [Thrun *et al.*, 2002], scan-matching [Gutmann and Konolige, 1999; Thrun, 2001; Hahnel *et al.*, 2002] and topological approaches [Kuipers and Beeson, 2002; Choset and Nagatani, 2001].

*This research has been funded in part by NSF Career Award BES-9733040, the MIT Sea Grant College Program under grant NA86RG0074, and the Office of Naval Research under grant N00014-97-0202.

[†]P. Newman is at the Robotics Research Group, University of Oxford and J. Leonard is at the Department of Ocean Engineering at MIT, email: pnewman@robots.ox.ac.uk, jleonard@mit.edu

This paper examines the SLAM problem using submaps. While there has been a considerable amount of work in SLAM along these lines, no previous method satisfies each of the three criteria of (1) provable consistency, (2) spatial convergence, and (3) constant-time updates. For example, Julier and Uhlmann [2001] provide a consistent, constant-time algorithm for large-scale SLAM, based on split covariance intersection, but this method does not achieve “tight” convergence to the error bounds that would be obtained with a full covariance solution. Methods such as the compressed filter [Guivant and Nebot, 2001], the constrained local submap filter [Williams *et al.*, 2002], and sequential map joining [Tardós *et al.*, 2002] are provably consistent and convergent, but are $\mathcal{O}(n^2)$, where n is the number of the features in the environment. Other techniques such as decoupled stochastic mapping (DSM) [Leonard and Feder, 2000] and SEIF’s [Thrun *et al.*, 2002] achieve $\mathcal{O}(1)$ performance, but make approximations that require empirical testing to verify state estimation consistency. The Atlas framework for large-scale SLAM [Bosse *et al.*, 2003] achieves constant-time performance during the motion of the robot, enabling closing of large loops, but does not compute state estimates with respect to a single global reference frame.

The FastSLAM technique stands out in the literature as perhaps the only published technique which has been successfully posed for the general nonlinear SLAM problem with computational effort $\mathcal{O}(\log(n))$. The performance of FastSLAM, however, depends linearly on a “parameter (the number of particles), whose scaling with environmental size is still poorly understood [Thrun *et al.*, 2002]”.

This paper considers SLAM with known data association. This is a major assumption, but it enables us to focus on the underlying structure of SLAM state estimation with submaps for the linear Gaussian case. Other research has shown the ability to perform feature detection and measurement association using techniques such as random sample consensus and Joint Compatibility Testing [Neira and Tardós, 2001].

In this paper, we demonstrate how the three criteria of consistency, convergence, and constant-time updates can be achieved with multiple, locally referenced submaps. The key idea is to decouple the estimate of a map’s global location from any of the state estimates within the map. This “inside-out” decomposition strategy succeeds because state estimates from one map are never mixed with state estimates from other

maps. This enables us to guarantee the consistency of the state estimates for the linear Gaussian case.

The structure of this paper is as follows. After reviewing the full (single map) SLAM problem in Section 2, we define our terminology in Section 3. Section 4 summarizes the new SLAM algorithm and Section 5 discusses its consistency and convergence properties. Section 6 describes the performance of the algorithm using simulations and Section 7 provides a concluding discussion.

2 SLAM within a single map

Let us assume that there are n features in the environment, and that they are static. The global frame, designated by G , is a unique, immutable coordinate-frame that is defined at the beginning of a mission. The true state at time k is designated by $\mathbf{x}(k) = [\mathbf{x}_v(k)^T \ \mathbf{x}_1(k)^T \ \dots \ \mathbf{x}_n(k)^T]^T$, where $\mathbf{x}_v(k)$ represent the location of the vehicle, and $[\mathbf{x}_1(k)^T \ \dots \ \mathbf{x}_n(k)^T]^T$ represents the locations of the environmental features. We assume that the vehicle moves from time k to time $k+1$ in response to a known control input, $\mathbf{u}(k)$, that is corrupted by noise. Let U^k designate the set of all control inputs from time 0 through time k , $Z(k)$ designate the set of sensor measurements obtained at time k , and Z^k designate the set of all measurements obtained from time 0 through time k . For each measurement $\mathbf{z}_j(k) \in Z(k)$, there is a corresponding assignment index $\mathbf{a}_j$. The value of $\mathbf{a}_j$ is i if measurement $\mathbf{z}_j(k)$ originates from feature i . Let A^k designate the set of all assignment indices from time 0 through time k . Assuming that the associations are known, the objective is to compute recursively the probability distribution for the location of the robot and the features, with reference to the global reference frame G , given the measurements, control inputs, and assignments:

$$p(\mathbf{x}_v(k), \mathbf{x}_1(k), \dots, \mathbf{x}_{n_k}(k) | Z^k, A^k, U^{k-1}). \quad (1)$$

For the Linear-Gaussian (LG) SLAM problem, the Kalman filter provides the optimal estimate of this pdf, which is described by its mean $[\hat{\mathbf{x}}_v(k)^T \ \hat{\mathbf{x}}_1(k)^T \ \dots \ \hat{\mathbf{x}}_n(k)^T]^T$ and covariance $\mathbf{P}(k)$. The properties of single-map LG SLAM solution are well-known [Dissanayake *et al.*, 2001].

3 Machinery for SLAM using multiple maps

We now proceed to define several terms and basic operations that will facilitate description of the new method. A **location vector** is a parameterization of both position and orientation of one coordinate-frame, i with respect to another, j . In $\mathcal{R}^2$ this is represented as a translation by $[x, y]$ followed by a rotation θ . These three parameters are encapsulated in the 3 vector $T_i^j = [x, y, \theta]^T$.

An **entity** is a parameterization of a vehicle or landmark. Each entity is labelled with a unique positive integer — this is referred to as the entity's ID. We can attach a coordinate frame $\mathcal{F}_i$ to any entity i and describe it using a location vector in another coordinate frame. The vector T_i^j should be understood to be a parameterization of a transformation from $\mathcal{F}_i$ to $\mathcal{F}_j$. The uncertainty in this transformation is represented by Σ_i^j .

A **map** is a collection of entities all described with respect to a local coordinate frame. Each map has a unique integer

id. Each map has associated with it a **Map Root** entity i and a **Map Location** vector T_m^G . The Map Location vector describes the pose of a map's local coordinate frame in the global frame G . The local coordinate frame of a map is coincident with one of the entities in the local map; this entity is referred to as the Map Root. In other words, the Map Root lies at the origin of the local map, and it is the entity to which all other entities in a map are referenced. If an entity is the Map Root, then by definition its location vector will be $[0, 0, 0]^T$ and its global location vector given by T_m^G — the location of the map in global coordinates.¹ We use the pre-superscript notation $[m, j]$ to denote that all entities in map m are referenced to $\mathcal{F}_j$ where j is the root entity of the map.

The notation is summarized as:

$$[\text{mapid}, \text{rootid}] \text{Variable}_{\text{variable id}}^{\text{w.r.t. id}}$$

Using this notation we can write the transformation from $\mathcal{F}_j$ to $\mathcal{F}_k$ in map m with root entity i as $[m, i]T_k^j$. We simplify notation by dropping the right superscript when describing a transformation with respect to the map root:

$$[m, i]T_j^i \rightarrow [m, i]T_j$$

The term $[m, i]T_j$ is the pose of an entity j in the local frame of the m^{th} map (which has its Map Root as entity i).

We can manipulate location vectors using the binary transformation operator $\oplus$ and the unary operator $\ominus$ where

$$\begin{aligned} T_j^i &= \ominus T_i^j \\ T_k^i &= T_j^i \oplus T_k^j \end{aligned}$$

Using the notation described above, we can describe the relationship between entities in any given map. Consider the case of map m referenced to feature i denoted $[m, i]$. Taking two entities j and k from m , we can express the transformation from j to k as

$$\begin{aligned} [m, i]T_k^j &= [m, i]T_i^j \oplus [m, i]T_k^i \\ &= \ominus [m, i]T_j^i \oplus [m, i]T_k^i \\ &= \ominus [m, i]T_j \oplus [m, i]T_k \end{aligned} \quad (2)$$

where the last step uses the simplification in notation described above.

We define **root-shifting** to be the operation, $\mathcal{S}$, that changes the root of a map from i to j . After this operation all location vectors in a map will be referenced to $\mathcal{F}_j$ rather than $\mathcal{F}_i$, $i, j \in m$. This operation is simply an extension of Equation 2 to act on all entities in the map:

$$[m, j]T_{1:n} = \mathcal{S}_{i \rightarrow j}([m, i]T_{1:n}) \quad (3)$$

$$= \begin{bmatrix} \ominus [m, i]T_j \oplus [m, i]T_1 \\ \vdots \\ \ominus [m, i]T_j \oplus [m, i]T_n \end{bmatrix} \quad (4)$$

¹In the SLAM literature, the term “base reference” [Tardós *et al.*, 2002] is a synonym for our term Map Root. Note that in the general case with orientation, a single point feature will be insufficient to define a reference frame. In 2-D, two points will be required and in 3-D three points will be required.

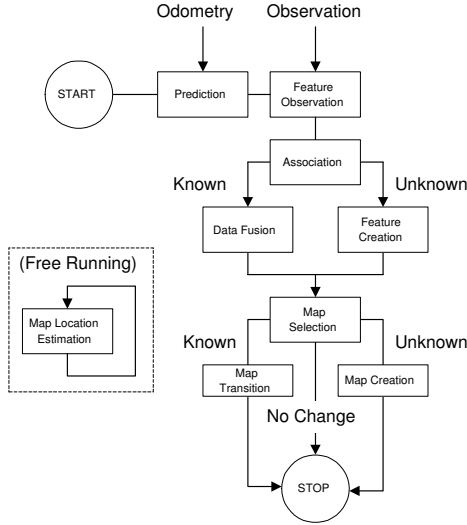


Figure 1: Flow chart for each cycle of the algorithm.

The global location of a feature j in a local map $^{[m,i]}T_j$ is computed by simple composition of the local location vector and the already globally referenced Map Location vector:

$$^{[m,\cdot]}T_j^G = T_m^G \oplus ^{[m,i]}T_j \quad (5)$$

4 The constant-time SLAM algorithm

Figure 1 illustrates the processing steps that are performed by the algorithm. The four key elements are (1) SLAM processing within local maps, (2) Map management (performing transitions between maps and creating new maps), (3) Map Location estimation (determining the best global location estimate for a submap), and (4) Computation of global state estimates for all features in a map. Each of these processes are described in detail below.

4.1 SLAM within local maps

At any one time, there is a single active map. For each map m , we compute a partial solution, $p(\mathbf{x}_m(k)|Z_m^k, A_m^k, U_m^{k-1})$, where $\mathbf{x}_m(k)$ designates the local map state, and Z_m^k, A_m^k , and U_m^{k-1} represent subsets of the measurements, associations, and control inputs, respectively. Each measurement is used in only a single map. (This is vital for ensuring consistency of the global Map Location estimation process.) Each map m contains an estimated mean $[\hat{\mathbf{x}}_v^m(k)^T \hat{\mathbf{x}}_1^m(k)^T \dots \hat{\mathbf{x}}_n^m(k)^T]^T$ and covariance $\mathbf{P}^m(k)$ corresponding to selected vehicle locations (for time steps when map m is the active map), and only a subset of the features. These estimates are the same as the location vectors $^{[m,i]}T_j$ and associated uncertainty $^{[m,i]}\Sigma_j$ for the features in the local map.

4.2 Map management

In our scheme, each map has a center, which is defined as the vehicle location at the time of the creation of a map. About this center is defined a region of radius r . This defines a

bound of vehicle location and not feature locations — any feature that is observed from a position inside the map region will be added to the local map. The estimated location of the vehicle is used to deduce which map(s) the vehicle is in, and when to make transitions. When the vehicle travels more than $r + h$ from the center, the vehicle is considered to have left the current map. The parameter h is a hysteresis term, to prevent excessive map switching. In the simulation results below, we use $r = 15$ and $h = 5$. (An alternative to circular map regions is the use of convex hulls, but this makes no difference to the fundamental performance of the algorithm.)

We assume that the density of discernible features in a local area is bounded. This provides a bound on the number of features that can belong to a map.

When a vehicle leaves a map, we must determine which map (if any) the vehicle has transitioned to. A list of possible candidates is drawn up from a look up table indexed by quantized vehicle locations. If more than one candidate exists, we choose the map with the lowest ID, i.e., the oldest map. If no candidates are found, then a new map is created at the current location (a distance $r + h$ from the center of the previous map). All of these operations can be performed in constant time.

4.3 Map Location estimation

Map Location estimation is the procedure by which global estimates for feature locations in each local map are improved, resulting in global convergence. This procedure is described as follows:

1. Select a map p , to improve which is currently referenced to the root entity i .
2. Create a set, $\mathcal{N}$, containing the ID's of all nearby maps including p — the map to be improved.
3. For each $q \in \mathcal{N}$, $q \neq p$ create a set $\mathcal{C}_{p,q}$ of ID's of features that are present in both maps.
4. For the frame $\mathcal{F}_k$ attached to feature $k \in \mathcal{C}_{p,q}$, calculate its globally referenced location $^{[q,\cdot]}T_k^G$ and uncertainty $^{[q,\cdot]}\Sigma_k^G$ using the location estimate of feature k within map q and its current location estimate.
5. Pick the map q^* and entity ID k^* such that:
$$[q^*, k^*] = \arg \min \{ |^{[q,\cdot]}\Sigma_k^G| \}$$
6. If $q^* = p$ and $k^* = i$ then stop. The map p cannot be improved
7. Root Shift map p to k^* from i — reference all entities in map p to a coordinate frame attached to entity k^* .
8. Replace T_p^G and Σ_p^G with $^{[q^*,\cdot]}T_{k^*}^G$ and $^{[q^*,\cdot]}\Sigma_{k^*}^G$ respectively.

For constant-time operation, Map Location estimation is performed only when the vehicle transitions from one map to another. Alternatively, the procedure can be performed periodically (or at the end of the mission) to all maps. Multiple iterations result in global convergence to a near-optimal solution, but the computation complexity is no longer $\mathcal{O}(1)$.

The work presented in this paper differs from that proposed in [Leonard and Feder, 2000] on the following counts:

- No vehicle information is carried between maps upon map transitions
- The SLAM scheme adopted within each map need not be based on a Kalman filter.
- Mapped entities are represented in local coordinate frames — one frame per map. In contrast, the DSM approach used multiple maps but all registered in one global coordinate frame.

4.4 Obtaining global location estimates for map features

Given an independent, consistent estimate of the location of a submap p (with root i) with respect to the global frame, G , we can produce a consistent global estimate of the location of any feature j in map p by composition of map and feature locations:

$$[p,\cdot]T_j^G = T_p^G \oplus [p,i]T_j \quad (6)$$

This estimate is consistent because T_p^G and $[p,i]T_j$ are independent. $[p,i]T_j$ is “internal” to the map and T_p^G is “external” to the map.

The existence of a “shared” feature, s , between two maps p and q allows the location estimate T_p^G , of map p , (with root feature s), to be replaced with T_{p+}^G where

$$T_{p+}^G = T_q^G \oplus [q,w]T_s \quad (7)$$

and w , the root of map q is any feature id in map q . Equation 7 should be interpreted as finding an alternative expression for the global location of a shared feature s using quantities associated with map q instead of p . As map p has a root at the shared feature s this expression is by definition an alternative expression for the map location. The minimization step of the algorithm is concerned with finding the best choice of shared feature s .

In the limit each map becomes internally fully correlated and the “min” operation will have no further effect so that for any feature j

$$T_p^G \oplus [p,\cdot]T_j = T_q^G \oplus [q,\cdot]T_j \quad (8)$$

where $\cdot$ is any choice of root. In other words no root-shifting and replace operation can be found that improves the global uncertainty of feature j . For the linear case

$$\Sigma_p^G + [p,\cdot]\Sigma_j = \Sigma_q^G + [q,\cdot]\Sigma_j \quad (9)$$

The maps are rooted on features and so as $k \rightarrow \infty$ then

$$[p,\cdot]\Sigma_j = [q,\cdot]\Sigma_j = 0 \quad (10)$$

and so

$$\Sigma_p^G = \Sigma_q^G \quad (11)$$

which is true for all choices of p and q . Therefore the globally referenced feature location uncertainty $[\cdot,\cdot]\Sigma_j^G$ is the same independent of choice of map ($\cdot$). The value of this limiting value is clearly given by the smallest possible uncertainty in Map Location which is the uncertainty of the first feature initialized in the first map [Dissanayake *et al.*, 2001]. This point is considered further in section 5.

5 Consistency and convergence

We begin by defining the term consistency with regard to an estimate $\hat{\mathbf{x}}(k|k)$ of an r.v $\mathbf{x}(k)$ at time k given all information up until time k . Defining the estimated error vector $\tilde{\mathbf{x}}(k|k) \triangleq \mathbf{x}(k) - \hat{\mathbf{x}}(k|k)$ and the estimate covariance as $P(k|k)$ we write the condition of consistency as:

$$E[\tilde{\mathbf{x}}(k|k)] = \mathbf{0} \quad (12)$$

$$E[[\tilde{\mathbf{x}}(k|k)][\tilde{\mathbf{x}}(k|k)]^T] \leq P(k|k) \quad (13)$$

To show that the global location estimates produced by Equation 6 are consistent, we rely on the following three properties: (1) local map state estimates $[p,i]T_j$ (obtained from the local SLAM solution $[\hat{\mathbf{x}}_v^m(k)^T \hat{\mathbf{x}}_1^m(k)^T \dots \hat{\mathbf{x}}_n^m(k)^T]^T$) are consistent, (2) global state estimates for Map Locations T_m^G are consistent, and (3) the composition of these two pieces of information — local state estimates within a map and global information concerning the location of the map — is consistent.

The consistency of local maps follows directly from the properties of the Kalman filter which in the linear gaussian case is the optimal Bayesian estimator. Clearly, choosing to use a possibly inconsistent estimator such as the EKF in a nonlinear scenario will invalidate these claims. However, the LG case allows statements to be made regarding the *underlying* properties of the CTS algorithm. In a non-linear implementation, the consistency of the LG case can be matched to an arbitrary degree by using Monte-Carlo estimators in each sub-map. Regardless of local estimation techniques, the CTS algorithm preserves its constant time property.

Local maps have three differences from the full solution (a) their base reference (root) is defined by one of the features in the map, (b) relocation is periodically performed to re-initialize the local map when the vehicle transitions back into it, and (c) the base reference of the local map is periodically shifted from one feature in the local map to another (Root Shifting). None of these three differences result in a loss of consistency for the local SLAM solution.

The global Map Location estimate T_m^G for a given map, m , is consistent because it is created via the composition of transformations derived from other local maps, and each local map is independent of other local maps. The composition of transformations from different local maps is a consistent operation (for the linear case).

Finally, the composition of the local map state estimates performed in Equation 6 is a consistent operation, because T_m^G and $[m,i]T_j$ are independent of one another.

While the location estimates for different maps are correlated with one another, and this correlation is not computed by the algorithm, the method is none-the-less consistent because this correlation is never needed. We never fuse Map Location estimates estimates, but rather, perform wholesale replacement. The algorithm keeps track of the best estimate for the global location of the root entity of a given map. A guiding principle of this algorithm is that estimated quantities that are “external” to a map never effect an internal quantity.

Due to the relocation step for map transitions, the information from dead-reckoning measurements for the time step

proceeding the map transition is effectively “lost”. This information, however, does not affect absolute convergence but only the rate of convergence. Because relocation is a consistent operation [Knight, 2002], each partial solution retains all the properties of a Kalman filter SLAM solution [Dissanayake *et al.*, 2001], and hence is provably consistent and convergent. Because each map is local with its base reference as one of the features in the map, in the limit the uncertainty for each local map converges to zero [Dissanayake *et al.*, 2001]. This implies that, in the limit, in any local map utilizing only a subset of Z^k , the relationship between features becomes perfectly known as $k \rightarrow \infty$. A consequence of this is that the covariance of the transformation between any two features, i and j , in any given map tends to zero as $k \rightarrow \infty$. Hence, for local maps in which the base reference (root) is a feature in the local map, the covariance of any feature tends to zero. The key driver for convergence is the behavior in the first submap — submap 1. To illustrate this, consider a robot which moves swiftly outside submap 1 and spends the rest of its mission driving just outside submap 1’s borders. Eventually the maps bordering submap 1 become completely known, including most of the features which appear in submap 1. The global uncertainty of any feature in the bordering submaps can never be less than the global uncertainty in the location of submap 1. Hence, in this scenario, the root shifting mechanism can never decrease the global uncertainty of features in submap 1 (nothing has lower global uncertainty). However as soon as submap 1 is re-entered and begins to be refined, the global uncertainties of maps sharing features with map 1 can be reduced by root-shifting. Hence it is the precision of the first submap built (submap 1) that drives the ultimate performance of the entire system. In addition it is the precision of the first feature mapped within it that drives ultimate performance of submap 1.

Thus in the limit, the lower bound achieved in submap 1 is “inherited” by all other submaps. There are two differences between what occurs in submap 1 in comparison to a full covariance solution that couples estimates for all features in a single map: (a) submap 1 has fewer features in it, and (b) not all of the observations of features that are contained in submap 1 are processed in the submap 1 solution. We believe that consideration (a), the fact that submap 1 has fewer features in it, is what is sacrificed in this approach. Even if some measurements are ignored in submap 1 (vs. the full solution), in the limit as $k \rightarrow \infty$, both maps will converge to a well-defined lower bound. With enough additional time the submap 1 solution can “catch up” to the full solution. However, the fact that the full solution has more features enables in it cannot be compensated for, and hence the full solution achieves a slightly tighter bound. This in effect is the “cost” of computing multiple partial solutions and subsequently combining them, rather than computing one full solution. Our simulations have shown that this result is extremely small (as shown below in Figure 3).

6 Results

This section analyzes the behavior of the technique presented using simulations analysis. The simulations consider an

(x, y) “point” vehicle that moves in the plane with process noise Q of 0.05 or 0.01 (Monte-Carlo) m/sec standard deviation in both x and y and measurement noise R of 0.2 m/sec standard deviation in both x and y . Features are visible if they are within 23 meters of the vehicle location. One visible feature is selected at random each time step to generate the observations. Results are presented for three different scenarios:

1. a simulation involving six cycles through an environment in which eight maps are created (to illustrate basic error convergence behavior),
2. Monte-Carlo analysis of 200 independent trials of a mission involving ten cycles through an environment in which twelve submaps are created (for empirical consistency verification), and
3. results off-line map adjustment for a mission involving a single cycle through an environment in which eighteen maps are created,

6.1 Comparison with the full covariance solution

Figure 2 show the vehicle path for a mission in which eight maps are created. Figure 3 shows the vehicle error as a function of time and active map ID. Spikes in the vehicle error estimate occur at each map transition when relocation (consistent re-initialization of the vehicle pose) is performed. This is particularly obvious towards the end of the experiment when the features are well known and subsequent observations swiftly bring the vehicle covariance down following relocation. Figure 4 shows the difference between the new method and the full solution in the determinant of error covariance for the feature marginals in each map, demonstrating the tight convergence of the method to near-optimal estimates, while never going below the minimum permissible error bounds.

6.2 Monte-Carlo consistency testing

For the purposes of consistency testing, two relative states were logged in the active submap: (1) the difference between the first feature in the state vector and the vehicle, and (2) the difference between the first and second features in the state vector. These two states are stored in a single vector $\hat{\mathbf{y}}^m$ with covariance $\mathbf{Y}^m$ formed from the active map (with id m). This relative vector can be compared to the ‘true’ relative relationships declared by the simulator producing the noise corrupted measurements. The difference between these vectors is the error vector $\tilde{\mathbf{y}}(k)$. The $\tilde{\mathbf{y}}$ vector calculated for each map m will in general involve different features (by definition as each map contains a subset of the set of all features with only some features in common). However each estimated component of $\hat{\mathbf{y}}^m$ should be consistent, hence any sequence $\{\hat{\mathbf{y}}^{m_1}(1), \dots, \hat{\mathbf{y}}^{m_k}(k)\}$ $m_{[0:k]} \in [1 : M]$ where M is the total number of maps built, should also be consistent. Figure 6.2 illustrates the results obtained from Monte-Carlo experiments of the CTS algorithm. The parameters used in the simulation of sensor data and its subsequent processing are given in Table 6.2. All the plots are concerned with the statistical properties of $\tilde{\mathbf{y}}(k)$. The first plot of Figure 6.2 shows the

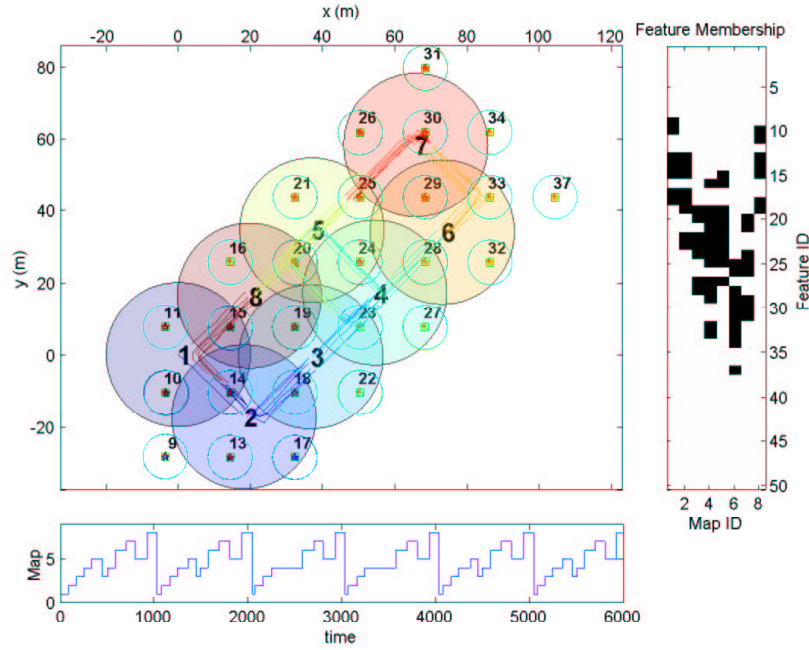


Figure 2: Vehicle path for a mission with eight maps. The eight shaded circles indicate the extent of the submaps in terms of vehicle location alone. Hence features can belong to maps even though they lie outside the submap boundaries. The map ID's are in bold. The right hand plot shows the distribution and sharing of features between maps. Some features are common to several submaps - for example feature 25 is found in submaps 3,4,5,6 and 7 whereas others are found in one alone — feature 37 is only mapped in submap 6. The lower plot shows the map transitions occurring as the vehicle moves. The basic trajectory is one of two overlapping rectangles aligned in North-East direction. The vehicle then returns to the origin (0,0) and repeats the pattern another six times. The feature estimate error ellipses are those resulting at the end of the simulations (in global coordinates) and are three-sigma bounds. The full covariance solution is also plotted on the central figure with its estimated feature locations are plotted with squares. The covariance bounds on features are at this resolution indistinguishable from those of the produced by the CTS algorithm.

mean values of $\tilde{\mathbf{y}}(k)$ — the difference for each time step k between the “true” relative vector $\mathbf{y}$ and the estimate $\hat{\mathbf{y}}$. The upper two plots correspond to the vehicle to feature relative states. Note how the error does not converge to a zero value owing to the continual injection of process noise (odometry errors etc) into the vehicle as it moves. The lower two plots however correspond to the error in the relative location of the first two features in what ever the active map is. No process noise is added to the feature estimate covariances during the prediction stage of the estimation process. The result is the expected convergence to zero error. The second plot of Figure 6.2 shows the plots the characteristics of the N-run average of the NEES (Normalized Estimated Error Squared) of $\hat{\mathbf{y}}(k)$. The 95% confidence region bounds for estimation consistency are plotted on the same axes. the final plot of Figure 6.2 shows the normalized mean estimated error (NMEE) and the associated 95% confidence bounds of the hypothesis that the NMEE sequences are from a consistent estimator.

6.3 Off-line Adjustment

The root shifting operation which searches for a better representation (in terms of uncertainty) of map location and features need not be done as the vehicle transitions between maps. Table 6.3 shows the improvement in successively ap-

Table 1: Monte-Carlo simulation parameters.

Parameter	Value
Monte-Carlo Runs	200
Vehicle Process Noise (std)	0.01 ms ⁻¹
Sensor Noise (std)	0.05 m
Cycle Length	360 m
Cycles per run	10
Total Distance Driven per run	3600 m
Loops per Cycle	2
Initial Vehicle uncertainty	0 m
Sensor Range	25 m
Sensor Field of View	100 deg
Feature Density	Every 18m in x and y
Map Radius	15 m
Vehicle Velocity	0.3 ms ⁻¹

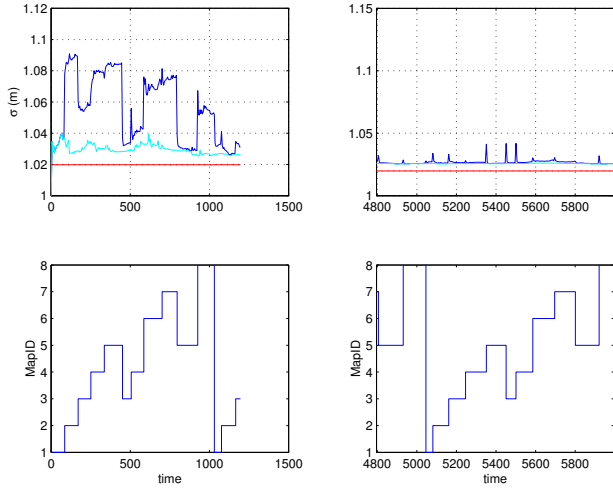


Figure 3: Convergence of vehicle state estimates for the eight map example. The left hand plots show the time steps at the opening stages of the mission while the right hand plots show the closing stages. The straight line is the lowest possible bound on vehicle uncertainty possible for either algorithm in the zero plant noise case. The full vehicle uncertainty given by the full covariance solution is the lighter of the two plots and is always less than or equal to the the CTS solution — the CTS solution is always consistent.

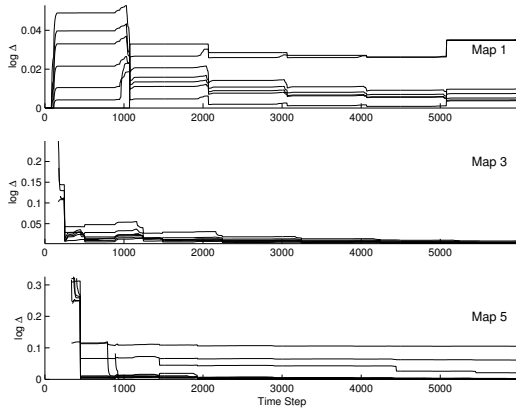


Figure 4: The log of the full-covariance / CTS difference between the determinants of the covariances in the first feature mapped in three randomly chosen submaps. Before the first submap transition occurs, submap 1 computes an answer that is identical to the full solution. Subsequently, submap one performs slightly worse due to the inclusion of less features. Note also that the convergence of submap 1 appears to be slower than the other maps and indeed at times the difference between full and CTS solutions seems to increase. This is because features in submap 1 can only be improved when this submap is active, whereas the full covariance solution updates all features with each new observation.

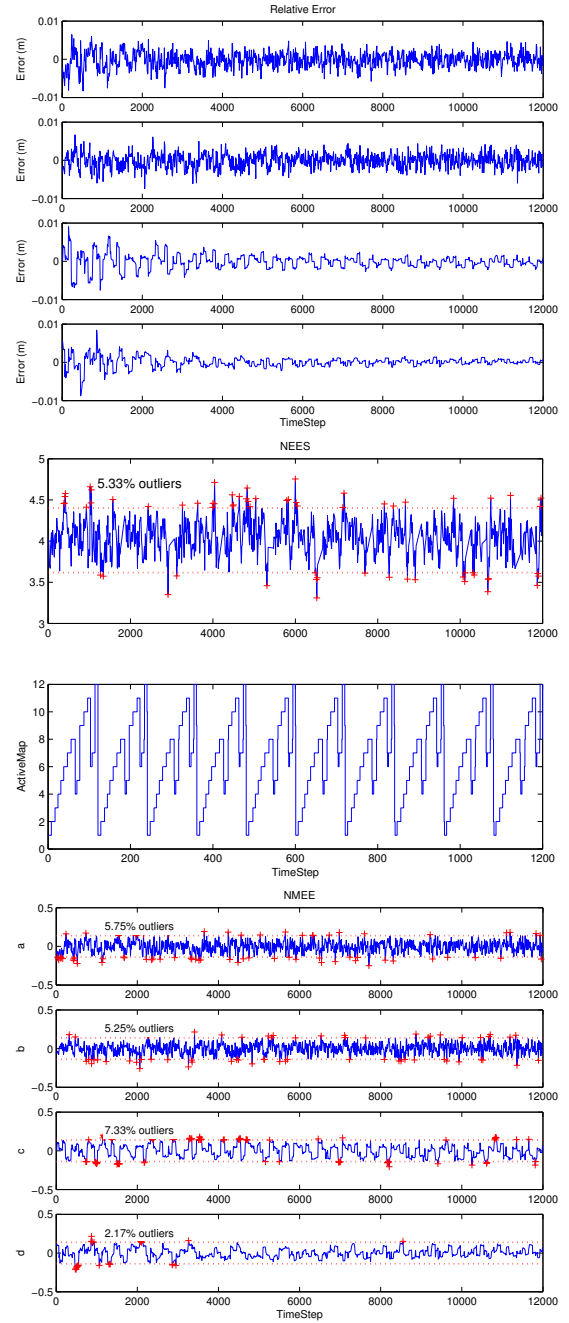


Figure 5: Monte-Carlo Consistency testing. The left hand plot shows the four components of the vector $\tilde{\mathbf{y}}$ which encodes the errors in relative positions of vehicle and the first two features (f_1^m, f_2^m) in the state vector of the active map m (the map in which the vehicle is currently moving). The first upper two sequences are the x and y components of the error in the relative position between vehicle and f_1^m . The second two are the x and y components of the error in the relative positions of f_1^m and f_2^m . The central plot is the NEES of $\tilde{\mathbf{y}}$. The bounds are the 95% confidence intervals on the NEES variable. The right hand plot shows the Normalized Mean Error (NMEE) for the components of $\tilde{\mathbf{y}}$. For consistency the mean estimated error should be zero. The limits plotted are the 95% confidence interval bounds for the consistency hypothesis.

Table 2: Percentage decrease in disparity between full covariance and CTS calculated feature uncertainties during sequential off-line adjustments using the CTS root shifting procedure. After three iterations all eighteen maps in this example showed no further improvement.

Map	iteration 1	iteration 2	iteration 3
1	0.0	0.0	0.0
2	13.6	13.6	13.6
3	10.0	16.8	16.8
4	10.6	18.9	18.9
5	6.3	13.0	17.9
6	7.1	12.6	12.6
7	20.7	25.4	26.6
8	1.8	7.1	8.5
9	6.6	8.3	13.5
10	3.5	5.0	9.5
11	1.6	6.3	7.6
12	15.0	17.8	19.0
13	3.7	5.4	10.1
14	4.1	5.5	9.6
15	4.5	6.0	10.2
16	7.5	9.0	13.4
17	14.4	22.1	22.1
18	12.9	12.9	12.9

plying the CTS algorithm to maps at the termination of the mission for a scenario with 18 maps. The simple approach adopted here begins with submap 2 and seeks to improve the map location estimate, it then progresses to submap 3 and so on until the highest index map has been adjusted. This then repeats until no maps show any further improvement. It is a topic of future research to determine, as a function of shared feature topography, the optimal adjustment sequence rather than the linear one used here. The adjustment was applied to a mission with a vehicle trajectory defined by six overlapping rectangles (similar to Figure 2 but the course was not repeated leading to a more uncertain collection of submaps at the end of the mission). The entries in the table are the percentage decrease in median disparity between the full covariance and CTS calculated feature uncertainties. As expected, the table shows no improvement for submap 1.

7 Conclusion

In this paper, we have presented a new technique for SLAM with multiple maps that achieves consistency of global state estimates with with $O(1)$ time complexity. If the mobile robot is able to make repeated visits to all parts of the environment, by performing map improvement operations only when a map is exited, near-optimal convergence is demonstrated in constant-time.

In this approach, all SLAM filtering is performed in local submaps. Each submap has an associated reference frame, whose global position is estimated using information from other submaps. The transition of the vehicle between submaps is consistently performed using relocation [Neira *et al.*, 2002; Knight, 2002]. By never mixing information between maps, the method yields provably consistent global state estimates, while still achieving global convergence.

The method successfully exploits the fact that overlapping features are estimated in different maps. By changing the base reference of a map to be the feature within the map that has the “best” globally referenced position estimate, the global location estimates for all the features in a local region

get improved via operations in other maps.

Ongoing research includes testing of the method with real data, the incorporation of probabilistic data association techniques, and extension of the approach to the nonlinear case.

References

- [Bosse *et al.*, 2003] M. Bosse, P. Newman, J. Leonard, M. Soika, W. Feiten, and S. Teller. An atlas framework for scalable mapping. In *Proc. IEEE Int. Conf. Robotics and Automation*, 2003.
- [Choset and Nagatani, 2001] H. Choset and K. Nagatani. Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization. *IEEE Transactions on Robotic and Automation*, 17(2):125–137, April 2001.
- [Dissanayake *et al.*, 2001] M. W. M. G. Dissanayake, P. Newman, H. F. Durrant-Whyte, S. Clark, and M. Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotic and Automation*, 17(3):229–241, June 2001.
- [Guivant and Nebot, 2001] J. Guivant and E. Nebot. Optimization of the simultaneous localization and map building algorithm for real time implementation. *IEEE Transactions on Robotic and Automation*, 17(3):242–257, June 2001.
- [Gutmann and Konolige, 1999] J.-S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *International Symposium on Computational Intelligence in Robotics and Automation*, 1999.
- [Hahnel *et al.*, 2002] D. Hahnel, D. Schulz, and W. Burgard. Map building with mobile robots in populated environments. In *Proc. IEEE Int. Workshop on Intelligent Robots and Systems*, 2002.
- [Julier and Uhlmann, 2001] S. J. Julier and J. K. Uhlmann. Building a Million Beacon Map. In *Sensor Fusion*. SPIE, 2001.
- [Knight, 2002] J. Knight. *Towards Fully Autonomous Visual Navigation*. PhD thesis, University of Oxford, 2002.
- [Kuipers and Beeson, 2002] B. Kuipers and P. Beeson. Bootstrap learning for place recognition. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002. AAAI.
- [Leonard and Feder, 2000] J. J. Leonard and H. J. S. Feder. A computationally efficient method for large-scale concurrent mapping and localization. In D. Koditschek and J. Hollerbach, editors, *Robotics Research: The Ninth International Symposium*, pages 169–176. Snowbird, Utah, 2000. Springer Verlag.
- [Montemerlo *et al.*, 2002] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fast-SLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002. AAAI.
- [Neira and Tardós, 2001] J. Neira and J.D. Tardós. Data association in stochastic mapping using the joint compatibility test. *IEEE Trans. on Robotics and Automation*, 17(6):890–897, 2001.
- [Neira *et al.*, 2002] J. Neira, J.D. Tardós, and J.A. Castellanos. Linear time vehicle relocation in SLAM. Technical Report RR-2002-08, Universidad de Zaragoza, Depto. de Informática e Ing. de Sistemas, Zaragoza, Spain, 2002.
- [Smith *et al.*, 1987] R. Smith, M. Self, and P. Cheeseman. A stochastic map for uncertain spatial relationships. In *4th International Symposium on Robotics Research*. MIT Press, 1987.
- [Tardós *et al.*, 2002] J.D. Tardós, J. Neira, P.M. Newman, and J.J. Leonard. Robust mapping and localization in indoor environments using sonar data. *Int. J. Robotics Research*, 21(4):311–330, April 2002.
- [Thrun *et al.*, 2002] S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, and Ng. A.Y. Simultaneous mapping and localization with sparse extended information filters. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Proceedings of the Fifth International Workshop on Algorithmic Foundations of Robotics*, Nice, France, 2002. Forthcoming.
- [Thrun, 2001] S. Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *Int. J. Robotics Research*, 20(5):335–363, May 2001.
- [Williams *et al.*, 2002] S. B. Williams, G. Dissanayake, and H. Durrant-Whyte. An efficient approach to the simultaneous localisation and mapping problem. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 406–411, May 2002.

FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges

Michael Montemerlo and Sebastian Thrun

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

mmde@cs.cmu.edu, thrun@cs.cmu.edu

Daphne Koller and Ben Wegbreit

Computer Science Department
Stanford University
Stanford, CA 94305-9010

koller@cs.stanford.edu, ben@wegbreit.com

Abstract

In [15], Montemerlo et al. proposed an algorithm called *FastSLAM* as an efficient and robust solution to the simultaneous localization and mapping problem. This paper describes a modified version of FastSLAM that overcomes important deficiencies of the original algorithm. We prove convergence of this new algorithm for linear SLAM problems and provide real-world experimental results that illustrate an order of magnitude improvement in accuracy over the original FastSLAM algorithm.

1 Introduction

Simultaneous localization and mapping (SLAM) is a highly active research area in robotics and AI. The SLAM problem arises when a moving vehicle (e.g. a mobile robot, submarine, or drone) simultaneously estimates a map of its environment and its pose relative to that map. In the absence of global position information, the vehicle's pose estimate will become increasingly inaccurate, as will its map. Since maps may contain thousands of entities, acquiring large, accurate maps is a challenging statistical estimation problem, especially when performed in real-time.

Most present-day research on SLAM originates from a seminal paper by Smith and Cheeseman [21], which proposed the use of the extended Kalman filter (EKF) for solving SLAM. This paper is based on the insights that errors in the map and pose errors are naturally correlated, and that the covariance matrix maintained by the EKF expresses such correlations. Newmann [18] recently proved that the EKF converges for linear SLAM problems, where the motion model and observation model are linear functions with Gaussian noise (see below).

Unfortunately, EKF covariance matrices are quadratic in the size of the map, and updating them requires time quadratic in the number of landmarks N . This quadratic complexity has long been recognized to be a major obstacle in scaling SLAM algorithms to maps with more than a few hundred features. It also limits the applicability of SLAM algorithms to problems with ambiguous landmarks, which induces a *data association problem* [2; 22]. Today's most robust algorithms for SLAM with unknown data association maintain multiple hypotheses (tracks), which increase their computational complexity.

Consequently, there has been a flurry on research on more efficient SLAM techniques (see e.g., [11]). One group of researchers has developed techniques that recursively divide maps into submaps, thereby confining most computation to small regions. Some of these approaches still maintain global

correlations among those submaps, hence are quadratic but with a much reduced constant factor [1; 7; 22; 25]. Others restrict the update exclusively to local maps [12], hence operate in constant time (assuming known data association).

A second group of researchers has developed techniques that represent maps through potential functions between adjacent landmarks, similar to Markov random fields. The resulting representations require memory linear in the number of landmarks [19; 23]. Under appropriate approximations, such techniques have been shown to provide constant time updating (again for known data association). Unfortunately, no convergence proof exists for any of these extensions of the EKF, even for the generic case of linear SLAM. Furthermore, if landmarks are ambiguous, all of these approaches have to perform search to find appropriate data association hypotheses, adding a logarithmic factor to their update complexity.

The FastSLAM algorithm, proposed in [15] as an efficient approach to SLAM based on particle filtering [6], does not fall into either of the categories above. FastSLAM takes advantage of an important characteristic of the SLAM problem (with known data association): landmark estimates are conditionally independent given the robot's path [17]. FastSLAM uses a particle filter to sample over robot paths. Each particle possesses N low-dimensional EKFs, one for each of the N landmarks. This representation requires $O(NM)$ memory, where M is the number of particles in the particle filter. Updating this filter requires $O(M \log N)$ time, with or without knowledge of the data associations. However, the number of particles M needed for convergence is unknown and has been suspected to be exponential in the size of the map, in the worst-case.

This paper proposes an improved version of the FastSLAM algorithm. The modification is conceptually simple: When proposing a new robot pose—an essential step in FastSLAM's particle filter—our proposal distribution relies not only on the motion estimate (as is the case in FastSLAM), but also on the most recent sensor measurement. Such an approach is less wasteful with its samples than the original FastSLAM algorithm, especially in situations where the noise in motion is high relative to the measurement noise.

To obtain a suitable proposal distribution, our algorithm linearizes the motion and the measurement model in the same manner as the EKF. As a result, the proposal distribution can be calculated in closed form. This extension parallels prior work by Doucet and colleagues, who proposed a similar modification for general particle filters [6] and Markov Chain Monte Carlo techniques for neural networks [4]. It is similar to the arc reversal technique proposed for particle filters applied to

Bayes networks [10], and it is similar to recent work by van der Merwe [24], who uses an unscented filtering step [9] for generating proposal distributions that accommodate the measurement.

While this modification is conceptually simple, it has important ramifications. A key contribution of this paper is a convergence proof for linear SLAM problems using a single particle. The resulting algorithm requires constant updating time. To our knowledge, the best previous SLAM algorithm for which convergence was shown requires quadratic update time. Furthermore, we observe experimentally that our new FastSLAM algorithm, even with a single particle, yields significantly more accurate results on a challenging real-world benchmark [7] than the previous version of the algorithm. These findings are of significance, as many mobile robot systems are plagued by control noise, but possess relatively accurate sensors. Moreover, they contradict a common belief that maintaining the entire covariance matrix is required for convergence [5].

2 Simultaneous Localization and Mapping

SLAM addresses the problem of simultaneously recovering a map and a vehicle pose from sensor data. The map contains N features (landmarks) and shall be denoted $\Theta = \theta_1, \dots, \theta_N$. The path of the vehicle will be denoted $s^t = s_1, \dots, s_t$, where t is a time index and s_t is the pose of the vehicle at time t .

Most state-of-the-art SLAM algorithms calculate (or approximate) variants of the following posterior distribution:

$$p(\Theta, s^t | z^t, u^t, n^t) \quad (1)$$

where $z^t = z_1, \dots, z_t$ is a sequence of measurements (e.g., range and bearing to nearby landmarks), and $u^t = u_1, \dots, u_t$ is a sequence of robot controls (e.g., velocities for robot wheels). (As usual, we assume without loss of generality that only a single landmark is observed at each time t .) The variables $n^t = n_1, \dots, n_t$ are *data association variables* — each n_t specifies the identity of the landmark observed at time t . Initially, we assume n^t is known; we relax this assumption below.

To calculate the posterior (1), the vehicle is given a probabilistic motion model, in the form of the conditional probability distribution $p(s_t | u_t, s_{t-1})$. This distribution describes how a control u_t , asserted in the time interval $[t-1; t)$, affects the resulting pose. Additionally, the vehicle is given a probabilistic measurement model, denoted $p(z_t | s_t, \Theta, n_t)$, describing how measurements evolve from state. In accordance to the rich SLAM literature, we will model both models by nonlinear functions with independent Gaussian noise:

$$p(z_t | s_t, \Theta, n_t) = g(s_t, \theta_{n_t}) + \varepsilon_t \quad (2)$$

$$p(s_t | u_t, s_{t-1}) = h(u_t, s_{t-1}) + \delta_t \quad (3)$$

Here g and h are nonlinear functions, and ε_t and δ_t are Gaussian noise variables with covariance R_t and P_t , respectively.

3 FastSLAM

FastSLAM [15] is based on the important observation [17] that the posterior can be factored

$$p(\Theta, s^t | z^t, u^t, n^t) = p(s^t | z^t, u^t, n^t) \prod_n p(\theta_n | s^t, z^t, u^t, n^t) \quad (4)$$

This factorization is exact and universal in SLAM problems. It states that if one (hypothetically) knew the path of the vehicle, the landmark positions could be estimated independently

of each other (hence the product over n). In practice, of course, one does not know the vehicle's path. Nevertheless, the independence makes it possible to factor the posterior into a term that estimates the probability of each path, and N terms that estimate the position of the landmarks, conditioned on each (hypothetical) path.

FastSLAM samples the path using a particle filter. Each particle has attached its own map, consisting of N extended Kalman filters. Formally, the m -th particle $S_t^{[m]}$ contains a path $s_t^{[m]}$ along with N Gaussian landmark estimates, described by the mean $\mu_{n,t}^{[m]}$ and covariance $\Sigma_{n,t}^{[m]}$:

$$S_t^{[m]} = s_t^{[m]}, \underbrace{\mu_{1,t}^{[m]}, \Sigma_{1,t}^{[m]}}_{\text{landmark } \theta_1}, \dots, \underbrace{\mu_{N,t}^{[m]}, \Sigma_{N,t}^{[m]}}_{\text{landmark } \theta_N} \quad (5)$$

We briefly review the key equations of the regular FastSLAM algorithm, and refer the reader to [15]. Each update in FastSLAM begins with sampling new poses based on the most recent motion command u_t :

$$s_t^{[m]} \sim p(s_t | s_{t-1}^{[m]}, u_t). \quad (6)$$

Note that this proposal distribution only uses the motion command u_t , but *ignores* the measurement z_t .

Next, FastSLAM updates the estimate of the observed landmark(s), according to the following posterior. This posterior takes the measurement z_t into consideration:

$$\begin{aligned} p(\theta_{n_t} | s_t^{[m]}, n_t, z_t) \\ = \eta \underbrace{p(z_t | \theta_{n_t}, s_t^{[m]}, n_t)}_{\sim \mathcal{N}(z_t; g(\theta_{n_t}, s_t^{[m]}), R_t)} \underbrace{p(\theta_{n_t} | s^{t-1, [m]}, z^{t-1}, n^{t-1})}_{\sim \mathcal{N}(\theta_{n_t}; \mu_{n_t, t-1}^{[m]}, \Sigma_{n_t, t-1}^{[m]})} \end{aligned} \quad (7)$$

Here η is a constant. This posterior is the normalized product of two Gaussians as indicated. However, if g is non-linear, the product will not be Gaussian in general. To make the result Gaussian, FastSLAM employs the standard EKF “trick” [13]: g is approximated by a linear function (see below). Under this approximation, (7) is equivalent to the measurement update equation familiar from the EKF literature [13].

In a final step, FastSLAM corrects for the fact that the pose sample $s_t^{[m]}$ has been generated without consideration of the most recent measurement. It does so by resampling the particles [20]. The probability for the m -th particle to be sampled (with replacement) is given by the following variable $w_t^{[m]}$, commonly referred to as *importance factor*:

$$w_t^{[m]} = \eta \int \underbrace{p(z_t | \theta_{n_t}, s_t^{[m]}, n_t)}_{\sim \mathcal{N}(z_t; g(\theta_{n_t}, s_t^{[m]}), R_t)} \underbrace{p(\theta_{n_t} | s^{t-1, [m]}, z^{t-1}, n^{t-1})}_{\sim \mathcal{N}(\mu_{n_t, t-1}^{[m]}, \Sigma_{n_t, t-1}^{[m]})} d\theta_{n_t}$$

As shown in [15], the resampling operation can be implemented in $O(M \log N)$ time using trees, where M is the number of samples and N the number of landmarks in the map. However, the number of particles M needed for convergence remains an open question.

FastSLAM has been extended to SLAM with unknown data associations [14]. If the data association is unknown, each particle m in FastSLAM makes its own local data association decision $\hat{n}_t^{[m]}$, by maximizing the measurement likelihood. The formula for finding the most likely data association maximizes the resulting importance weight:

$$\hat{n}_t^{[m]} = \underset{n_t}{\operatorname{argmax}} w_t^{[m]}(n_t) \quad (8)$$

Here $w_t^{[m]}(n_t)$ makes the dependence of the factor $w_t^{[m]}$ on the variable n_t explicit. A key characteristic of FastSLAM is that each particle makes its own local data association. In contrast, EKF techniques must commit to a single data association hypothesis for the entire filter. Results in [14] show empirically that this difference renders FastSLAM significantly more robust to noise than EKF-style algorithms.

4 FastSLAM 2.0

Our new FastSLAM algorithm is based on an obvious inefficiency arising from the proposal distribution of regular FastSLAM. In regular FastSLAM, the pose $s_t^{[m]}$ is sampled in accordance to the prediction arising from the motion command u_t , as specified in (6). It does not consider the measurement z_t acquired at time t ; instead, the measurement is incorporated through resampling. This approach is particularly troublesome if the noise in the vehicle motion is large relative to the measurement noise. In such situations, sampled poses will mostly fall into areas of low measurement likelihood, and will subsequently be terminated in the resampling phase with high probability. Unfortunately, many real-world robot systems are characterized by relatively high motion noise. As illustrated in the experimental results section of this paper, the waste incurred by this inefficient sampling scheme can be significant.

4.1 Sampling The Pose

FastSLAM 2.0 implements a single new idea: Poses are sampled under consideration of both the motion u_t and the measurement z_t . This is formally denoted by the following sampling distribution, which now takes the measurement z_t into consideration:

$$s_t^{[m]} \sim p(s_t | s^{t-1,[m]}, u^t, z^t, n^t) \quad (9)$$

In comparison to (6), incorporating the measurement only makes sense if we incorporate our current estimate of the observed landmark—obtained from the variables $s^{t-1,[m]}, u^{t-1}, z^{t-1}, n^{t-1}$ (which are included of the conditioning variables above). So in essence, the difference to FastSLAM is that the measurement z_t is incorporated. However, this change has important ramifications.

The proposal distribution (9) can be reformulated as follows:

$$\begin{aligned} p(s_t | s^{t-1,[m]}, u^t, z^t, n^t) &= \eta^{[m]} \int \underbrace{p(z_t | \theta_{n_t}, s_t, n_t)}_{\sim \mathcal{N}(z_t; g(\theta_{n_t}, s_t), R_t)} \underbrace{p(\theta_{n_t} | s^{t-1,[m]}, z^{t-1}, n^{t-1})}_{\sim \mathcal{N}(\theta_{n_t}; \mu_{n_t, t-1}^{[m]}, \Sigma_{n_t, t-1}^{[m]})} d\theta_{n_t} \\ &\quad \underbrace{p(s_t | s_{t-1}^{[m]}, u_t)}_{\sim \mathcal{N}(s_t; h(s_{t-1}^{[m]}, u_t), P_t)} \end{aligned} \quad (10)$$

That is, the proposal distribution is the product of two factors: the familiar next state distribution $p(s_t | s_{t-1}^{[m]}, u_t)$, and the probability of the measurement z_t . Calculating the latter involves an integration over possible landmark locations θ_{n_t} .

Unfortunately, sampling directly from this distribution is impossible in the general case; it does not even possess a closed form. Luckily, a closed form solution can be attained if g is approximated by a linear function (h may remain non-linear!):

$$g(\theta_{n_t}, s_t) \approx \hat{z}_t^{[m]} + G_\theta \cdot (\theta_{n_t} - \mu_{n_t, t-1}^{[m]}) + G_s \cdot (s_t - \hat{s}_t^{[m]})$$

where $\hat{z}_t^{[m]} = g(\hat{\theta}_{n_t}^{[m]}, \hat{s}_t^{[m]})$ denotes the predicted measurement, $\hat{s}_t^{[m]} = h(s_{t-1}^{[m]}, u_t)$ the predicted robot pose, and $\hat{\theta}_{n_t}^{[m]} = \mu_{n_t, t-1}^{[m]}$ the predicted landmark location. The matrices G_θ and G_s are the Jacobians (first derivatives) of g with respect to θ and s , respectively:

$$G_\theta = \nabla_{\theta_{n_t}} g(\theta_{n_t}, s_t) \big|_{s_t=\hat{s}_t^{[m]}; \theta_{n_t}=\hat{\theta}_{n_t}^{[m]}} \quad (11)$$

$$G_s = \nabla_{s_t} g(\theta_{n_t}, s_t) \big|_{s_t=\hat{s}_t^{[m]}; \theta_{n_t}=\hat{\theta}_{n_t}^{[m]}} \quad (12)$$

Under this EKF-style approximation, the proposal distribution (9) is Gaussian with the following parameters:

$$\Sigma_{s_t}^{[m]} = \left[G_s^T Q_t^{[m]-1} G_s + P_t^{-1} \right]^{-1} \quad (13)$$

$$\mu_{s_t}^{[m]} = \Sigma_{s_t}^{[m]} G_s^T Q_t^{[m]-1} (z_t - \hat{z}_t^{[m]}) + \hat{s}_t^{[m]} \quad (14)$$

where the matrix $Q_t^{[m]}$ is defined as follows:

$$Q_t^{[m]} = R_t + G_\theta \Sigma_{n_t, t-1}^{[m]} G_\theta^T \quad (15)$$

4.2 Updating The Observed Landmark Estimate

The updating step remains the same as in FastSLAM (see (7)). As stated in the previous section, g is linearized to retain Gaussianity of the posterior. This leads to the following update equations, whose derivation is equivalent to that of the standard EKF measurement update [13]:

$$K_t^{[m]} = \Sigma_{n_t, t-1}^{[m]} G_\theta^T Q_t^{[m]-1} \quad (16)$$

$$\mu_{n_t, t}^{[m]} = \mu_{n_t, t-1}^{[m]} + K_t^{[m]} (z_t - \hat{z}_t^{[m]}) \quad (17)$$

$$\Sigma_{n_t, t}^{[m]} = (I - K_t^{[m]} G_\theta) \Sigma_{n_t, t-1}^{[m]} \quad (18)$$

4.3 The Importance Weights

Resampling is necessary even in our new version of FastSLAM, since the particles generated do not yet match the desired posterior. The culprit is the normalizer $\eta^{[m]}$ in (10), which may be different for different particles m . This normalizer is the inverse of the probability of the measurement under the m -th particle: $\eta^{[m]} = p(z_t | s^{t-1,[m]}, u^t, z^{t-1}, n^{t-1})^{-1}$. To account for this mismatch, our algorithm resamples in proportion to the following importance factor:

$$\begin{aligned} w_t^{[m]} &\propto p(z_t | s^{t-1,[m]}, u^t, z^{t-1}, n^{t-1}) \\ &= \int \int \underbrace{p(z_t | \theta_{n_t}, s_t, n_t)}_{\sim \mathcal{N}(z_t; g(\theta_{n_t}, s_t), R_t)} \underbrace{p(\theta_{n_t} | s^{t-1,[m]}, u^{t-1}, z^{t-1}, n^{t-1})}_{\sim \mathcal{N}(\theta_{n_t}; \mu_{n_t, t-1}^{[m]}, \Sigma_{n_t, t-1}^{[m]})} d\theta_{n_t} \underbrace{p(s_t | s_{t-1}^{[m]}, u_t)}_{\sim \mathcal{N}(s_t; h(s_{t-1}^{[m]}, u_t), P_t)} ds_t \end{aligned}$$

This expression can once again be approximated as a Gaussian by linearizing g . The mean of this Gaussian is $\hat{z}_t$, and its covariance is

$$G_s P_t G_s^T + G_\theta \Sigma_{n_t, t-1}^{[m]} G_\theta^T + R_t \quad (19)$$

4.4 Unknown Data Associations

The approach for handling data association is similar to the one in regular FastSLAM: Again, we select the data association n_t that maximizes the probability of the measurement z_t for the m -th particle:

$$\hat{n}_t^{[m]} = \underset{n_t}{\operatorname{argmax}} p(z_t | n_t, \hat{n}_t^{t-1,[m]}, s_t^{t,[m]}, z^{t-1}, u^t) \quad (20)$$

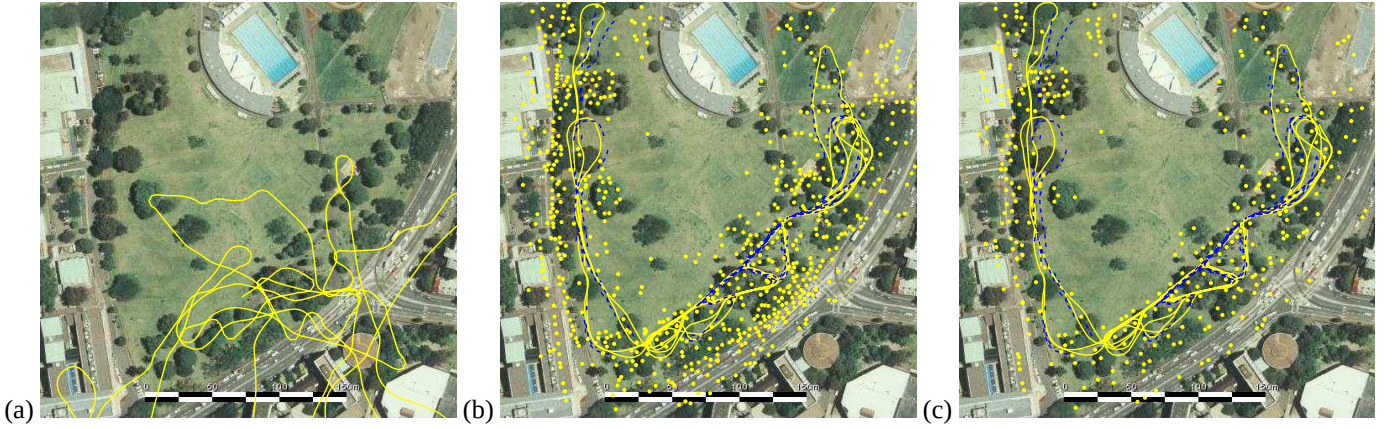


Figure 1: FastSLAM 2.0 applied to the Victoria Park benchmark data set. (a) Raw vehicle odometry (b) FastSLAM 2.0, $M=1$ particle (c) FastSLAM 2.0 with dynamic feature management.

At first glance, one may be tempted to substitute $w_t^{[m]}$ for the probability on the right-hand side, as in regular FastSLAM. However, $w_t^{[m]}$ does not consider the sampled pose $s_t^{[m]}$, whereas the expression here does. This leads to a slightly different probability, which is calculated as follows.

$$p(z_t | n_t, \hat{n}^{t-1, [m]}, s^{t, [m]}, z^{t-1}, u^t) \quad (21)$$

$$= \int \underbrace{p(z_t | \theta_{n_t}, n_t, s_t^{[m]})}_{\sim \mathcal{N}(z_t; g(\theta_{n_t}, s_t^{[m]}), R_t)} \underbrace{p(\theta_{n_t} | \hat{n}^{t-1, [m]}, s^{t-1, [m]}, z^{t-1})}_{\sim \mathcal{N}(\mu_{n_t, t-1}^{[m]}, \Sigma_{n_t, t-1}^{[m]})} d\theta_{n_t}$$

Linearization of g leads to a Gaussian over z_t with mean $g(\mu_{n_t, t-1}^{[m]}, s_t^{[m]})$ and covariance $Q_t^{[m]}$. Both are functions of the data association variable n_t .

4.5 Feature Management

Finally, in cases with unknown data associations, features have to be created dynamically. As is common for SLAM algorithms [5], our approach creates new features when the measurement probability in (20) is below a threshold. However, real-world data with frequent outliers will generate spurious landmarks using this rule. Following [5], our approach removes such spurious landmarks by keeping track of their posterior probability of existence. Our mechanism analyzes measurement to the presence *and* absence of features. Observing a landmark provides positive evidence for its existence, whereas not observing it when $\mu_n^{[m]}$ falls within the robot's perceptual range provides negative evidence. The posterior probability of landmark existence is accumulated by the following Bayes filter, whose log-odds form is familiar from the literature on occupancy grid maps [16]:

$$\tau_n^{[m]} = \sum_t \ln \frac{p(i_n^{[m]} | s_t^{[m]}, z_t, \hat{n}_t^{[m]})}{1 - p(i_n^{[m]} | s_t^{[m]}, z_t, \hat{n}_t^{[m]})} \quad (22)$$

Here $\tau_n^{[m]}$ are the log-odds of the physical existence of landmark $\theta_n^{[m]}$ in map m , and $p(i_n^{[m]} | s_t^{[m]}, z_t, \hat{n}_t^{[m]})$ is the probabilistic evidence provided by a measurement. Under appropriate definition of the latter, this rule provides for a simple evidence counting rule. If the log odds drops below a predefined threshold, the corresponding landmark is removed from the map. This mechanism enables particles to free themselves of spurious features.

5 Convergence

A key result in this paper is the fact that our new version of FastSLAM converges for $M=1$ particle, for a restricted class of linear Gaussian problems (the same for which KFs converge [5; 18]). Specifically, our result applies to SLAM problems characterized by the following linear form:

$$g(s_t, \theta_{n_t}) = \theta_{n_t} - s_t \quad (23)$$

$$h(u_t, s_{t-1}) = u_t + s_{t-1} \quad (24)$$

Linear SLAM can be thought of as a robot operating in a Cartesian space equipped with a noise-free compass, and sensors that measure distances to features along the coordinate axes. The following theorem, whose proof can be found in the appendix, states the convergence of our new FastSLAM variant:

Theorem. *For linear SLAM, FastSLAM with $M=1$ particles converges in expectation to the correct map if all features are observed infinitely often, and if the location of one feature is known in advance.*

This theorem parallels a similar result previously published for the Kalman filter [5; 18]. However, this result applies to the Kalman filter, whose update requires time quadratic in the number of landmarks N . With $M=1$, the resampling step becomes obsolete and each update takes constant time. To our knowledge, our result is the first convergence result for a constant-time SLAM algorithm. It even holds if all features are arranged in a large loop, a situation often thought of as the worst case for SLAM problems [8].

6 Experimental Results

Systematic experiments showed that FastSLAM 2.0 provides excellent results with surprisingly few particles, including $M=1$. Most of our experiments were carried out using a benchmark data set collected with an outdoor vehicle in Victoria Park, Sydney [7]. The vehicle path is 3.5km long, and the map is 320 meters wide. The vehicle is equipped with differential GPS that is used for evaluation only. Fig. 1a shows the map of the terrain, along with the path obtained by raw odometry (which is very poor, the average RMS error is 93.6 meters). This data set is presently the most popular benchmark in the SLAM research community [3].

Figs. 1b&c show the result of applying FastSLAM with $M=1$ particle to the data set, without (Fig. 1b) and with (Fig. 1c) the feature management approach described in Section 4.5. In both cases, the estimated vehicle path is shown

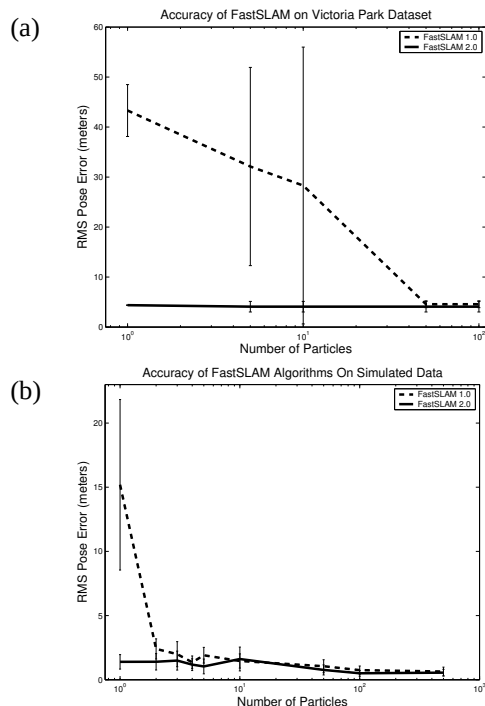


Figure 2: RMS map error for regular FastSLAM (dashed line) versus FastSLAM 2.0 (solid line) on (a) the Victoria Park data (b) simulated data. FastSLAM 2.0's results even with a single particle are excellent, as a solid line, and the GPS information is shown as a dashed line. Results of the same accuracy were previously achieved only with $O(N^2)$ EKF-style methods [7] and with FastSLAM using $M=50$ particles. The feature management rule reduces the number of landmarks in the map from 768 (Fig. 1b) to 343 (Fig. 1c).

Fig. 2 plots the RMS error of the vehicle position estimate as function of the number of particles for the Victoria data set (panel a) and for synthetic simulation data (panel b) taken from [15]. While our new algorithm does approximately equally well for any number of particles, regular FastSLAM performs poorly for very small particle sets. We suspect that the poor performance of regular FastSLAM is due to the fact that the vehicle possesses relatively inaccurate odometry (see Fig. 1a), yet uses a low-noise range finder for landmark detection (a common configuration in outdoor robotics), leading to the generation of many particles of low likelihood.

The small number of examples needed to obtain state-of-the-art estimation translates to unprecedented efficiency of the new filter. The following table shows the results required to process the Victoria Park data set on a 1GHz Pentium PC:

EKF	7,807 sec
regular FastSLAM, $M=50$ particles	315 sec
FastSLAM 2.0, $M=1$ particle	54 sec

In comparison, the data acquisition required 1,550 seconds. Thus, while EKFs cannot be run in real-time, our new algorithm requires less than 4% of the vehicle's trajectory time.

7 Discussion

This paper describes a modified FastSLAM algorithm that is uniformly superior to the FastSLAM algorithms proposed in [15]. The new FastSLAM algorithm utilizes a different proposal distribution which incorporates the most recent measurement in the pose prediction process. In doing so, it makes more

efficient use of the particles, particularly in situations in which the motion noise is high in relation to the measurement noise.

A main contribution of this paper is a convergence proof for FastSLAM with a single particle. This proof is an improvement over previous formal results, which applied to algorithms much less efficient than the current one. In fact, this result is a first convergence result for a constant time SLAM algorithm.

The theoretical finding is complemented by experimental results using a standard benchmark data set. The new algorithm is found to outperform the previous FastSLAM algorithm and the EKF approach to SLAM by a large margin. In fact, a single particle suffices to generate an accurate map of a challenging benchmark data set. Despite this surprising result, the use of multiple particles is clearly warranted in situations with ambiguous data association. We believe that our results illustrate that SLAM can be solved robustly by algorithms that are significantly more efficient than EKF-based algorithms.

8 Acknowledgments

This work was supported by DARPA's MARS and MARS2020 program. Daphne Koller was supported by the Office of Naval Research, Young Investigator (PECASE) grant N00014-99-1-0464. We thank the Hertz Foundation for their support of Michael Montemerlo's graduate research.

References

- [1] T. Bailey. *Mobile Robot Localisation and Mapping in Extensive Outdoor Environments*. PhD thesis, Univ. of Sydney, 2002.
- [2] Y. Bar-Shalom and T. E. Fortmann. *Tracking and Data Association*. Academic Press, 1988.
- [3] SLAM summer school 2002 <http://www.cas.kth.se/SLAM/>
- [4] N. de Freitas, M. Niranjan, A. Gee, and A. Doucet. Sequential monte carlo methods to train neural network models. *Neural Computation*, 12(4), 2000.
- [5] G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localisation and map building (SLAM) problem. *IEEE Trans. Robotics and Automation*, 2001.
- [6] A. Doucet, J.F.G. de Freitas, and N.J. Gordon, editors. *Sequential Monte Carlo Methods In Practice*. Springer, 2001.
- [7] J. Guivant and E. Nebot. Optimization of the simultaneous localization and map building algorithm for real time implementation. *Trans. of Robotics and Automation*, 2001.
- [8] J.-S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. *Proc. CIRA*, 2000.
- [9] S. J. Julier and J. K. Uhlmann. A new extension of the Kalman filter to nonlinear systems. *Proc. AeroSense*, 1997.
- [10] K. Kanazawa, D. Koller, and S.J. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. *UAI-95*.
- [11] *Notes ICRA Workshop Concurrent Mapping and Localization for Autonomous Mobile Robots*, 2002
- [12] J.J. Leonard and H.J.S. Feder. A computationally efficient method for large-scale concurrent mapping and localization. *Proc. IRSS*, 1999.
- [13] P. Maybeck. *Stochastic Models, Estimation, and Control, Volume 1*. Academic Press, 1979.
- [14] M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using FastSLAM. *Proc. ICRA*, 2003, to appear.
- [15] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. *Proc. AAAI*, 2002.
- [16] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2), 1988.

- [17] K. Murphy. Bayesian map learning in dynamic environments. *Proc. NIPS*, 1999.
- [18] P. Newman. *On the Structure and Solution of the Simultaneous Localisation and Map Building Problem*. PhD thesis, Univ. of Sydney, 2000.
- [19] M.A. Paskin. Thin junction tree fi lters for simultaneous localization and mapping. TR UCB/CSD-02-1198, UC Berkeley, 2002.
- [20] D.B. Rubin. Using the SIR algorithm to simulate posterior distributions. In *Bayesian Statistics 3*. Oxford Univ. Press, 1988.
- [21] R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *Int. J. Robotics Research*, 5(4), 1986.
- [22] J.D. Tardós, J. Neira, P.M. Newman, and J.J. Leonard. Robust mapping and localization in indoor environments using sonar data. *Int. J. Robotics Research*, 21(4), 2002.
- [23] S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, and A.Y. Ng. Simultaneous mapping and localization with sparse extended information fi lters. *Proc. WAFR*, 2002.
- [24] R. van der Merwe, N. de Freitas, A. Doucet, and E. Wan. The unscented particle fi lter. *Proc. NIPS*, 2001.
- [25] S.B. Williams, G. Dissanayake, and H. Durrant-Whyte. An efficient approach to the simultaneous localisation and mapping problem. *Proc. ICRA*, 2002.

Appendix

The linear form (23) and (24) implies $\hat{s}_t^{[m]} = s_{t-1}^{[m]} + u_t$, $\hat{z}_t^{[m]} = \mu_{n_t,t-1}^{[m]} - s_{t-1}^{[m]} - u_t$, $G_\theta = I$, $G_s = -I$, and $Q_t^{[m]} = R_t + \Sigma_{n_t,t-1}^{[m]}$. From that we obtain for the mean and covariance (13) and (14) of the proposal distribution:

$$\Sigma_{s_t}^{[m]} = \left[(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1} + P_t^{-1} \right]^{-1} \quad (25)$$

$$\mu_{s_t}^{[m]} = -\Sigma_{s_t}^{[m]} (R_t + \Sigma_{n_t,t-1}^{[m]})^{-1} (z_t - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} + u_t) + s_{t-1}^{[m]} + u_t \quad (26)$$

The update of the landmark mean (Eq. (16) and (17)) resolves to:

$$\mu_{n_t,t}^{[m]} = \mu_{n_t,t-1}^{[m]} + \Sigma_{n_t,t-1}^{[m]} (R_t + \Sigma_{n_t,t-1}^{[m]})^{-1} (z_t - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} + u_t) \quad (27)$$

We define the error in the robot pose and landmark locations as:

$$\alpha_t^{[m]} = s_t^{[m]} - s_t \quad \text{and} \quad \beta_{n,t}^{[m]} = \mu_{n,t}^{[m]} - \theta_n \quad (28)$$

We first characterize the effect of map errors β on the pose error α :

Lemma 1. *If the error $\beta_{n,t}^{[m]}$ of the observed landmark z_t at time t is smaller in magnitude than the robot pose error $\alpha_t^{[m]}$, $\alpha_t^{[m]}$ shrinks in expectation as a result of this measurement. Conversely, if $\beta_{n,t}^{[m]}$ is larger than the pose error $\alpha_t^{[m]}$, the latter may increase, but in expectation will not exceed $\beta_{n,t}^{[m]}$.*

Proof. The expected error of the robot pose at time t is given by

$$E[\alpha_t^{[m]}] = E[s_t^{[m]} - s_t] = E[s_t^{[m]}] - E[s_t] \quad (29)$$

The first term is obtained via the sampling distribution (26), and the second term is obtained from the linear motion model (24), giving:

$$E[\alpha_t^{[m]}] = -\Sigma_{s_t}^{[m]} (R_t + \Sigma_{n_t,t-1}^{[m]})^{-1} (E[z_t] - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} + u_t) + \alpha_{t-1}^{[m]} \quad (30)$$

For linear SLAM, the expectation $E[z_t] = \theta_{n_t} - E[s_t] = \theta_{n_t} - u_t - s_{t-1}$. With that, the expression in the brackets becomes

$$\begin{aligned} E[z_t] - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} + u_t \\ = \theta_{n_t} - u_t - s_{t-1} - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} + u_t \\ = \alpha_{t-1}^{[m]} - \beta_{n_t,t-1}^{[m]} \end{aligned} \quad (31)$$

Substituting this back into (30) and subsequently substituting $\Sigma_{s_t}^{[m]}$ according to (25) gives us:

$$\begin{aligned} E[\alpha_t^{[m]}] \\ = \alpha_{t-1}^{[m]} + \Sigma_{s_t}^{[m]} (R_t + \Sigma_{n_t,t-1}^{[m]})^{-1} (\beta_{n_t,t-1}^{[m]} - \alpha_{t-1}^{[m]}) \\ = \alpha_{t-1}^{[m]} + \left[I + (R_t + \Sigma_{n_t,t-1}^{[m]}) P_t^{-1} \right]^{-1} (\beta_{n_t,t-1}^{[m]} - \alpha_{t-1}^{[m]}) \end{aligned} \quad (32)$$

The lemma follows from the fact that R_t , $\Sigma_{n_t,t-1}^{[m]}$, and P_t^{-1} are positive semidefinite, hence the inverse of $I + (R_t + \Sigma_{n_t,t-1}^{[m]}) P_t^{-1}$ is a contraction matrix. $E[\alpha_t^{[m]}]$ is larger in magnitude if and only if $\alpha_{t-1}^{[m]}$ depends on the sign of $\beta_{n_t,t-1}^{[m]} > \alpha_{t-1}^{[m]}$; however, $E[\alpha_t^{[m]}]$ cannot exceed $\beta_{n_t,t-1}^{[m]}$ in this case. *qed.*

Of particular interest is the result of observing the *anchoring landmark*, by which we mean the landmark whose location is known. Without loss of generality, we assume that this landmark is θ_1 .

Lemma 2. *If the robot observes the anchoring landmark θ_1 , its pose error will shrink in expectation.*

Proof. The anchoring landmark has zero error: $\beta_{1,t}^{[m]} = 0$, and its covariance is also zero: $\Sigma_{1,t}^{[m]} = 0$. Plugging this into (32), we get:

$$\begin{aligned} E[\alpha_t^{[m]}] &= \alpha_{t-1}^{[m]} + [I + (R_t + 0) P_t^{-1}]^{-1} (0 - \alpha_{t-1}^{[m]}) \\ &= \alpha_{t-1}^{[m]} - [I + R_t P_t^{-1}]^{-1} \alpha_{t-1}^{[m]} \end{aligned} \quad (33)$$

qed.

Finally, a lemma similar to Lemma 1 can be stated on the effect of pose errors α on map errors β . Its proof is analogous that of Lemma 1, with reverse roles of α and β .

Lemma 3. *If the pose error $\alpha_{t-1}^{[m]}$ is smaller than the error $\beta_{n_t,t}^{[m]}$ of the observed landmark z_t in magnitude, observing z_t shrinks the landmark error $\beta_{n_t,t}^{[m]}$ in expectation. Conversely, if $\alpha_{t-1}^{[m]}$ is larger than the landmark error $\beta_{n_t,t}^{[m]}$, the latter may increase, but in expectation will not exceed $\alpha_{t-1}^{[m]}$.*

Proof of Theorem. Let $\hat{\beta}_t^{[m]}$ denote landmark error that is largest in magnitude among all landmark errors at time t .

$$\hat{\beta}_t^{[m]} = \underset{\beta_{n,t}^{[m]}}{\operatorname{argmax}} |\beta_{n,t}^{[m]}| \quad (34)$$

Lemma 3 suggests that this error may increase in expectation, but only if the absolute robot pose error $\alpha_{t-1}^{[m]}$ exceeds this error in magnitude. However, in expectation this will only be the case for a limited number of iterations. In particular, Lemma 1 guarantees that $\alpha_{t-1}^{[m]}$ may only shrink in expectation. Furthermore, Lemma 2 states that every time the anchoring landmark is observed, this error will shrink by a finite amount, regardless of the magnitude of $\hat{\beta}_t^{[m]}$. Hence, $\alpha_{t-1}^{[m]}$ will ultimately become smaller in magnitude (and in expectation) than the largest landmark error. Once this has happened, Lemma 3 states that the latter will shrink in expectation every time the landmark is observed whose error is largest. It is now easy to see that both $\hat{\beta}_t^{[m]}$ and $\alpha_{t-1}^{[m]}$ converge to zero: Observing the anchoring landmark induces a finite reduction as stated in (33). To increase $\alpha_{t-1}^{[m]}$ to its old value in expectation, the total landmark error must shrink in expectation (Lemma 3). This leads to an eternal shrinkage of the total landmark error down to zero. Since this error is an upper bound for the expected pose error (see Lemma 1), we also have convergence in expectation for the robot pose error. *qed.*

Thin Junction Tree Filters for Simultaneous Localization and Mapping

Mark A. Paskin

Computer Science Division
University of California, Berkeley
Berkeley, CA 94720
paskin@cs.berkeley.edu

Abstract

Simultaneous Localization and Mapping (SLAM) is a fundamental problem in mobile robotics: while a robot navigates in an unknown environment, it must incrementally build a map of its surroundings and, at the same time, localize itself within that map. One popular solution is to treat SLAM as an estimation problem and apply the Kalman filter; this approach is elegant, but it does not scale well: the size of the belief state and the time complexity of the filter update both grow quadratically in the number of landmarks in the map. This paper presents a filtering technique that maintains a tractable approximation of the belief state as a thin junction tree. The junction tree grows under filter updates and is periodically “thinned” via efficient maximum likelihood projections so inference remains tractable. When applied to the SLAM problem, these thin junction tree filters have a linear-space belief state and a linear-time filtering operation. Further approximation yields a filtering operation that is often constant-time. Experiments on a suite of SLAM problems validate the approach.

1 Introduction

Simultaneous Localization and Mapping (SLAM)—where a robot navigating in an unknown environment must incrementally build a map of its surroundings and localize itself within that map—has attracted significant attention because it is required by many applications in mobile robotics [Thrun, 2002]. Typically the environment is idealized so that it consists of an unknown number of stationary “landmarks”; for example, in a given SLAM application these landmarks may be low-level visual features or structural features such as walls and corners. SLAM can then be viewed as the problem of incrementally estimating the locations of the robot and landmarks from noisy and incomplete observations.

One popular approach treats SLAM as a filtering problem [Smith *et al.*, 1990]. The hidden state of the system at time t is represented by a random variable $\mathbf{m}_t$ which includes $\mathbf{x}_t$, the state of the robot at time t , and $\mathbf{l}_1, \dots, \mathbf{l}_{n_t}$, the locations of the n_t landmarks observed up to time t . Thus, the size of the state vector is linear in the number of observed landmarks

and grows over time. The Kalman filter is used to compute the filtered belief state $p(\mathbf{m}_t | \text{observations to time } t)$, which in this case takes the form of a multivariate Gaussian distribution $\mathcal{N}(\mu_t, \Sigma_t)$. We regard the mean μ_t as the estimate of the map and the covariance matrix Σ_t a measure of confidence.

The Kalman filter solution is elegant, but it does not scale well to large SLAM problems. Because Σ_t explicitly represents correlations between all pairs of variables, the size of the belief state grows as $O(n_t^2)$; and because each of these correlations must be updated whenever a landmark is reobserved, the time complexity of its filter operation is also $O(n_t^2)$. This quadratic complexity renders the Kalman filter inapplicable to large SLAM problems and gives rise to the need for principled, efficient approximations.

Unfortunately, the simplest approach—discarding correlations so each variable is estimated independently—presents problems. Ignoring correlations between the robot state and the landmarks’ states leads to overconfidence and divergence because correlated observations are treated as if they conveyed independent information [Hébert *et al.*, 1996]. Furthermore, correlations between pairs of landmark states are required for quick convergence when the robot *closes a loop*, i.e., it reobserves known landmarks after an extended period of mapping unknown territory (see Figure 1). When the robot closes a loop, it reobserves landmarks whose positions are known with relative certainty; this helps the robot localize. The robot-landmark correlations translate this improved localization estimate into improved position estimates for recently-observed landmarks. The inter-landmark correlations translate these improved position estimates into improvements for the remaining landmarks on the tour.¹ Thus, the correlations give the Kalman filter a valuable property normally associated with smoothing algorithms: it can use current observations to improve estimates “from the past”.

Because quadratically many correlations are necessary to close loops, we view the challenge of scalable SLAM filtering as that of *estimating and reasoning with quadratically many correlations without quadratic time or space complexity*. In this paper, we present a novel and general approximate filtering method satisfying this criterion. Our point of departure

¹Robot-landmark correlations, which decay over time due to motion noise, cannot translate an improved localization estimate into improvements for landmarks observed in the distant past; in contrast, inter-landmark correlations do not decay over time.

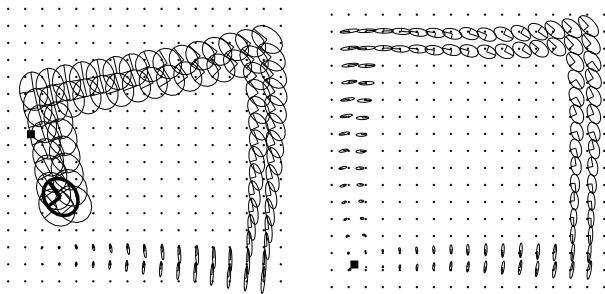


Figure 1: The robot is travelling counter-clockwise on a square path. Dots represent landmarks; the true position of the robot is shown by a square; the filter belief state is visualized using the 95% confidence ellipses of the variable marginals (bold for the robot). Left: accumulated noise and error has led to uncertain and drifted estimates for the robot and landmark positions. Right: after closing the loop, all of the position estimates improve and their confidences increase.

(in Section 2) is to view the filtered belief state of the Kalman filter as a Gaussian graphical model [Cowell *et al.*, 1999] that evolves over time; this allows us to express correlations in terms of direct dependencies (edges) and indirect dependencies (paths). Analyzing the evolution of this graphical model reveals that filter updates add edges to the graphical model, making inference more expensive. This motivates an approximation scheme in which weak or redundant edges are periodically removed to improve the complexity of inference. Note that edge removal is very different than simply discarding correlations; because other edges are left intact, paths—and thus correlations—persist between each pair of variables.

Graphical models give us valuable insight into how good approximate filters can be designed, but using them to represent the belief state presents problems. First, variable marginals like the robot’s current position would not be immediately available as they are in the Kalman filter representation; we would require inference to obtain them. Second, while it is possible to remove edges from a Gaussian graphical model using the Iterative Proportional Fitting algorithm [Speed and Kiiveri, 1986], its application in this context would be prohibitively slow. Finally, choosing edges whose removal leaves a distribution for which inference is tractable is itself a complicated process [Kjærulff, 1993].

Our solution to these problems is to use a different representation of the belief state. Exact inference in graphical models is often implemented by message passing on a *junction tree* [Cowell *et al.*, 1999]. Rather than view the junction tree algorithm as an “inference engine”, we use the junction tree itself as our representation of the belief state. This representation has many advantages: the belief state has a “built-in” inference algorithm (namely, message passing); it gives immediate access to the marginal distribution over any variable; and as we demonstrate, it gives us efficient methods of selecting edges to prune and pruning them.

To implement such a *junction tree filter*, we develop methods for updating the junction tree to reflect filtering updates in Section 3. These updates can cause the width of the junction

tree to grow, making inference more expensive; in Section 4 we present a novel “thinning” operation over junction trees called *variable contraction*. We prove that each variable contraction is a maximum likelihood projection that removes a set of edges from the corresponding graphical model. The approximation error introduced by a variable contraction can be computed efficiently, which allows us to choose which edges to remove at each time step so as to minimize the error.

In Section 5 we apply these techniques to the SLAM problem and obtain a *thin junction tree filter* (TJTF) with a $O(n_t)$ space belief state representation and a $O(n_t)$ time filter operation. By delaying the incorporation of recent evidence into the majority of the map, we can improve the filter’s time complexity; we present a method of evaluating the significance evidence has on different portions of the map, which can be used to adaptively interpolate between constant and linear-time filter operations. Empirically, we find that these adaptive filters choose constant-time updates when mapping new territory, and when closing a loop, they use time linear in the length of the loop. This is perhaps the best time complexity one would hope for in the SLAM problem, since linearly-many estimates cannot be improved in constant time. Section 6 presents the results of simulation experiments that compare TJTF to other SLAM filters and Section 7 concludes. A companion technical report contains proofs of all propositions as well as additional background, analysis, and experiments [Paskin, 2002].

1.1 Related work

Significant interest in the SLAM complexity problem has led to a number of approaches [Thrun, 2002]. For example, there are several *submap* approaches that decompose the problem into a set of small mapping problems yielding a block-diagonal landmark covariance matrix. These techniques can achieve constant time complexity, but converge slowly because information cannot pass between the submaps.

Recently, the FastSLAM algorithm [Montemerlo *et al.*, 2002]—a Rao-Blackwellized particle filter—has attracted attention because of its logarithmic time complexity. However, our experiments show FastSLAM is susceptible to divergence in large, noisy SLAM problems. We believe this is because the number of particles required for a satisfactory solution can grow exponentially over time; see [Paskin, 2002] for details.

Sparse extended information filters (SEIF) [Thrun *et al.*, 2002] can be viewed in terms of the graphical model representation described above; at each time-step, edges are removed so that a constant-time filter operation can be guaranteed. To avoid the additional complexity of inference, SEIF employs approximate inference over this approximate model. Thus, the SEIF paper provided the valuable insight that sparse graphical models can constitute an efficient solution to SLAM. Implementing this insight while avoiding additional approximation was one of the primary motivations of this work.

Each of these approaches described above uses a sublinear-time filter update, and therefore, none can improve all of the landmark estimates in a single update (like the Kalman filter). TJTF has the best of both worlds: its update step takes constant time unless the observation is significant enough to warrant a linear-time update.

Outside of the SLAM literature, there are two works that are especially relevant. Kjærulff [1993] investigated edge removal as a means of reducing the complexity of inference in graphical models. Our approach is somewhat simpler, as it operates directly on the junction tree without referring to the underlying graphical model. Kjærulff’s analysis of the approximation error inspired ours, and several of his results apply directly to our case.

Thin junction tree filtering is an *assumed density filtering* (ADF) algorithm because it periodically “projects” the filter’s belief state to some tractable family of distributions—in this case, the family of Gaussian distributions characterized by thin junction trees. This makes other work on ADF relevant, especially that of Boyen and Koller [1998], in which the belief state of a dynamic Bayesian network is periodically projected to a product-of-marginals approximation. In fact, the connection to this work is stronger: Boyen and Koller [1999] extended their earlier analysis to filters where the belief state is represented by a junction tree whose structure evolves over time; however, no algorithms were presented. To our knowledge, TJTF is the first algorithm to which this analysis applies. Here we apply TJTF to a Gaussian graphical model, but nothing prevents its application to the discrete variable networks considered by Boyen and Koller.

2 A graphical model perspective on SLAM

We begin by presenting the SLAM model and then formulating SLAM filtering in terms of graphical models.

2.1 The SLAM model

We assume a general SLAM model where in each time step the robot moves, obtains an odometry measurement of its motion, and makes several observations of landmarks. As in the Kalman filter context, we assume that the motion and measurement models are known and that they are linear-Gaussian.² The robot motion at time t is governed by

$$\mathbf{x}_{t+1} = A_t \mathbf{x}_t + \mathbf{v}_t, \quad \mathbf{v}_t \sim \mathcal{N}(\mathbf{b}_t, Q_t) \quad (1)$$

and the odometry measurement y_t at time t is governed by

$$\mathbf{y}_t = C_t \mathbf{x}_t + \mathbf{w}_t, \quad \mathbf{w}_t \sim \mathcal{N}(\mathbf{d}_t, R_t). \quad (2)$$

y_t is typically a noisy measurement of the robot’s velocities.

Landmark measurements are typically assumed to depend only upon the state of the robot and the state of the observed landmark; for example the observation may consist of the range and bearing to the landmark in the robot’s coordinate frame. If the i th landmark measurement at time t issued from landmark j , it is governed by

$$\mathbf{z}_t^i = E_t^i \mathbf{x}_t + F_t^i \mathbf{l}_j + \mathbf{u}_t^i, \quad \mathbf{u}_t^i \sim \mathcal{N}(\mathbf{g}_t^i, S_t^i) \quad (3)$$

For simplicity, we assume the correspondence between each measurement and the landmark from which it issued is known. This question of *data association*, while critically important in SLAM, is largely orthogonal to the issues we address here; in particular, the standard technique of choosing the maximum likelihood data association applies without

²When these models are not linear-Gaussian, they can be approximated as such as in the Extended or Unscented Kalman Filter.

change in our treatment. When a landmark is first observed, its state variable is added to the belief state with a uninformative (infinite variance, zero covariance) prior; the measurement update yields an informed posterior estimate of its state.

2.2 Gaussian graphical models

Under the assumptions outlined above, the filtered belief state $p(\mathbf{m}_t | y_{1:t}, z_{1:t})$ is a multivariate Gaussian distribution. The Kalman filter represents this distribution using the *moment parameters*—the mean vector μ_t and covariance matrix Σ_t . If $\mathbf{u} \sim \mathcal{N}(\mu, \Sigma)$ then its probability distribution is

$$p(\mathbf{u}) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp \left\{ -\frac{1}{2} (\mathbf{u} - \mu)^T \Sigma^{-1} (\mathbf{u} - \mu) \right\} \quad (4)$$

where d is the length of $\mathbf{u}$. In contrast, Gaussian graphical models are usually based upon the *canonical parameters*—the information vector η and matrix Λ . If $\mathbf{u} \sim \mathcal{N}^{-1}(\eta, \Lambda)$

$$p(\mathbf{u}) = \exp \left\{ a + \eta^T \mathbf{u} - \frac{1}{2} \mathbf{u}^T \Lambda \mathbf{u} \right\} \quad (5)$$

where $a = -\frac{1}{2}(d \log(2\pi) - |\Lambda| + \eta^T \Lambda^{-1} \eta)$ is the (log) normalization constant. The canonical and moment parameters are related by $\Lambda = \Sigma^{-1}$ and $\eta = \Sigma^{-1} \mu$. An advantage of the canonical parameterization is that multiplication/division of Gaussians reduces to addition/subtraction of the parameters.

Let $\{\mathbf{u}_i : i \in V\}$ be a set of random variables indexed by elements of the finite set V . We will call a subset of V a *family*. For a family $a \subseteq V$, let $\mathbf{u}_a = \{\mathbf{u}_i : i \in a\}$ be the associated set of random variables. A *potential* over a family $a \subseteq V$ is a non-negative function of $\mathbf{u}_a$. Let F be a set of families and let $\Psi = \{\psi_a : a \in F\}$ be a set of potential functions over these families. (F, Ψ) defines a distribution

$$p(\mathbf{u}) = \frac{1}{Z} \prod_{a \in F} \psi_a(\mathbf{u}_a) \quad (6)$$

when the normalizer $Z = \int \prod_{a \in F} \psi_a(\mathbf{u}_a) d\mathbf{u}$ is finite.

The *Markov graph* associated with (F, Ψ) has vertex set V and a clique of edges over each $a \in F$, i.e., there is an edge between i and j iff $\mathbf{u}_i$ and $\mathbf{u}_j$ are bound by a potential. The primary value of the Markov graph representation comes from the Hammersley-Clifford theorem, which states that s separates a from b in the Markov graph (for $a, b, s \subseteq V$) iff $\mathbf{u}_a \perp\!\!\!\perp \mathbf{u}_b | \mathbf{u}_s$ (provided $p > 0$). In other words, graph separation in the Markov graph encodes the conditional independence properties of p . Because conditional independence properties often translate into efficient inference algorithms (e.g., junction tree), the Markov graph gives good intuitions into the design of efficient approximations.

We can represent the Gaussian (5) by a Markov graph, since if we partition the vector $\mathbf{u} = \{\mathbf{u}_1, \dots, \mathbf{u}_n\}$

$$p(\mathbf{u}) = \frac{1}{Z} \prod_{i=1}^n \psi_{\{i\}}(\mathbf{u}_i) \prod_{j=i+1}^n \psi_{\{i,j\}}(\mathbf{u}_i, \mathbf{u}_j) \quad (7)$$

where

$$\psi_{\{i\}}(\mathbf{u}_i) = \exp \left\{ \eta_i \mathbf{u}_i^T - \frac{1}{2} \mathbf{u}_i^T \Lambda_{ii} \mathbf{u}_i \right\} \quad (8)$$

$$\psi_{\{i,j\}}(\mathbf{u}_i, \mathbf{u}_j) = \exp \left\{ -\mathbf{u}_i^T \Lambda_{ij} \mathbf{u}_j \right\} \quad (9)$$

and $Z = e^{-a}$ is the normalization constant. Thus, all the potentials of a Gaussian graphical model are either unary (*node potentials*) or binary (*edge potentials*). We also have the important interpretation that if $\Lambda_{ij} = 0$, then $\psi_{\{i,j\}}(u_i, u_j)$ is unity (and therefore superfluous), meaning there is no edge between i and j in the corresponding Markov graph.

2.3 Filtering in Gaussian graphical models

Filtering can be viewed as a three-step procedure: *estimation*, in which we incorporate the current time step's measurements; *prediction*, in which we augment the model with the state variables of the next time step; and *roll-up*, in which we marginalize out the state variables from the past time step. When the measurement and motion models are linear-Gaussian, the prediction and estimation steps reduce to multiplying small Gaussian potentials into the model; these updates are summarized by

Proposition 1.³ *Ignoring irrelevant normalization constants, the motion update of equation (1) can be implemented by multiplying the potential*

$$\psi \begin{pmatrix} \mathbf{x}_t \\ \mathbf{x}_{t+1} \end{pmatrix} = \mathcal{N}^{-1} \left(\begin{bmatrix} -b^T Q^{-1} A \\ b^T Q^{-1} \end{bmatrix}, \begin{bmatrix} A^T Q^{-1} A & A^T Q^{-1} \\ Q^{-1} A & Q^{-1} \end{bmatrix} \right)$$

into the model; the odometry measurement update of equation (2) can be implemented by multiplying in the potential

$$\psi(\mathbf{x}_t) = \mathcal{N}^{-1}(C^T R^{-1}(y_t - d), C^T R^{-1} C)$$

and the landmark measurement update of equation (3) can be implemented by multiplying in the potential

$$\psi \begin{pmatrix} \mathbf{x}_t \\ \mathbf{l}_j \end{pmatrix} = \mathcal{N}^{-1} \left(\begin{bmatrix} E^T S^{-1}(z_t^j - g) \\ F^T S^{-1}(z_t^j - g) \end{bmatrix}, \begin{bmatrix} E^T S^{-1} E & E^T S^{-1} F \\ F^T S^{-1} E & F^T S^{-1} F \end{bmatrix} \right)$$

The final step of filtering is roll-up, or marginalizing out the past state. The standard rule for marginalization in the canonical parameterization is given by [Cowell *et al.*, 1999]

Fact 1. *If $a = V \setminus i$ and*

$$\begin{bmatrix} \mathbf{u}_a \\ \mathbf{u}_i \end{bmatrix} \sim \mathcal{N}^{-1} \left(\begin{bmatrix} \eta_a \\ \eta_i \end{bmatrix}, \begin{bmatrix} \Lambda_{aa} & \Lambda_{ai} \\ \Lambda_{ia} & \Lambda_{ii} \end{bmatrix} \right) \quad (10)$$

then $\mathbf{u}_a \sim \mathcal{N}^{-1}(\eta_a^m, \Lambda_a^m)$ where

$$\eta_a^m = \eta_a - \Lambda_{ai} \Lambda_{ii}^{-1} \eta_i \quad (11)$$

$$\Lambda_a^m = \Lambda_{aa} - \Lambda_{ai} \Lambda_{ii}^{-1} \Lambda_{ia} \quad (12)$$

The time complexity of computing (11) and (12) is quadratic in the dimension of $\mathbf{u}_a$ and cubic in the dimension of $\mathbf{u}_i$.

The additive updates above can also be viewed as multiplying in a new potential $\mathcal{N}^{-1}(-\Lambda_{ai} \Lambda_{ii}^{-1} \eta_i, -\Lambda_{ai} \Lambda_{ii}^{-1} \Lambda_{ia})$ into the model. The *Markov blanket* $n(i)$ of $i \in V$ is the set of i 's neighbors in the Markov graph. Because missing edges in the Markov graph correspond to zeros in Λ , we can infer that this is really a potential over $n(i)$, and therefore that marginalizing $\mathbf{u}_i$ out of the model places a clique of edges over the Markov blanket of i .

³Model parameter indices are omitted for notational simplicity.

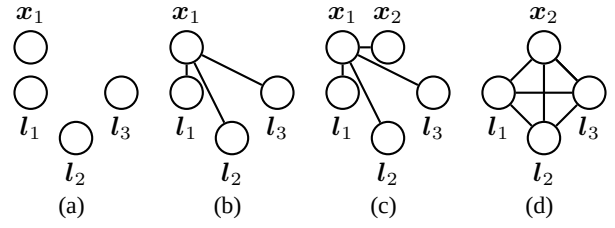


Figure 2: Example evolution of a SLAM graphical model. (a) In the initial belief state, the robot's state x_1 and the landmarks' states l_1, l_2 , and l_3 are marginally independent. (b) Observing each landmark j induces a correlation between x_1 and l_j , resulting in a new edge. (c) The prediction update adds the new robot state x_2 to the model and joins it to the current robot state x_1 . (d) The roll-up phase marginalizes x_1 out of the model, adding a clique edges over all of x_1 's neighbors.

2.4 Filtering the SLAM graphical model

Using these results we can characterize how the structure of the SLAM belief state evolves over time (see Figure 2). For each observed landmark j , we multiply a measurement potential $\psi(\mathbf{x}_t, \mathbf{l}_j)$ into the graphical model; this adds an edge between $\mathbf{x}_t$ and $\mathbf{l}_j$. Thus, after the estimation phase, the robot's state will be connected to the states of all landmarks it has observed. The prediction phase then connects $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$. Finally, the roll-up phase marginalizes out $\mathbf{x}_t$; this places a potential over the Markov blanket of $\mathbf{x}_t$, which now includes all observed landmarks and $\mathbf{x}_{t+1}$. Now the SLAM graphical model takes the form of a complete graph—i.e., *the belief state has no conditional independencies*. By induction, this will be true after every time step.

An intuition for why the graphical model becomes dense over time is valuable. When the robot measures a landmark, the landmark's state becomes directly correlated with that of the robot, and thus *indirectly* correlated with all covariates of the robot state, e.g., other landmark states. When the robot's state is eliminated from the model during roll-up, these indirect correlations must be expressed *directly* via new edges.

Importantly, these indirect correlations are often much weaker than the direct ones. Thus, even though the SLAM belief state has no true conditional independencies, there are many “approximate” conditional independencies; e.g., the landmarks observed at the beginning and end of a tour are almost independent given those observed in the middle. By removing “weak” edges from the graphical model we can enforce these approximate conditional independencies so they can be used to speed inference.

3 Junction tree filtering

As discussed in the introduction, the graphical model representation is valuable for motivating our approximate filter, but it is not an appropriate representation for its implementation. Instead, we represent the belief state of the filter using a junction tree. We begin by briefly summarizing the relevant concepts; see [Cowell *et al.*, 1999] for details.

3.1 Junction trees

Let p be a distribution of the form (6) with families F and potentials $\Psi = \{\psi_a : a \in F\}$. Let $T = (C, E)$ be an undirected graph where each vertex (or *cluster*) $c \in C$ is a subset of V ; T is a *junction tree* for p if the following three properties hold:

1. *Singly connected property*: T is a tree.
2. *Potential property*: For every family $a \in F$ there is some cluster c such that $c \supseteq a$.
3. *Running intersection property*: If $i \in V$ is present in two clusters c_i and c_j of T , it is also present in all clusters on the (unique) path between c_i and c_j .

With each edge $\{c_i, c_j\} \in E$ we associate a *separator* $s = c_i \cap c_j$; let S be the set of T 's separators.

Given a junction tree T , we can perform inference in the model p by passing messages between the clusters of T . We begin by associating with T a set of potential functions $\Phi = \{\phi_c : c \in C\} \cup \{\phi_s : s \in S\}$, one for each cluster and separator. The *charge* on T is defined to be

$$\phi_T(u) = \frac{\prod_{c \in C} \phi_c(u_c)}{\prod_{s \in S} \phi_s(u_s)} \quad (13)$$

We initialize Φ by setting all cluster and separator potentials to unity, multiplying each potential $\psi \in \Psi$ into ϕ_c for some $c \in C$ (which is possible by the potential property), and multiplying Z^{-1} into an arbitrary ϕ_c ; then $\phi_T = p$.

Let c and d be adjacent clusters with separator $s = c \cap d$. Passing a message from c to d updates the separator potential ϕ_s and the cluster potential ϕ_d as follows:

$$\phi_s^*(u_s) = \int \phi_c(u_c) du_{c-s} \quad (14)$$

$$\phi_d^*(u_d) = \phi_d(u_d) \frac{\phi_s^*(u_s)}{\phi_s(u_s)} \quad (15)$$

Importantly, these updates leave the charge (13) invariant, so $\phi_T^* = p$. Thus, we can view them as reparameterizing the distribution p . When messages are passed along every edge in both directions (in an appropriate schedule), the cluster and separator potentials are updated so that they are marginals of p over their respective variables. A junction tree in this state is called *consistent* and it can be used to obtain marginals over any set of variables that reside together in some cluster.

When T has no nonmaximal clusters, $|C| \leq |V|$ so the number of messages required for inference is bounded by $2 \cdot |V|$. In the case of a Gaussian graphical model, the cluster and separator potentials are Gaussians; if they are represented by their canonical parameters, the time complexity of passing a message is dominated by the cost of the marginalization in (14) which is implemented via (11) and (12); thus, it is at worst cubic in the size of the cluster. In sum, inference is linear in $|V|$ and cubic in the *width* of T , traditionally defined as the size of the largest cluster minus one.

3.2 Incremental junction tree maintenance

We adopt consistent junction trees as the belief state representation of our filter; i.e., the belief state will be represented by the charge (13) of a consistent junction tree. Recall from Section 2.3 that the prediction and estimation phases of the filter

update can be implemented by multiplying in small, simple potentials to the probability distribution, and that the roll-up phase is implemented by marginalizing variables out of the model. In this section we describe how to incrementally maintain a consistent junction tree under these updates.

In what follows we will make use of three nonstandard operations to restructure a consistent junction tree.

- *Cloning*: To clone a cluster c , we create a copy d , attach d to c with separator $s = c \cap d$, and set $\phi_s^* = \phi_d^* = \phi_c$.
- *Merging*: Let c and d be neighboring clusters with separator s . To *merge* d into c , we: (1) update $c^* = c \cup d$; (2) update $\phi_c^* = \phi_c \phi_d / \phi_s$; (3) swing all edges incident to d over to c ; and (4) remove d from C and s from S .
- *Pushing*: Let c and d be neighboring clusters with separator s such that $i \in c$ but $i \notin d$. To *push* i from c to d we update $s^* = s \cup \{i\}$ and $d^* = d \cup \{i\}$ and pass a message from c to d to update ϕ_s and ϕ_d . By extension we can push i from c to a nonadjacent cluster d by successive pushes along the unique path from c to d . (Any nonmaximal clusters created by pushing are subsequently merged into their subsuming neighbors.)

It is easy to check that all of these operations preserve the three structural constraints as well as the charge and consistency of a junction tree.

Multiplying in potentials

Assume our belief state is represented by a consistent junction tree T . In order to update the charge of T to reflect the multiplication of a potential $\psi_a(u_a)$ into p , we must find a cluster $c \supseteq a$ and multiply ψ_a into ϕ_c . To restore consistency, we could pass messages throughout T , but this is twice the work needed: a simple consequence of the message-passing updates (14) and (15) is that we need only *distribute evidence* from c , i.e. we must pass messages along edges in a preorder traversal from c .

If there is no cluster that covers the family a of the new potential, then we must first modify the junction tree T to create one. Draper [1995] presents several techniques to do this; in the Gaussian case the problem is somewhat simpler, since the potentials bind at most two variables. When multiplying in an edge potential $\psi(u_i, u_j)$ requires us to create a cluster covering $\{i, j\}$, we find the closest pair of clusters c and d such that $i \in c$ and $j \in d$ and push i from c to d . We then multiply $\psi(u_i, u_j)$ into $\phi_d(u_d)$ and distribute evidence from d .

It is worth noting that in several cases, conditional independencies obviate the evidence distribution step. This is a significant optimization, since message passing is by far the most expensive operation. This occurs, for example, when performing the prediction step (because x_{t+1} is an unobserved directed leaf of the graphical model and therefore does not impact the distributions of the other nodes), when observing a landmark for the first time (due to its uninformative prior), and in certain types of odometry updates.

Marginalizing out variables

Assume again that we have a consistent junction tree T representing p . As described in Section 2.3, marginalizing u_i out of p places a potential over the Markov blanket of u_i . Because the junction tree must have a cluster that covers this new

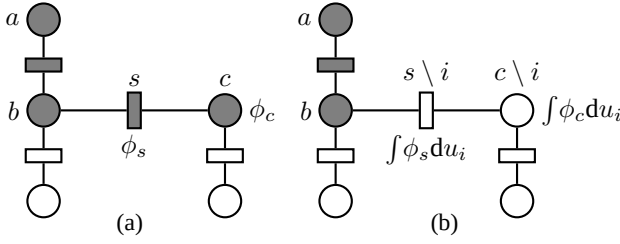


Figure 3: Illustration of variable contraction. Clusters are circles and separators are rectangles; T_i is shaded. (a) i can be contracted from a or c because they are leaves of T_i ; i cannot be contracted from b because the running intersection property would be violated. (b) Contracting i from c removes it from c and s and marginalizes u_i out of ϕ_c and ϕ_s .

potential, marginalizing out u_i is not as simple as marginalizing it out of all the cluster potentials that bind it.

Let $n(i)$ be the Markov blanket of i . Because T has the potential property, we are guaranteed that $n(i) \subseteq \cup_{c \in C, c \ni i} c$, i.e., that i 's Markov blanket is covered by the clusters containing i . (In fact, in the sequel this containment will often be strict equality.) Moreover, because T has the running intersection property, all clusters containing i constitute a subtree of T , which we denote T_i . By successively merging the clusters of T_i into each other, we can obtain a new junction tree where i resides in a single cluster $c^* \supseteq n(i)$. Marginalizing u_i out of this junction tree is simple: we remove i from c^* and marginalize u_i out of ϕ_{c^*} . It is simple to check that this operation results in a consistent junction tree for $\int p(u) du_i$.

4 Thinning the junction tree

The updates described in the previous section can cause the clusters of the junction tree to grow; in particular, the merging of clusters required by marginalizations can cause the width of the junction tree to increase quickly. The complexity of message passing scales with the width of the junction tree, and therefore our goal is to define a “thinning” operation that reduces the width (see Figure 3):

Definition 1. Let $i \in V$ appear in more than one cluster of the consistent junction tree T , let c be a leaf of T_i (the subtree of T induced by i), and let s be the separator joining c to T_i . A *variable contraction of i from c* removes i from c and s and marginalizes u_i out of ϕ_c and ϕ_s . (c is merged into its subsuming neighbor if it becomes nonmaximal.)

We now consider some properties of variable contraction.

Proposition 2. *Variable contractions preserve consistency and the singly connected and running intersection properties.*

Thus, the new junction tree is valid for some distribution, although perhaps not p : the potential property may be violated. Variable contraction is local and efficient: it requires marginalizing a variable out of one cluster potential and one separator potential, which in the Gaussian case can be accomplished in $O(|c|^2)$ time using (11) and (12). Also, variable contraction is a general method of “thinning” a junction tree:

Proposition 3. *In combination with cloning, variable contraction can reduce the width of any junction tree.*

The following proposition relates the original distribution and the distribution resulting from a variable contraction:

Proposition 4. *Let $\hat{T}$ be the junction tree obtained from the variable contraction of Definition 1. Then $\phi_{\hat{T}}$ minimizes the Kullback-Liebler divergence $D(\phi_T \parallel \cdot)$ over all distributions in which $u_i \perp\!\!\!\perp u_{c-s} \mid u_{s \setminus i}$.*

Alternatively, the probability distribution represented by $\hat{T}$ has maximum likelihood (under the original junction tree's distribution) over all distributions in which u_i is conditionally independent of u_{c-s} given $u_{s \setminus i}$. Thus, we can consider each variable contraction to be a *maximum likelihood projection* that cuts edges between i and $c - s$.

To reduce the width of a given junction tree, we should choose the variable contraction that minimizes the approximation error, which we take to be the Kullback-Liebler divergence from the original to the approximate distribution, $D(\phi_T \parallel \phi_{\hat{T}})$. This approximation error can be computed efficiently, as shown by the following result (cf. [Kjærulff, 1993, Theorem 11]):

Proposition 5. *Let $\hat{T}$ be the junction tree obtained from the variable contraction of Definition 1. Then*

$$D(\phi_T \parallel \phi_{\hat{T}}) = I(u_i; u_{c-s} \mid u_{s \setminus i}) \quad (16)$$

To compute the conditional mutual information (16) we need only the marginal $p(u_c)$. In a consistent junction tree, this marginal is simply ϕ_c , and therefore the approximation error of a variable contraction can be computed locally. When $p(u_c)$ is a Gaussian distribution, the computation is especially efficient: its cost depends only the dimension of u_i .

Proposition 6. *Let c index a set of Gaussian random variables u_c , let $s \subseteq c$, and let $i \in s$. Then*

$$I(u_i; u_{c-s} \mid u_{s \setminus i}) = \frac{1}{2} (\log |\Lambda_{ii}^c| - \log |\Lambda_{ii}^s|) \quad (17)$$

where $\Lambda^c = \text{Cov}(u_c)^{-1}$ and $\Lambda^s = \text{Cov}(u_s)^{-1}$.

Λ^c and Λ^s are parameters of the potentials ϕ_c and ϕ_s , so we can simply extract from each the sub-blocks corresponding to u_i and compute the difference of their log determinants.

5 Thin junction tree filters for SLAM

We have now assembled most of the machinery required to design a thin junction tree filter for the SLAM problem. All that remains is the logic to decide into which clusters new potentials are multiplied and also how variable contractions are employed to thin the junction tree. There are many possibilities; the method below presents a nice compromise between simplicity and performance. We then describe a refinement that can reduce the time complexity from linear to constant.

5.1 Linear-time approximate filtering

Recall from Section 3.1 that if the width of our junction tree is k , then it will require $O(|V|k^2)$ space and message passing will take $O(|V|k^3)$ time. In SLAM $|V| = n_t + 1$, so we can obtain a $O(n_t)$ space filter with a $O(n_t)$ time filter operation by periodically thinning the junction tree so that its width remains bounded by a constant k .

We start with roll-up. When the robot state $\mathbf{x}_t$ is marginalized out, we must merge all of the clusters in which it resides. In the worst case $\mathbf{x}_t$ can reside in all of the clusters, in which case our belief state will collapse to one large cluster. To prevent this, we iteratively contract $\mathbf{x}_t$ (choosing the contraction that minimizes the error (17) each time) until it resides in only one cluster c . Then, we perform the time update, which consists of multiplying the motion potential $\psi(\mathbf{x}_t, \mathbf{x}_{t+1})$ into ϕ_c , marginalizing $\mathbf{x}_t$ out of c , multiplying the odometry potential $\psi(\mathbf{x}_{t+1})$ into ϕ_c , and distributing evidence from c .

When multiplying in a landmark measurement potential $\psi(\mathbf{x}_t, \mathbf{l}_j)$ for a landmark that is currently in the model, we use the method of Section 3.2, i.e., we push $\mathbf{x}_t$ until it resides in a cluster c with $\mathbf{l}_j$. (This may increase the sizes of some clusters, but the subsequent contraction of $\mathbf{x}_t$ in the next roll-up ensures this increase in cluster size is temporary.) We then multiply $\psi(\mathbf{x}_t, \mathbf{l}_j)$ into ϕ_c and distribute evidence from c .

If instead the landmark j has not previously been observed, we must add the new variable $\mathbf{l}_j$ to the model. If the smallest cluster that contains $\mathbf{x}_t$ (call it c) can admit another variable without violating the width limit k , we add $\mathbf{l}_j$ to c and multiply $\psi(\mathbf{x}_t, \mathbf{l}_j)$ into ϕ_c . If not, then we clone c to obtain d , contract $\mathbf{x}_t$ until it resides only in d , and thin d via a sequence of greedy optimal variable contractions. A *cluster overlap* parameter h governs the size to which d is thinned, and therefore how many variables reside in the separator s that joins it to c (since in this case $s = d \setminus \mathbf{x}_t$). If h is small, d will admit more new landmark variables before another cloning is required; the trade-off is that its separator s will shrink, reducing the amount of information it can transmit.

5.2 Constant-time approximate filtering

The linear time complexity of the filter above arises mainly because we pass messages to every cluster each time we distribute evidence from some cluster c . We can get a constant-time filter operation by employing a *lazy message passing* scheme, where we distribute evidence only to constantly many nearby clusters; the approximation is that the marginals of the remaining clusters will not be conditioned on the observation. This introduces minimal error when the observation is uninformative about distant variables; this occurs, e.g., when the robot is mapping new territory. Moreover, because we are still updating the charge correctly, this approximation is temporary: at any later time a full round of message passing (taking linear time) will yield the same estimate we would have obtained by passing all messages at every time step.

Alternatively, we can interpolate between the linear-time update and this constant-time update by employing an *adaptive message passing* scheme in which messages are propagated only as long as they induce significant changes in the belief state. If we define “significant” sensibly, this scheme will take constant time when mapping new territory; when closing loops, it will take time linear in the length of the loop.

We measure the significance of a message $c_i \rightarrow c_j$ over the separator s by $D(\phi_s^* || \phi_s)$, the Kullback-Liebler divergence from the new separator marginal to the original separator marginal. In the Gaussian case this is

$$\frac{1}{2} \left[\log \frac{|\Sigma|}{|\Sigma^*|} - |s| + \text{tr}(\Sigma^{-1}(\Sigma^* + (\mu^* - \mu)(\mu^* - \mu)^T)) \right]$$

where $\phi_s = \mathcal{N}(\mu, \Sigma)$ and $\phi_s^* = \mathcal{N}(\mu^*, \Sigma^*)$. Importantly, the significance of evidence propagated from a cluster c_i to another cluster c_k (measured in this way) decreases with the distance between them in the junction tree [Kjærulff, 1993, Theorem 13]. Thus, if a message was not significant for a cluster, it need not continue the evidence distribution.

6 Experiments

Here we present a summary of our findings; the technical report contains more detail and further experiments.

We compared TJTF, the Kalman filter, and FastSLAM on large-scale SLAM simulations in which a robot moves around in a square world that is populated with uniformly distributed point-landmarks. Its motion and measurement models are all subject to significant noise and are linearized using the unscented transformation. We used two types of trajectories: a square loop (similar to that a robot mapping an indoor environment might travel) and a switchback trajectory (which could be used to map a large open area). Noise and controls were determined in advance so the robot followed the desired path and each filter received identical observations.

Figure 4 shows two examples of our simulations. The filters are evaluated by their computational cost (millions of floating point operations), localization error (the distance from the robot’s position to the filter’s estimate) and map error (the average distance from each landmark to the filter’s estimate). TJTF was run with the width limit $k = 16$, the cluster overlap $h = 4$, and adaptive message passing with the significance threshold set at 0.1 nats; FastSLAM was run with 100 particles, as recommended in [Montemerlo *et al.*, 2002].

We found that the estimation error of TJTF with maximum cluster sizes as small as 16 can be comparable with the Kalman filter, and that it gets smaller as k increases. This indicates that the edges removed by TJTF indeed carry little information; it also suggests that the estimation error of TJTF will be at least competitive with that of SEIF (an approximate form of the Kalman filter) and less than that of the submap approaches (which neglect long-distance correlations).

We also found that TJTF is good at closing loops; in Figure 4(b) we can see the localization and mapping error of TJTF suddenly drop at $t = 780$, when the robot first reobserves its starting point; also evident is a sudden increase in the computational cost: the filter is choosing to update the entire map in linear time rather than using cheaper constant-time updates. We found that FastSLAM had difficulty closing large loops (notice its divergence in Figure 4(b)) and that its estimation error in general was larger than that of TJTF.

Finally, using accurate counts of floating point operations, we found that TJTF can be as fast as FastSLAM, and that it becomes more efficient than the Kalman filter when the map contains a few hundred or more landmarks.

7 Conclusion

We believe thin junction tree filters are a promising approximation technique for dynamic probabilistic inference. First, they are flexible, in that they allow the practitioner to trade computational complexity for approximation accuracy by varying the width of the junction tree and the depth of

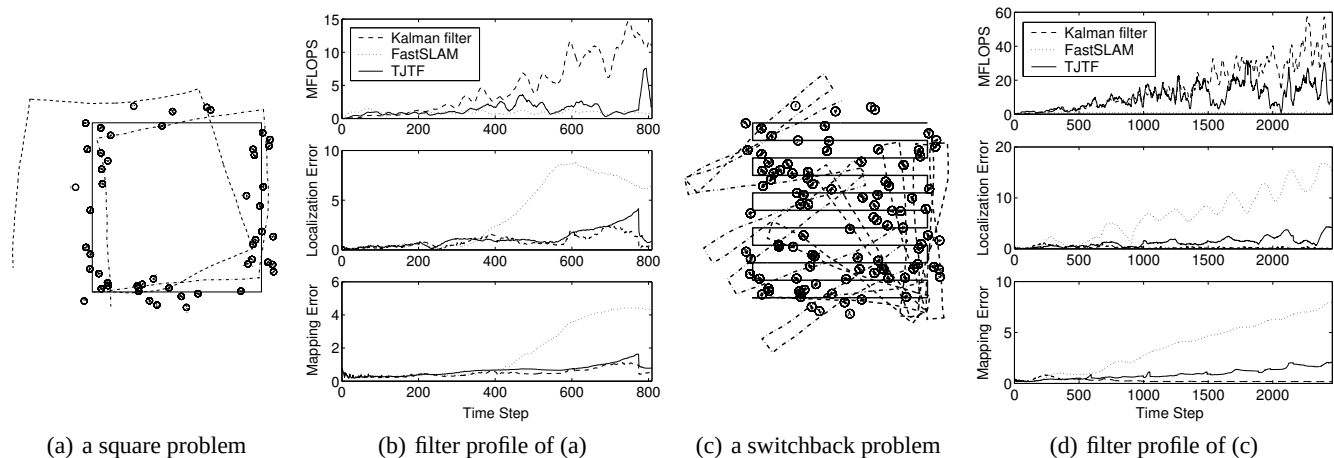


Figure 4: In (a) and (c): the solid line is the actual robot path; the dashed line is the integrated odometry; the dash-dotted line is the integrated control signal; circles are landmarks; dots are landmark observations (relative to the unknown actual viewpoint); for clarity, only some of the 1000 landmarks are plotted. In (b) and (d) the floating point counts are time-averaged for clarity.

evidence propagation. Second, the error of each local approximation can be computed exactly, giving an important indication of how trustworthy the approximate estimates are. Finally, the TJTF approximation is context sensitive in that it is not chosen in advance; rather, the approximation is chosen adaptively to minimize the approximation error.

When applied to the SLAM problem, TJTF performs competitively with the exact filter, but with superior asymptotic space and time complexity. Interestingly, the approach presented here has significant connections to both the submap approach and SEIF. First, like SEIF, TJTF cuts “weak” edges from the graphical model to speed inference; however, in TJTF we can use exact inference over this approximate model, whereas SEIF must use approximate inference. Second, the belief state of a TJTF has a natural interpretation as a coupled set of local maps, just as in the submap approaches. In particular, each cluster of the junction tree can be viewed as a submap. The TJTF formulation gives concrete semantics to the relationships between the maps, including how they must be updated, how consistency is maintained, and how the set of local maps can be determined online to minimize the approximation error subject to a complexity constraint.

Acknowledgements

I gratefully acknowledge Intel Corporation for supporting this research via an Intel Research Internship, as well as Barbara Engelhardt, Kevin Murphy, Stuart Russell, and Sekhar Tatikonda for valuable comments on a draft of this paper.

References

[Boyer and Koller, 1998] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proceedings of the 14th Annual Conference on Uncertainty in AI*, pages 33–42, 1998.

[Boyer and Koller, 1999] X. Boyen and D. Koller. Exploiting the architecture of dynamic systems. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99)*, pages 313–320, 1999.

[Cowell *et al.*, 1999] R. Cowell, P. Dawid, S. Lauritzen, and D. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, New York, NY, 1999.

[Draper, 1995] D. Draper. Clustering without (thinking about) triangulation. In *Uncertainty in Artificial Intelligence: Proceedings of the Eleventh Conference (UAI-95)*, pages 125–133, 1995.

[Hébert *et al.*, 1996] P. Hébert, S. Betgé-Brezetz, and R. Chatila. Probabilistic map learning: Necessity and difficulties. In L. Dorst, M. van Lambalgen, and F. Voorbraak, editors, *Reasoning with Uncertainty in Robotics*, volume 1093 of *Lecture Notes in Computer Science*. Springer, 1996.

[Kjærulff, 1993] U. Kjærulff. Approximation of bayesian networks through edge removals. Dept. of Mathematics and Computer Science Research Report IR-93-2007, Aalborg University, 1993.

[Montemerlo *et al.*, 2002] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, pages 593–598, 2002.

[Paskin, 2002] M. Paskin. Thin junction tree filters for simultaneous localization and mapping. Technical Report CSD-02-1198, University of California, Berkeley, September 2002.

[Smith *et al.*, 1990] R. C. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In I. J. Cox and G. T. Wilfong, editors, *Autonomous Robot Vehicles*, pages 167–193. Springer-Verlag, 1990.

[Speed and Kiiveri, 1986] T. P. Speed and H. T. Kiiveri. Gaussian markov distributions over finite graphs. *Annals of Statistics*, 14(1):138–150, March 1986.

[Thrun *et al.*, 2002] S. Thrun, D. Koller, Z. Ghahmarani, H. Durrant-Whyte, and A. Ng. Simultaneous localization and mapping with sparse extended information filters: Theory and initial results. Technical Report CMU-CS-02-112, Carnegie Mellon University, September 2002.

[Thrun, 2002] S. Thrun. Robotic mapping: A survey. In G. Lake-meyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

SATISFIABILITY

A Structure-Based Variable Ordering Heuristic for SAT

Jinbo Huang and Adnan Darwiche

Computer Science Department

University of California, Los Angeles, CA 90095

{jinbo,darwiche}@cs.ucla.edu

Abstract

We propose a variable ordering heuristic for SAT, which is based on a structural analysis of the SAT problem. We show that when the heuristic is used by a Davis–Putnam SAT solver that employs conflict-directed backtracking, it produces a divide-and-conquer behavior in which the SAT problem is recursively decomposed into smaller problems that are solved independently. We discuss the implications of this divide-and-conquer behavior on our ability to provide structure-based guarantees on the complexity of Davis–Putnam SAT solvers. We also report on the integration of this heuristic with ZChaff—a state-of-the-art SAT solver—showing experimentally that it significantly improves performance on a range of benchmark problems that exhibit structure.

1 Introduction

The class of Boolean satisfiability (SAT) problems has been of perpetual interest to researchers in many areas of computer science. Although these problems have a potentially exponential complexity, fresh techniques have continued to be proposed and implemented, allowing an increasing number of (previously) intractable problems to be solved in reasonable amounts of time.

Most existing complete SAT solvers are based on the Davis–Putnam (DP) procedure [Davis and Putnam, 1960; Davis *et al.*, 1962], which formulates the SAT problem as a systematic search problem in the space of variable instantiations, and uses depth-first search to find a solution. Various techniques are employed by DP solvers in an attempt to prune the search space, or to focus it on regions that promise earlier discovery of a solution. It has been noted that, among other things, the order in which variables are instantiated has a great effect on the resulting complexity of the SAT algorithm; see [Li and Anbulagan, 1997] for example.

We propose a new variable ordering heuristic for SAT, which is based on a structural analysis of the SAT problem. We show that when the suggested ordering heuristic is used by a DP solver that employs conflict-directed backtracking [Silva and Sakallah, 1996], it produces a divide-and-conquer

behavior in which the SAT problem is recursively decomposed into smaller problems that are solved independently. The general concept is that while the initial SAT problem may not be decomposable, it can be broken into independent subproblems after instantiating a certain number of variables that can be determined structurally. Hence, putting such a decomposing set of variables first in the ordering leads to an early decomposition of the problem. The key, however, is that this process can be repeated recursively, allowing one to recursively decompose the SAT problem down to single clauses. The other key point which we establish analytically is that it is critical for the DP solver to employ conflict-directed backtracking for the suggested order to have this recursive compositional effect.¹

By way of experimentation, we incorporated this technique in the publicly available ZChaff program [ZChaff, URL] and tested both the original and modified programs on a range of SAT benchmarks. Our results indicate a significant improvement in speed on the majority of the instances.

The rest of the paper is structured as follows. We start with a brief review of DP solvers in Section 2. We then turn in Section 3 to the proposed variable ordering heuristic, where we define it formally in terms of a graphical model, known as a dtree (decomposition tree). We then describe in Section 4 our integration of the new heuristic with ZChaff [ZChaff, URL], and provide experimental results that illustrate its effectiveness. We then consider a theoretical analysis of the proposed ordering heuristic in Section 5, where we show that when coupled with conflict-directed backtracking, the proposed ordering generates a behavior which is equivalent to a divide-and-conquer search algorithm that recursively decomposes the given SAT problem all the way to single clauses. This correspondence allows us to provide some guarantees on the complexity of the resulting search, depending on the structure of the given SAT problem. We finally close in Section 6 with some concluding remarks.

2 Davis–Putnam (DP) Search

We start by reviewing the basic DP search for the satisfiability problem. As is customary, a propositional theory is expressed in Conjunctive Normal Form (CNF). Recall that to satisfy a

¹See [Bayardo and Pehoushek, 2000] for an example where (dynamic) decomposition is used, but in the context of model counting.

Algorithm 1 $\text{sat}(\text{Theory} : C)$

```

1: if there is an inconsistent clause in  $C$  then
2:   return false (unsatisfiable)
3: if there is no uninstantiated variable in  $C$  then
4:   return true (satisfiable)
5: select an uninstantiated variable  $v$  in  $C$ 
6: return  $\text{sat}(C|_{v=\text{true}}) \vee \text{sat}(C|_{v=\text{false}})$ 

```

propositional theory in CNF each of its clauses simultaneously has to be satisfied. We will use $C = \{c_1, c_2, \dots, c_m\}$ to denote our target theory having m clauses where c_i denotes its i th clause. We will write $C|_{v=\text{true}}$ ($C|_{v=\text{false}}$) to represent the theory obtained by replacing the occurrences of variable v to *true* (*false*) in the theory C . Algorithm 1 provides a recursive description of the basic DP algorithm except that it omits the use of unit propagation which we discuss later.

This algorithm is often implemented iteratively instead of recursively, where it explores the search space by instantiating variables one at a time (we will call these instantiations decisions and the corresponding variables decision variables). Each of these decisions is pushed onto the decision stack and the theory is updated to reflect the new variable assignments. When inconsistency is discovered in the theory it goes back on the stack to the last decision whose variable has not been tried both ways, flips it, and proceeds therefrom. In case all previous decision variables have been tried both ways, it declares the theory as unsatisfiable. The theory is declared satisfiable if all variables have been successfully instantiated, or if all clauses have been subsumed (satisfied by the current, possibly partial, variable assignment).

Note that there is no need to attempt both recursive calls on Line 6 in case one of the calls succeeds. Hence, one opportunity for optimization is to first try the most promising disjunct in Line 6. Note also that we did not specify how to choose the next variable on Line 5. In fact, a different chain of decisions can often lead to a remarkable difference in complexity [Li and Anbulagan, 1997].

We describe in the next section a variable ordering heuristic based on a structural analysis of the SAT problem. We then provide an experimental and a theoretical analysis of the heuristic in later sections.

3 A Variable Group Ordering

Recall that $C = \{c_1, c_2, \dots, c_m\}$ is our target propositional theory in CNF. We have alluded to our desire of decomposing C into disconnected components. However, if we simply split C into two arbitrary subsets C_L and C_R , these two sub-theories will in general have some variables in common and hence not qualify as disconnected components.

To overcome this difficulty we propose the following method. Let V_L and V_R denote the sets of variables mentioned in C_L and C_R , respectively. We start the DP search on CNF C , but insist that it only make decisions on variables in $V_L \cap V_R$ at this stage. If *unsatisfiable* is declared during this process, we are done; otherwise all variables in $V_L \cap V_R$ will at some point have been instantiated.

Observe that the resulting sub-theories C'_L and C'_R now

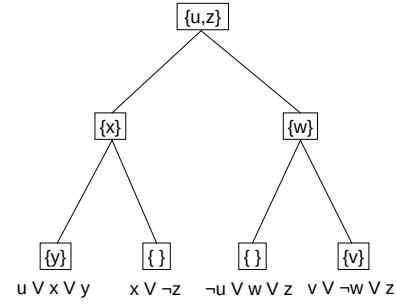


Figure 1: A dtree for a four-clause CNF.

have disjoint sets of (uninstantiated) variables: $V'_L = V_L - V_L \cap V_R$ and $V'_R = V_R - V_L \cap V_R$; they are hence completely independent sub-problems. Ordering one set of variables before the other therefore seems a good strategy from this point on. We have thus obtained a constraint on the order of variables: variables $V_L \cap V_R$ are processed first, followed by V'_L , and then V'_R .² Note that V'_L and V'_R can each be recursively broken up in a similar fashion, until all variables are partitioned into a sequence of groups, which serves as our proposed variable ordering. Within each of these groups, the algorithm is free to use any variable order.

To generate such a variable group sequence we will need to specify how the theory C is to be partitioned into sub-theories C_L and C_R , how the sub-theories in turn are to be partitioned, and so on. A graphical model known as a dtree serves nicely for this purpose, as it allows us to offer guarantees on the complexity of the resulting search [Darwiche, 2002].

A tree (decomposition tree) for a CNF C is a full binary tree whose leaves correspond to the clauses of C ; see Figure 1. Each internal node represents a subset of C corresponding to all the leaves under it; the root in particular represents the original theory C . Such a tree naturally induces a recursive decomposition scheme, partitioning a theory into two parts represented by the node's two children.

A number of properties can be defined for the nodes of a dtree [Darwiche and Hopkins, 2001].

Definition 3.1 The *variables* of an internal dtree node is the set of variables mentioned in the clauses represented by the leaves under that node; the *variables* of a leaf node is the set of variables mentioned in the clause it represents.

Definition 3.2 The *cutset* of an internal dtree node is the intersection of its children's variables, minus all its ancestors' cutsets; the *cutset* of a leaf node is its variables minus all its ancestors' cutsets.³

Figure 1 depicts a dtree for a four-clause CNF theory. The clauses are listed at the bottom below their corresponding leaves. The cutset is shown inside of each node.

²We assume the left-right order. Choice between the two orders affords another opportunity for applying heuristics, but is beyond the scope of this paper.

³A cutset is usually not defined for a leaf node in a dtree, but we extend the definition here for convenience.

Recall that each dtree node t corresponds to a part of the CNF theory; hence the variables of a dtree node are simply the variables of that part of the CNF theory; and the cutset represents all variables that need to be instantiated before the two sub-theories become disconnected (assuming that all cutsets of t 's ancestors have been instantiated). The following two concepts will help formally state our proposed heuristic for ordering variables.

Definition 3.3 *A variable group ordering of set V is a partition of variables V into an ordered sequence of subsets.*

With respect to the set of variables in Figure 1, the following is a possible variable group ordering: $\{u, v, w\}$, $\{x, y\}$, $\{z\}$; which dictates that variables $\{u, v, w\}$ should be considered before $\{x, y\}$, and $\{x, y\}$ before $\{z\}$. Note that a strict variable ordering is a special case of variable group ordering in which all groups have size one.

Definition 3.4 *The variable group ordering (v.g.o.) induced by a dtree is one obtained recursively as follows: output the cutset of the root; output the v.g.o. of its left child; output the v.g.o. of its right child; discard empty sets.*

Our proposed variable ordering scheme can now be stated as simply the variable group ordering induced by the dtree. The variable group ordering induced by the dtree in Figure 1, for example, is $\{u, z\}$, $\{x\}$, $\{y\}$, $\{w\}$, $\{v\}$.

So far we have not discussed how dtrees are to be constructed. There are obviously many distinct dtrees for any nontrivial CNF theory. To reduce the complexity of the SAT algorithm we have adopted a heuristic that calls for relatively balanced dtrees with small cutsets. A good balance ensures a near-logarithmic tree height, which allows the recursive decomposition to finish faster. And smaller cutsets lead to minimizing the number of cases that may need to be considered for each decomposition. Hence, both of these strategies will help reduce the overall complexity.

To generate dtrees with the above properties, we employ the same *hypergraph partitioning* technique as described in [Darwiche and Hopkins, 2001]. Our hypergraph for a CNF theory C is constructed by having a node for each clause in C and having a hyperedge for each variable in C connecting all nodes (clauses) that mention the variable. The hypergraph partitioning tool recursively partitions the set of nodes into two (balanced) parts while attempting to minimize the number of edges across. A small number of crossing edges translates into a small number of variables shared between two sets of clauses. Hence the resulting dtree is expected to have relatively small cutsets. The degree of balance is controlled by specifying what is called the *balance factor*. We have used a balance factor of 35 in our experiments, which tells the program not to let the ratio of the two partitions in size (large over small) exceed $\frac{50+35}{50-35} = \frac{85}{15}$. The reader is referred to [Darwiche and Hopkins, 2001] for a more detailed description of this technique.

4 Experimental Results

To evaluate the effectiveness of our proposed ordering technique, we decided to integrate it with an existing SAT solver to see whether it would improve performance on a variety of

benchmarks. Our choice of SAT solver is ZChaff [ZChaff, URL] as it has ranked first in the recent (2001 and 2002) SAT Competitions [SATEX, URL].

Our additions and modifications to ZChaff consist of 1) the package that implements dtree generation; 2) a separate chunk of code that extracts a variable group ordering from the dtree; and 3) change to the ZChaff code such that the program is given the variable group ordering as a second input (the first being the CNF) and forced to select variables from the same group, one group at a time following the specified group order. Within the same variable group, ZChaff is left to use its own heuristic, known as the Variable State Independent Decaying Sum decision heuristic [Moskewicz *et al.*, June 2001]. The new code is in the form of add-on and replacement files that compile with the original ZChaff package to produce the modified program; it is available for download at http://reasoning.cs.ucla.edu/dtree_sat/.

Our experiments were carried out on a Redhat system with a 930MHz CPU and 700MB of memory. The original ZChaff program and its modified version, which we call Dtree-ZChaff, were run on the same sets of CNFs. We report in Table 1 the comparative performance of these two programs. Note that only those instances that were solved by both programs are included. Instances on which Dtree-ZChaff succeeded but ZChaff ran out of memory are reported in Table 3; the reverse case did not happen for the given test suite. Some of our benchmarks are from [Aloul, URL]; others are from the SATLIB website [SATLIB, URL].

The times shown are in seconds and each represent the total time for all instances in the group. We generated two dtrees per CNF and chose the one with smaller width⁴. During the generation of each dtree, we repeated each hypergraph partitioning step twice and chose the smaller cut. The reported “Dtree Time” includes the time to thus obtain a dtree and that to compute a variable group ordering from it. Adding “Dtree Time” and “Dtree-ZChaff Time” gives the actual running time of Dtree-ZChaff, which is compared with that of the original ZChaff. Figures in bold highlight the benchmarks on which the proposed ordering lead to an improvement. Scores on a single-instance basis are shown in the last column. Considering whole groups, we observe that Dtree-ZChaff leads to improvement on seven out of the ten groups by a factor of between 1.1 and 7.8. Considering individual instances, it also leads to improvement on the majority of them. For a more detailed picture, we have included in Table 2 individual results on a select number of typical instances from each group.

We would like to point out that most of these CNFs are relatively “hard” instances for ZChaff—they require more than a few seconds. On instances where ZChaff finishes in a flash—such as *ii16*, *par16*, many of those in *UF250*, and some of those in *Pigeonhole*—our proposed ordering was not helpful since generating the dtree alone could take more time than that needed by the plain ZChaff to solve the SAT problem. Finally, to complete the picture, Table 3 reports on instances (not included in the first two tables) on which ZChaff ran out of memory but Dtree-ZChaff succeeded. We also note that

⁴The notion of width captures both the cutset sizes and the degree of balance; a formal definition can be found in [Darwiche, 2001].

Table 1: Overall results on all benchmarks

Benchmark	#Instances	SAT/UNSAT	Dtree Time	Dtree-ZChaff Time	ZChaff Time	Improved Instances
Pigeonhole	8	UNSAT	16	944	7368	5
FPGA-UNSAT	9	UNSAT	64	3379	26849	9
URQ	8	UNSAT	6	236	927	8
UUF250	100	UNSAT	533	32040	41292	84
UF250	100	SAT	538	3515	4378	22
FPGA-SAT	11	SAT	38	21251	18207	6
DIFP_A	14	SAT	447	54070	142755	13
DIFP_W	14	SAT	764	103726	228653	7
ii16	10	SAT	223	73	63	0
par16	10	SAT	33	23	9	0

Table 3: Instances on which ZChaff failed

Instance	#Vars/#Clauses	Dtree Time	Dtree-ZChaff Time
hole14	210/1485	6	3636
hole15	240/1816	8	14977
fpga11_20_uns_rcr	440/4220	17	7041
urq3_5	46/470	1	589
urq3_6	46/470	1	181
urq3_7	46/470	1	577
urq3_8	46/470	1	1366

Algorithm 2 $sat(Theory : C, DtreeNode : T)$

```

1: if there is an inconsistent clause in  $C$  then
2:   return false
3: if  $T$  is null then
4:   return true
5:  $V = T.cutset$ 
6: for all instantiations  $\alpha$  of  $V$  do
7:   if  $sat(C|_{\alpha}, T.left) \wedge sat(C|_{\alpha}, T.right)$  then
8:     return true
9: return false

```

the total running time of all experiments was about 2 weeks.

5 Decompositional Semantics of the Heuristic

We show in this section that when the proposed variable ordering heuristic is integrated with a DP solver that employs conflict-directed backtracking, such as ZChaff, it produces a divide-and-conquer behavior in which the solver recursively breaks down the SAT problem into smaller sub-problems that are then solved independently. Specifically, even though the solver is using a DP search method, we will show that it emulates Algorithm 2, which explicitly uses the dtree to realize a divide-and-conquer search. One significance of this correspondence is that we can now offer structure-based guarantees on the complexity of DP solvers that use such orders. Such guarantees have usually been restricted to DP solvers based on resolution [Davis and Putnam, 1960; Dechter and Rish, 1994], which have not been as influential for SAT given their intractable space complexity [Davis and Putnam, 1960; Davis *et al.*, 1962].

Algorithm 3 $sat(Theory : C, DtreeNode : T)$

```

1: if there is an inconsistent clause in  $C$  then
2:   return false
3: if  $T$  is null then
4:   return true
5: if there is no uninstantiated variable in  $T.cutset$  then
6:   return  $sat(C, T.left) \wedge sat(C, T.right)$ 
7: select an uninstantiated variable  $v$  in  $T.cutset$ 
8: return  $sat(C|_{v=true}, T) \vee sat(C|_{v=false}, T)$ 

```

Given a CNF C and a corresponding dtree, recall that each dtree node T corresponds to a subset of the CNF C . From here on, we will use $C : T$ to refer to the set of clauses in CNF C corresponding to node T . Algorithm 2 takes two inputs: a CNF C and a corresponding dtree rooted at node T . The following invariant is maintained by the algorithm: clauses $C : T$ are disconnected from other clauses. This is true trivially for the very first call to Algorithm 2, and remains true for recursive calls since cutsets associated with ancestors of node T must be instantiated by the time the recursive call is made. Algorithm 2 decomposes the clauses in $C : T$ by setting variables in the cutset of node T to some value α . Specifically, clauses $C|_{\alpha} : T$ can now be decomposed into two smaller sets, $C|_{\alpha} : T.left$ and $C|_{\alpha} : T.right$ which are solved independently. If both of these sets of clauses are solved successfully for some instantiation α , the original CNF C is declared satisfiable. Otherwise it is declared unsatisfiable.

Note that cutset variables do not need to be instantiated simultaneously as given on Line 6 of Algorithm 2— instantiating these variables one at a time leads to the variant Algorithm 3. The order in which variables are instantiated by this algorithm is indeed consistent with the group ordering induced by the given dtree, as formalized by Definition 3.4. In fact, the only key difference between Algorithm 3 and a DP solver that uses the dtree group ordering is that when all cutset variables of dtree node T are instantiated, Algorithm 3 will spawn two independent computations on $T.left$ and $T.right$. However, the DP solver will sequence these two computations, potentially creating a dependence between them. That is, if a contradiction is reached while solving the second problem $T.right$ in the sequence, the search may backtrack to variables in the first problem $T.left$, therefore

Table 2: Results on select instances

Instance	#Vars/#Clauses	SAT/UNSAT	Dtree Time (s)	Dtree-ZChaff Time (s)	ZChaff Time (s)
hole10	110/561	UNSAT	2	6	19
hole11	132/738	UNSAT	3	22	160
hole12	156/949	UNSAT	3	213	1580
fpga10_15_uns_rcr	300/2130	UNSAT	8	388	2279
fpga11_14_uns_rcr	308/2030	UNSAT	7	234	3968
fpga10_20_uns_rcr	400/3840	UNSAT	15	1859	8912
urq3_9	37/236	UNSAT	1	1	5
urq3_3	43/334	UNSAT	1	48	58
urq4_4	64/356	UNSAT	1	44	465
uuf250-010	250/1065	UNSAT	5	266	436
uuf250-030	250/1065	UNSAT	5	65	119
uuf250-050	250/1065	UNSAT	5	286	301
uuf250-070	250/1065	UNSAT	5	913	1360
uuf250-090	250/1065	UNSAT	5	29	37
uf250-010	250/1065	SAT	5	4	0
uf250-030	250/1065	SAT	5	95	55
uf250-050	250/1065	SAT	5	5	3
uf250-070	250/1065	SAT	5	1	0
uf250-090	250/1065	SAT	5	70	229
fpga10_9_sat_rcr	135/549	SAT	2	1	3
fpga12_9_sat_rcr	162/684	SAT	3	119	145
fpga13_9_sat_rcr	176/759	SAT	3	848	265
difp_19_99_arr_rcr	1201/6563	SAT	29	209	401
difp_20_99_arr_rcr	1201/6563	SAT	29	9	220
difp_21_99_arr_rcr	1453/7967	SAT	37	1393	17968
difp_19_3_wal_rcr	1755/10446	SAT	50	294	1041
difp_20_3_wal_rcr	1755/10446	SAT	50	49	838
difp_21_3_wal_rcr	2125/12677	SAT	66	4206	28495
ii16a1	1650/19368	SAT	34	0	0
par16-1	1015/3310	SAT	6	1	0

eliminating the benefit of decomposition. As we discuss next though, this is impossible to happen if the solver employs conflict-directed backtracking, as in ZChaff. Specifically, we will show that in such a case, the solver will effectively conduct two independent searches on the sub-problems corresponding to T_{left} and T_{right} . Hence, the combination of a DP solver, with our ordering heuristic, and conflict-directed backtracking corresponds to an iterative implementation of the divide-and-conquer search conducted by Algorithm 3.

We start by offering a brief description of conflict-directed (nonchronological) backtracking and refer the reader to [Silva and Sakallah, 1996] for more details. In a DP solver, a variable is instantiated either as a *decision* or as an implication through *unit propagation*. In the latter case, the assignments previously made and causing the clause to become unit are recorded as the *causes* of the implication. These recordings enable a backward construction of an *implication graph* when a conflict (contradiction) occurs. With careful analysis of the implication graph, conflict-directed backtracking chooses a decision variable such that, by backtracking to it, at least one of the causes of the conflict is eliminated. One key property of this form of backtracking is this. Suppose that after instantiating variables V , we split the SAT problem into two independent pieces: one involving variables V_L and the other involving variables V_R . We solve the first problem by instan-

tiating variables V_L , and then start solving the second problem involving variables V_R , only to realize that this problem is not satisfiable under the current settings of variables $V \cup V_L$. Conflict-directed backtracking is guaranteed *not* to backtrack to any of the variables in V_L as it is clever enough to realize that none of these variables are contributing to the contradiction. Instead, conflict-directed backtracking is guaranteed to immediately backtrack to some variable in V . Hence, even though the two problems are sequenced, the resulting behavior is similar to that of Line 6 of Algorithm 3.

We will now present an important implication of this correspondence we just established. Suppose that our CNF has a *connectivity graph* G , which is known to have a *treewidth* w .⁵ It is known that a dtree must then exist whose height is $\log n$, where n is the number of clauses, and in which every cutset has size $\leq w$ [Darwiche, 2001]. It is also known that running a divide-and-conquer algorithm, such as Algorithm 3, on such a dtree will lead to a time complexity of $O(n \exp(w \log n))$. This complexity will then apply to a DP solver that uses the group ordering derived from the men-

⁵The connectivity graph G is obtained by including a vertex to G for each CNF variable, and then adding an edge between two vertices iff their variables appear in the same clause. Treewidth is a positive integer that measures graph connectivity: the less connected the graph, the smaller the treewidth. Trees have treewidth 1.

tioned dtree, and employs conflict-directed backtracking.⁶

The time complexity of divide-and-conquer algorithms, such as Algorithm 3, can be reduced by caching of results. Specifically, each time a call is made to $\text{sat}(C, T)$, the cut-sets associated with ancestors of dtree node T are guaranteed to be instantiated to some α . Moreover, C is a CNF that results from applying the instantiation α to the original CNF we started with. It is hence possible that the call $\text{sat}(C, T)$ will be made multiple times for the same C since the original CNF may lead to the same CNF C under different variable instantiations. In fact, if all results of $\text{sat}(C, T)$ are cached and their re-computation is avoided, the complexity of the divide-and-conquer algorithm given above drops to $O(n \exp(w))$ [Darwiche, 2001]. This is a linear complexity for any CNF whose connectivity graph has a bounded treewidth.

It is interesting to note, however, that for unsatisfiable sub-problems, caching is already in place if the DP solver implements a conflict-based learning mechanism, as is the case with ZChaff. Hence, not only does the use of our ordering heuristic on top of ZChaff correspond to the divide-and-conquer Algorithm 3, but also to a version of it where unsatisfiable sub-problems are cached so they are not conquered again. There is still an opportunity though to improve performance by caching solutions of satisfiable sub-problems, but that would require a more substantial modification to a DP solver, which is beyond the scope of this paper.

We close this section by a note on static versus dynamic variable ordering heuristics. A static heuristic is one which fixes the variable order before search has started. A dynamic heuristic computes the order during search and is more promising from a pruning viewpoint since the ordering of variables can be done in the context of existing variable settings. However, dynamic heuristics can incur substantial overhead. Our proposed ordering heuristic is neither static nor dynamic, since the order of variables within a group is decided dynamically, while the order of groups is determined statically. A related static variable ordering heuristic is MINCE [Aloul *et al.*, Nov 2001], which is mostly for OBDDs but applies to SAT problems as well. MINCE is also a structural heuristic, but has a different semantics than that of our heuristic. Moreover, although the integration of MINCE with ZChaff has been attempted, no experimental results are provided to this effect in [Aloul *et al.*, Nov 2001] except for saying “Our preliminary experiments with the recently published Chaff SAT solver indicate that MINCE is not helpful on most standard benchmarks.” MINCE was shown to be quite helpful in conjunction with GRASP [Silva and Sakallah, 1996] though, but the benchmarks used appear too easy for ZChaff. In particular, MINCE did best on *pigeonhole*, *ii16* and *par16* which are among the more difficult problems tried, but they become easy problems for ZChaff; see Table 1.

⁶We are assuming here that time used by conflict-directed backtracking is dominated by the mentioned complexity, which appears to be a reasonable assumption given the low overhead recorded for this type of backtracking [Silva and Sakallah, 1996].

6 Conclusion

We have proposed a structure-based variable ordering heuristic for use by SAT solvers based on Davis-Putnam search. We have shown that when the ordering heuristic is used by a solver that employs conflict-directed backtracking, it leads to a divide-and-conquer behavior in which the SAT problem is divided recursively into smaller problems that are solved independently. Based on this decompositional behavior, one can then provide formal guarantees on the complexity of DP solvers in terms of the structure of given SAT problems. We have implemented this heuristic on top of the fastest SAT solver published—ZChaff—where we demonstrated its effectiveness in significantly boosting performance on a range of benchmarks. The proposed ordering works particularly well on the Pigeonhole, FPGA-UNSAT, and URQ instances, all of which are unsatisfiable theories with structure.

Acknowledgment

This work has been partially supported by NSF grant IIS-9988543 and MURI grant N00014-00-1-0617.

References

- [Aloul *et al.*, Nov 2001] Fadi Aloul, Igor Markov, and Karem Sakallah. Faster sat and smaller bdds via common function structure. In *ICCAD'01*.
- [Aloul, URL] <http://www.eecs.umich.edu/~faloul/benchmarks.html>.
- [Bayardo and Pehoushek, 2000] Roberto Bayardo and Joseph Pehoushek. Counting models using connected components. In *AAAI'00*.
- [Darwiche and Hopkins, 2001] Adnan Darwiche and Mark Hopkins. Using recursive decomposition to construct elimination orders, jointrees and dtrees. In *Trends in Artificial Intelligence, Lecture notes in AI, 2143*, pages 180–191. Springer-Verlag, 2001.
- [Darwiche, 2001] Adnan Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001.
- [Darwiche, 2002] Adnan Darwiche. A compiler for deterministic decomposable negation normal form. In *AAAI'02*.
- [Davis and Putnam, 1960] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *JACM*, 7:201–215, 1960.
- [Davis *et al.*, 1962] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *JACM*, (5)7:394–397, 1962.
- [Dechter and Rish, 1994] Rina Dechter and Irina Rish. Directional resolution: The Davis-Putnam procedure, revisited. In *KR'94*.
- [Li and Anbulagan, 1997] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *IJCAI'97*.
- [Moskewicz *et al.*, June 2001] Matthew Moskewicz, Conor Madihan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *DAC'01*.
- [SATEX, URL] <http://www.lri.fr/~simon/satex>.
- [SATLIB, URL] <http://www.intellektik.informatik.tu-darmstadt.de/SATLIB>.
- [Silva and Sakallah, 1996] Joao Silva and Karem Sakallah. Grasp—a new search algorithm for satisfiability. In *ICCAD'96*.
- [ZChaff, URL] <http://www.ee.princeton.edu/~chaff/zchaff.php>.

Backdoors To Typical Case Complexity

Ryan Williams *

Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213
ryanw@cs.cmu.edu

Carla P. Gomes †

Dept. of Computer Science
Cornell University
Ithaca, NY 14853
gomes@cs.cornell.edu

Bart Selman †

Dept. of Computer Science
Cornell University
Ithaca, NY 14853
selman@cs.cornell.edu

Abstract

There has been significant recent progress in reasoning and constraint processing methods. In areas such as planning and finite model-checking, current solution techniques can handle combinatorial problems with up to a million variables and five million constraints. The good scaling behavior of these methods appears to defy what one would expect based on a worst-case complexity analysis. In order to bridge this gap between theory and practice, we propose a new framework for studying the complexity of these techniques on practical problem instances. In particular, our approach incorporates general structural properties observed in practical problem instances into the formal complexity analysis. We introduce a notion of “backdoors”, which are small sets of variables that capture the overall combinatorics of the problem instance. We provide empirical results showing the existence of such backdoors in real-world problems. We then present a series of complexity results that explain the good scaling behavior of current reasoning and constraint methods observed on practical problem instances.

1 Introduction

Most interesting AI formalisms for reasoning, planning, and learning have been shown to be worst-case intractable. In the eighties and early nineties, such negative complexity results led to an extensive search for tractable subclasses of the general formalisms. Unfortunately, these tractable subclasses were often too restrictive for real-world applications. In the mid-nineties, we saw the emergence of a more practical approach to computationally hard problems in AI, with the introduction of fast satisfiability solvers and fast constraint based reasoning methods [17]. For example, in planning we saw the success of constraint-based planners, such as Graphplan [2] and SatPlan [13], and most recently, heuristic search

based planners, *e.g.*, [11; 8; 1]. Somewhat surprisingly, on practical problem instances these methods scale well beyond what one might expect based on a formal complexity analysis. In fact, current state-of-the-art SAT solvers can handle problem instances, as they arise in finite model-checking and planning, with up to a million variables and five million clauses [15]. The success of these methods appears to hinge on a combination of two factors: (1) practical combinatorial problem instances generally have a substantial amount of (hidden) tractable sub-structure, and (2) new algorithmic techniques exploit such tractable structure, through, *e.g.*, randomization and constraint learning.

These developments suggest that a standard worst-case complexity analysis does not capture well the true complexity of typical problem instances encountered in practical applications. Theoretical computer scientists have been well-aware of the limitations of worst-case complexity results and have explored alternatives, such as average-case complexity and smoothed analysis [20]. In average-case analysis, one studies the computational cost of solving problem instances drawn from a predefined problem distribution. Such an analysis can provide valuable insights, as demonstrated by the work on uniform random instance distributions (*e.g.* random k -SAT). However, the relatively basic distributions for which one can obtain average-complexity results appear to be quite far removed from the instance distributions one encounters in practice. In fact, formally defining the distribution of real-world problem instances is generally an open problem in itself. Smoothed analysis attempts to unify worst-case and average-case, but suffers from limited applicability: it works well on algorithms for problems defined over dense fields such as the simplex algorithm, but the applicability of smoothed analysis on discrete problem domains is unclear.

An alternative approach, which we will pursue in this paper, is to identify special structural properties common to known problem instances and rigorously show how clever algorithms can exploit such properties. Informal insights about what such special structure might be are currently already used in the design of, for example, branching and variable choice heuristics in combinatorial search methods. A common feature of these techniques is an understanding that different groups of variables in a problem encoding often play quite distinct roles. For example, at the highest level,

*Supported in part by an NSF Graduate Fellowship and the NSF ALADDIN Center.

†Research supported by AFOSR, Darpa, and the NSF.

one can distinguish between dependent and independent variables. The dependent or auxiliary variables are needed to obtain compact problem encodings but the true combinatorics arises from the independent variables; *e.g.*, the independent variables in an encoding of a planning domain represent the various operators applicable in a given state of the world, whereas the dependent variables encode the consequences of selecting a particular operator. A plan search technique that branches purely on the independent variables can obtain substantial speedups over search methods that do not exploit variable dependencies [4].

Another powerful intuition in the design of search methods is that one wants to select variables that *simplify the problem instance as much as possible* when these variables are assigned values. This intuition leads to the common heuristic of branching on the most constrained variable first. In terms of Boolean satisfiability, this amounts to, in effect, focusing in on the tractable substructure of the problem, namely the unit clauses (1-SAT structure) and the binary clauses (2-SAT structure). The true effectiveness of this approach arises from the fact that setting most constraint variables also simplifies higher arity clauses, which either become satisfied or in turn shrink themselves eventually to binary or unary clauses.

These general insights have been incorporated in state-of-the-art SAT and constraint solvers, and their effectiveness has been demonstrated empirically on a significant number of benchmark problems [18]. However, a more formal underpinning explaining the practical success of these strategies has been lacking. In this paper, we introduce a formal framework directly inspired by these techniques and present rigorous complexity results that support their effectiveness.

Preview of results. We first introduce the notion of “backdoor” variables. This is a set of variables for which there is a value assignment such that the simplified problem can be solved by a poly-time algorithm, called the “sub-solver”. The sub-solver captures any form of poly-time simplification procedure as used in current SAT/CSP solvers. We also consider the notion of a “strong backdoor” where any setting of the backdoor variables leads to a poly-time solvable subproblem. The set of all problem variables forms a trivial backdoor set, but many interesting practical problem instances possess much smaller backdoors and strong backdoors. We will study backdoors in several practical problem instances, and identify backdoors that contain only a fraction of the total number of variables. For example, the SAT encoding of a logistics planning problem (`logistics.d.cnf`) contains a backdoor with only 12 variables out of a total of nearly 7,000 variables. When given a set of backdoor variables of a problem instance, one can restrict the combinatorial search by branching only on the backdoor variables and thus search a drastically reduced space.

In general, finding a small set of backdoor variables for a problem instance is, however, itself a computationally hard problem. One contribution of this paper is that we formally show how the presence of a small backdoor in a problem provides a concrete computational advantage in solving it. We analyze three scenarios. First, we consider a deterministic

$B(n)$	deterministic	randomized	heuristic
n/k	small $exp(n)$	smaller $exp(n)$	tiny $exp(n)$
$O(\log n)$	$\left(\frac{n}{\sqrt{\log n}}\right)^{O(\log n)}$	$\left(\frac{n}{\log n}\right)^{O(\log n)}$	$poly(n)$
$O(1)$	$poly(n)$	$poly(n)$	$poly(n)$

Table 1: Time bounds for solving CSPs in the various scenarios considered in this work. $B(n)$ is an upper bound on the size of the smallest backdoor, where n is the number of variables in the problem. k is a fixed constant. Empirical results (Section 3) suggest that for practical instances the backdoor is often a relatively small fraction of n , *e.g.*, $n/100$, or even of size $\log n$.

scenario with an exhaustive search of backdoor sets. We show that one obtains provably better search complexity when the backdoor contains up to a certain fraction of all variables. We then show that a randomized search technique, which in effect repeatedly guesses backdoor sets, provably outperforms a deterministic search. Finally, in our third scenario we consider the availability of a variable selection heuristic, which provides guidance towards the backdoor set. This strategy can yet further reduce the search space. Table 1 gives a high-level summary of the results. By exploiting restart strategies, we can identify a polynomially solvable case when the backdoor contains at most $\log(n)$ variables. We believe that this final scenario is closest to the behavior of current effective SAT and constraint solvers. Our formal analysis also suggests several novel algorithmic strategies that warrant further empirical exploration.

2 Hidden structure: Backbones and Backdoors

Our approach and analysis applies both to SAT and CSP problems [17]. SAT is the abbreviation for the well-studied Boolean satisfiability problem. CSP is the abbreviation for the more general problem of constraint satisfaction.

A CSP problem, C , is characterized by a set $V = \{x_1, x_2, \dots, x_n\}$ of variables, with respective domains $D_1, D_2, \dots, D_n$ (which list the possible values for each variable) and a set of constraints. A constraint is defined on a subset of variables $S_i \subset V$ denoting the variables’ simultaneous legal assignments. That is, if $S_i = \{x_{i_1}, x_{i_2}, \dots, x_{i_r}\}$, then the constraint defines a subset of the Cartesian product $D_{i_1} \times \dots \times D_{i_r}$. To simplify notation, we will assume that all variables have the same domain D . We use d to denote the size of D . An assignment a is a function from variables to D . A solution to a CSP is a complete variable assignment that satisfies all constraints. A partial assignment defines the values of a subset of the variables in V . SAT is a special case of CSP with only Boolean variables ($D = \{True, False\}$) and constraints given in the form of clauses. A clause is a disjunction of literals and a literal is a Boolean variable or its negation.

We use the notation $C[v/x]$ to denote the simplified CSP obtained from a CSP, C , by setting the value of variable x to value v . (A constraint involving x is simplified by keeping only the allowed tuples that have x assigned to v .) Let $a_S :$

$S \subseteq V \rightarrow D$ be a partial assignment. We use $C[a_S]$ to denote the simplified CSP obtained by setting the variables defined in a_S . In a SAT problem, this corresponds to simplifying the formula by fixing the truth values of some of the variables.

Our goal is to capture structural properties of real world problem instances. We start by reviewing the concept of a backbone in a SAT/CSP problem, as introduced in [14]. A variable is called a backbone variable if in all solutions to the CSP the variable is assigned the same value. Such variables are also called frozen variables [6]. Backbone variables are useful in studying the properties of the solution space of a constraint satisfaction problem.

Definition 2.1 [backbone] *S is a backbone if there is a unique partial assignment $a_S : S \rightarrow D$ such that $C[a_S]$ is satisfiable.*

We contrast this variable type with the kind we introduce, backdoors. Backdoors are variable subsets defined with respect to a particular algorithm; once the backdoor variables are assigned a value, the problem becomes easy under that algorithm. (Note that contrarily to the backbone there can be different sets of backdoor variables.)

To begin our exposition of backdoors, we define the sort of algorithms we have in mind. We will call them *sub-solvers*, as they solve tractable subcases of the general constraint satisfaction problem.

Definition 2.2 *A sub-solver A given as input a CSP, C , satisfies the following:*

- (Trichotomy) *A either rejects the input C , or “determines” C correctly (as unsatisfiable or satisfiable, returning a solution if satisfiable),*
- (Efficiency) *A runs in polynomial time,*
- (Trivial solvability) *A can determine if C is trivially true (has no constraints) or trivially false (has a contradictory constraint),*
- (Self-reducibility) *if A determines C , then for any variable x and value v , then A determines $C[v/x]$.*

For instance, A could be an algorithm that solves 2-SAT instances but rejects all other instances. It is important to note that the results we will show in this paper are independent of a particular sub-solver; our results will hold for *any* A satisfying the above four properties.

In what follows, let A be a sub-solver, and C be a CSP.

We first consider a notion of “backdoor” that is suitable for satisfiable CSPs.

Definition 2.3 [backdoor] *A nonempty subset S of the variables is a backdoor in C for A if for some $a_S : S \rightarrow D$, A returns a satisfying assignment of $C[a_S]$.*

Intuitively, the backdoor corresponds to a set of variables, such that when set correctly, the sub-solver can solve the remaining problem. In a sense, the backdoor is a “witness”

to the satisfiability of the instance, given a sub-solver algorithm.¹ We also introduce a stronger notion of the backdoor to deal with both satisfiable and unsatisfiable (inconsistent) problem instances.

Definition 2.4 [strong backdoor] *A nonempty subset S of the variables is a strong backdoor in C for A if for all $a_S : S \rightarrow D$, A returns a satisfying assignment or concludes unsatisfiability of $C[a_S]$.*

In contrast to backbones which are *necessarily* set to a certain value, a (strong) backdoor S is *sufficient* for solving a problem. For example, when given the backdoor for a SAT problem, the search cost is of order $|D|^{|S|}$. (Simply check all possible assignments of S .) This means if S is relatively small, one obtains a large improvement over searching the full space of variable/value assignments.

We observe that *independent* variables are a particular kind of backdoor. As stated in [12], they are a set S of variables for which all other variables may be thought of as *defined* in terms of S . For example, a maximal subset of independent variables in a SAT encoding of a hardware verification problem is a backdoor for unit propagation, as the other variables’ values may be directly determined after setting the independent ones [19].

There are two key questions concerning backdoors:

- What is the size of the backdoor in practical problem instances?
- When taking into account the cost of searching for a backdoor set, can one still obtain an overall computational advantage in solving the CSP?

We address these two key questions below. We will first show that practical problem instances can have surprisingly small backdoors. In the subsequent section, we show how even by taking into account the cost of searching for a backdoor, one can provably obtain an overall computational advantage by using the backdoor. As we will see, the magnitude of this improvement is, of course, a function of the size of the backdoor.

3 Size of backdoors

We did an empirical study of the size of backdoors in several practical SAT instances, using the SAT solver Satz-rand, a randomized version of Satz [16]. Satz incorporates powerful variable selection heuristics and an efficient simplification strategy (*i.e.*, a good sub-solver). We modified Satz-rand to trace the variables selected for branching, and to keep track of the minimum number of variables that need to be set before Satz-rand’s simplification found a satisfying assignment efficiently. (We are currently modifying this procedure to also handle unsatisfiable instances and find strong backdoors.)

¹Observe that *any* satisfiable CSP has a backdoor of size at most $n-1$; however, we will see that significantly smaller backdoors arise in practice and give a computational advantage in search.

instance	# vars	# clauses	backdoor	fract.
logistics.d	6783	437431	12	0.0018
3bitadd_32	8704	32316	53	0.0061
pipe_01	7736	26087	23	0.0030
qg_30_1	1235	8523	14	0.0113
qg_35_1	1597	10658	15	0.0094

Table 2: Size of backdoors for several practical SAT instances.

Table 2 summarizes our results. Our instances are from a variety of domains [18]. These instances are now well within the range of the fastest current solvers, such as Chaff [15]. However, they are non-trivial and cannot be solved with the previous generation of SAT solvers, *e.g.* Tableau [3]. Clearly, the new solvers are better able to discover and exploit hidden structure, such as small backdoors. In fact, as we can see from the table, these instances have fairly tiny backdoors. That is, only a very small fraction of all variables can be used to “unlock” a satisfying assignment. We conjecture that such small backdoors occur in many other real-world problem instances.

4 Exploiting backdoors formally

We will analyze three, increasingly powerful strategies: *deterministic*, *randomized*, and *heuristic branching variable selection*. The first two are meant to work for any CSP where the instance has a small fraction of backdoor variables, with respect to the sub-solver. The randomized strategy generally outperforms the deterministic one with high probability ($1 - 1/n$, where n is the number of variables). This reflects the performance gain found in practice when backtracking SAT solvers are augmented with randomization [15; 9]. The third strategy yields tighter runtime bounds than the first two, but requires us to assume the existence of a good heuristic for choosing backdoor variables (which we find to be the case in practice).

4.1 Deterministic strategy

The deterministic procedure may be construed as a generalization of iterative deepening that runs over all possible search trees of each depth. We assume the algorithm has access to a particular sub-solver A running in $T(n)$ (polynomial) time, which defines the backdoor variables, and C is an arbitrary CSP instance.

Algorithm 4.1 Given a CSP C with n variables,

For $i = 1, \dots, n$,

For all subsets S of the n variables with $|S| = i$,

Perform a standard backtrack search (just on the variables in S) for an assignment that results in C being solved by sub-solver A .

An analogous algorithm works for finding and exploiting *strong backdoors* in a CSP to prove *unsatisfiability*: simply keep track of whether all assignments to the variables in S result in C being a contradiction (as determined by A). All

of the following we will say holds for strong backdoors and unsatisfiable CSPs under this modified algorithm.

Note the procedure uses only polynomial time for CSPs with a constant sized backdoor. We are interested in the case where a backdoor of size $B(n)$ exists, for some $B(n) < n/2$ almost everywhere. The following gives a simple runtime bound in terms of n and $B(n)$.

Theorem 4.1 *If C has a variable domain of size d and a backdoor of size at most $B(n)$, then Algorithm 4.1 runs in $O(p(n)(\frac{d \cdot n}{B(n)^{1/2}})^{B(n)})$ time, for some polynomial $p(n)$.*

Proof. The runtime is bounded by $T(n) \sum_{i=1}^{B(n)} \binom{n}{i} d^i$ (recall $T(n)$ is a runtime bound on A). Since $B(n) < n/2$, $\sum_{i=1}^{B(n)-1} \binom{n}{i} \leq q(n) \binom{n}{B(n)}$ for some polynomial $q(n)$. This and $\sum_{i=1}^{B(n)-1} d^i \leq d^{B(n)}$ imply that the runtime is dominated by the last term of the sum. Putting it together, the bound is $T(n) \sum_{i=1}^{B(n)} \binom{n}{i} d^i \leq T(n) q(n) \binom{n}{B(n)} d^{B(n)} \leq T(n) q(n) \frac{(d \cdot n)^{B(n)}}{B(n)!} \leq p(n) \frac{(d \cdot n)^{B(n)}}{B(n)^{B(n)/2}}$ asymptotically, for $p(n) = T(n) \cdot q(n)$. $\square$

The theorem implies that when small backdoors (or strong backdoors) are present, a substantial speedup almost always results. For example:

Corollary 4.1 *If C has a backdoor of size $B(n) = O(\log n)$, then C is solvable in $(\frac{n}{(\log n)^{1/2}})^{O(\log n)}$ time.*

In our exposition of heuristic branching variable selection, we will see an improvement on this (a poly-time bound). For a visual representation of the deterministic strategy’s runtime, when $d = 2$ and backdoors of size n/k are considered, see Figure 1. This graph also indicates the following corollary in the case of SAT (proof omitted):

Corollary 4.2 *For Boolean formulas with a backdoor of size at most $n/4.404$, Algorithm 4.1 solves the formula in $O(c^n)$ time, where $c < 2$.*

As we have seen in the previous section, in practice, backdoors can be quite tiny ($\approx 1/566 = 0.18\%$ of the variables, for `logistics.d.cnf`). Therefore, these results have real bearing on the improved solvability of real-world CSPs.

4.2 Randomized strategy

Better performance results from adding randomization. This speed-up formally verifies a well-known fact about real-world solvers: augmenting a solver with randomization can dramatically improve performance [9; 10].

Again, we assume a sub-solver A is on tap, with runtime $T(n)$. Let $B(n)$ be a poly-time computable function on $\mathbb{N}$ that bounds the backdoor size, and b be a parameter to be later determined. The idea is to repeatedly choose random subsets of variables that are larger than $B(n)$, searching these subsets for a backdoor.

Algorithm 4.2 Given a CSP C with n variables,

Repeat $n \left(\frac{(n/B(n)-1)}{(b-1)} \right)^{B(n)}$ times (and at least once):

Randomly choose a subset S of the n variables, of size $b \cdot B(n)$. Perform a standard backtrack search on variables in S . If C is ever solvable by A , return the satisfying assignment.

As before, an analogous algorithm works for general (satisfiable or unsatisfiable) CSPs with strong backdoors: if every leaf in the search tree ends with A reporting unsatisfiability, then the C is unsatisfiable.

The algorithm as stated requires *a priori* knowledge of $B(n)$. This may be corrected by choosing a constant $\alpha > 1$, then running the algorithm assuming a backdoor of size 1. If that fails, run it again assuming a backdoor of size α , then α^2 , α^3 , etc., until a solution to C is found.

Theorem 4.2 If C has a backdoor of size $B(n)$, Algorithm 4.2 finds a satisfying assignment with probability approaching 1.

Proof. Given there is a $B(n)$ -sized backdoor in C , the probability that a randomly chosen S of size $j \geq B(n)$, $j < n$ contains the entire backdoor is at least

$$\frac{\binom{n-B(n)}{j-B(n)}}{\binom{n}{j}} = \frac{j(j-1)\cdots(j-B(n)+1)}{n(n-1)\cdots(n-B(n)+1)} \geq \left(\frac{j-B(n)}{n-B(n)} \right)^{B(n)}.$$

Setting $j = b \cdot B(n)$, the probability that backtracking results in A finding a solution is at least $\left(\frac{(bB(n)-B(n))}{(n-B(n))} \right)^{B(n)} = \left(\frac{(b-1)}{(n/B(n)-1)} \right)^{B(n)}$, due to the self-reducibility property of A . Repeating this experiment $n \left(\frac{(n/B(n)-1)}{(b-1)} \right)^{B(n)}$ times, the algorithm succeeds with probability at least $1 - 1/n$. $\square$

One can show that the algorithm runs in $O \left(\max \{ 1, n \left(\frac{(n/B(n)-1)}{(b-1)} \right)^{B(n)} \} d^{b \cdot B(n)} T(n) \right)$ time. It remains to choose b to minimize this expression. As b depends directly on $B(n)$, we evaluate two natural cases for $B(n)$.

- When $B(n) = k \log n$ for some constant k , the runtime is $n \left(\frac{(n/(k \log n)-1)}{(b-1)} \right)^{k \log n} n^{b \cdot k'}$ for some constant k' . For large k' , the runtime is optimized when b is constant; it is $\left(\frac{n}{\log n} \right)^{O(\log n)}$, an improvement over the deterministic bound.

- When $B(n) = n/k$ for some constant k , we can show the runtime is minimized when $b = \frac{\ln(d)k}{1+\ln(d)}$, resulting in a $O(d^{n/k \cdot (1+\ln(d))/\ln(d)} (\ln(d)(k-1))^{n/k})$ time bound. For example, when $d = 2$ (the case of SAT), $b = k/2.443$ and the following holds.

Corollary 4.3 For Boolean formulas with at most $n/2.443$ backdoor variables, Algorithm 4.2 solves the formula in $O(c^n)$ time, where $c < 2$.

In the Corollary, c is a function of k . See Figure 1.

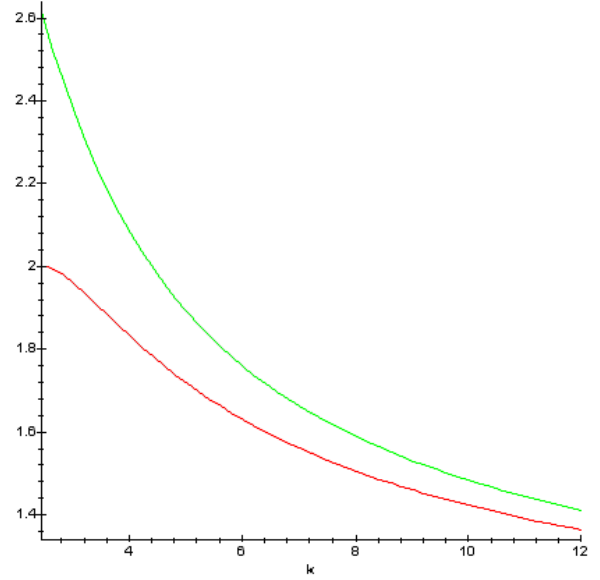


Figure 1: **Improved exponential time.** When $d = 2$ (SAT) and the size of the backbone is a constant fraction of the number of variables ($B(n) = n/k$), the runtime of Alg. 4.1 (deterministic) and 4.2 (randomized) is of the form c^n . c (vertical axis) is a function of k . The top curve gives c as a function of k for the deterministic procedure. The bottom curve gives c for the randomized procedure. Note that for $k \geq 2.443$, the randomized algorithm performs exponentially better than 2^n , whereas such an exponential improvement for the deterministic algorithm does not occur until $k \geq 4.04$.

4.3 Heuristic strategy

So far, we have considered general systematic and randomized search strategies for finding and exploiting backdoors. However, practical combinatorial solvers generally use heuristics to guide the variable selection process. As noted in the introduction, a common principle is to first branch on variables that simplify an instance the most. In effect, this means such heuristics steer the variable choice towards variables in a backdoor set. We will now formally analyze such heuristic guidance.

Restart Strategies for Heuristic Search. By incorporating the notion of a variable choice heuristic into our framework, our results are further sharpened. We consider the case where a randomized depth-first search (DFS) solver with a sub-solver A is running on an instance C having a backdoor of size B . The solver chooses variables to branch on according to a heuristic H , which has a success probability of at least $1/h$ of choosing a backdoor variable at any point in the search. We will use the notation (DFS, H, A) to denote a solver with the above properties.

Informally, a *restart strategy* is simply a policy that restarts a solver after running it for a specified amount of time, until a solution is found. Our main result here gives a condition under which a polynomial time restart strategy exists for DFS solving CSPs with small backdoors.

Theorem 4.3 If the size of a backdoor of a CSP C is $B \leq$

$\frac{c \log n}{\log h + \log d}$ for some constant c , then $(\mathbf{DFS}, H, A)$ has a restart strategy that solves C in polynomial time.

Proof. Since the probability of choosing a backdoor variable is at least $1/h$, the probability that we consecutively choose them is $1/h^B$. The probability of choosing the correct solution with only a polynomial amount of backtracking in the DFS is at least $1/d^{B-k \log n}$ for some constant k . Suppose $1/d^{B-k \log n} \cdot 1/h^B \geq 1/n^e$, for some constant e . Then by restarting the solver after every $T_A(n)$ steps (where $T_A(n)$ is the runtime of A), there is $1/n^e$ probability in each run that the backdoor will be found within a $O(n^k)$ amount of backtracking, and set correctly. From this one can show that the above inequality holds precisely when $B \leq \frac{c \log n}{\log h + \log d}$ for some constant c . $\square$

An analogous result holds for strong backdoors. It turns out that the given bound on B is asymptotically tight; we will not prove that here. When the variable domain size is constant (e.g. SAT, 3-coloring, etc.), we have the following. Let f be any poly-time computable function on the natural numbers.

Corollary 4.4 *Given CSPs with a $O(\frac{\log n}{\log f(n)})$ backdoor for which H has success probability $1/f(n)$, $(\mathbf{DFS}, H, A)$ has a polynomial time restart strategy.*

When the success probability is constant, then CSPs with $O(\log n)$ backdoors can be solved using a polynomial time restart strategy on $(\mathbf{DFS}, H, A)$. This result is the best possible in terms of backdoor size, as it would take super-polynomial time to search for a solution among $\omega(\log n)$ backdoor variables. The heuristic search runtime when $B(n) = n/k$ is still exponential, but this exponential drops dramatically as n/k decreases, even when compared to the previous two algorithms. That is, the runtime is on the order of $c^{n/k}$, where $c = d \cdot 1/h$ (recall d is the domain size and $1/h$ is success probability).

Formal Discovery of Heavy-Tails in Heuristic Search. We briefly outline our theoretical results connecting the heuristic search model described earlier with heavy-tailed runtime phenomena found empirically [9]. It was conjectured that “critically constrained” variables were a cause of the heavy-tailed behavior. We can prove that small sets of backdoor variables lead to runtime profiles that are bounded from below by heavy-tails.

The analysis that achieves this result introduces a self-similar binary tree structure, which we call a *variable choice tree*. Such trees recursively model a heuristic’s selection of backdoor variables; as more backdoor variables are chosen, the resulting search cost is much lower. It turns out that backtracking solvers with variable choice heuristics can be modeled precisely by these variable choice trees, when the size of a backdoor in the instance is small. Analysis of these trees leads to the following:

Theorem 4.4 (Heavy-tail lower bound) *If the backdoor size of an CSP C is $o(n/\log n)$, then the runtime distribution of $(\mathbf{DFS}, A, H)$ on C is lower-bounded by a Pareto-Levy distribution, when the success probability of H is constant.*

5 Conclusions

We have formalized the idea of backdoor variables in CSP/SAT instances. Backdoor variables can be used to significantly reduce the search needed in solving CSP/SAT problems. We showed that practical instances can have surprisingly small backdoors. We also provided a detailed formal analysis demonstrating that one can obtain a concrete computational advantage by exploiting such backdoors.

References

- [1] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *AIJ* 116, 2000.
- [2] A. Blum and M. Furst. Fast planning through planning graph analysis. *Proc. IJCAI-95*, 1636–1642, 1995.
- [3] J. Crawford. Tableau SAT solver. www.informatik.tu-darmstadt.de/AI/SATLIB/
- [4] A. Gerevini and I. Serina. Planning as Propositional CSP: from Walksat to Local Search. *CONSTRAINTS*, to appear.
- [5] H. Chen, C. Gomes, and B. Selman. Formal models of heavy-tailed behavior in combinatorial search, *Proc. CP’01*, 408–422, 2001.
- [6] J. Culberson and I. P. Gent. Well out of reach: why hard problems are hard, Tech Report APES-13-1999, APES Research Group, 1999.
- [7] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming, *ACM Computing Surveys* 33(3): 374–425, 2001.
- [8] H. Geffner. Perspectives on Artificial Intelligence Planning, *Proc. AAAI*, 1013–1023, 2002.
- [9] C. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems, *J. Autom. Reasoning* 24: 67–100, 2000.
- [10] C. Gomes, B. Selman, and H. Kautz. Boosting Combinatorial Search Through Randomization, *Proc. AAAI’98*, 1998.
- [11] J. Hoffman and B. Nebel. The FF Planning System, *JAIR* 14:253–302, 2001.
- [12] H. Kautz, D. McAllester, and B. Selman. Exploiting Variable Dependency in Local Search, *Proc. IJCAI*, 1997.
- [13] H. Kautz and B. Selman. Planning as satisfiability, *Proc. 10th European Conf. on AI*, 359–363, 1992.
- [14] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining Computational Complexity from Characteristic ‘Phase Transitions’, *Nature* 400: 133–137, 1999.
- [15] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver, *Proc. DAC*, 2001.
- [16] C.M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems, *Proc. IJCAI*, 366–371, 1997.
- [17] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*, 2nd ed., Prentice Hall, 2002.
- [18] SAT benchmarks. <http://www.lri.fr/~simon/satex/satex.php3>
- [19] J. P. M. Silva, *Search Algorithms for Satisfiability Problems in Combinatorial Switching Circuits*, Ph.D. Thesis, Dept. of EECS, U. Michigan, 1995.
- [20] D. Spielman and S.-H. Teng. Smoothed Analysis: Why the Simplex Algorithm Usually Takes Polynomial Time, *Proc. STOC*, 296–305, 2001.

Backbone Guided Local Search for Maximum Satisfiability*

Weixiong Zhang, Ananda Rangan and Moshe Looks

Department of Computer Science and Engineering

Washington University in St. Louis

St. Louis, MO 63130, USA

Email: zhang@cse.wustl.edu

Abstract

Maximum satisfiability (Max-SAT) is more general and more difficult to solve than satisfiability (SAT). In this paper, we first investigate the effectiveness of Walksat, one of the best local search algorithms designed for SAT, on Max-SAT. We show that Walksat is also effective on Max-SAT, while its effectiveness degrades as the problem is more constrained. We then develop a novel method that exploits the backbone information in the local minima from Walksat and applies the backbone information in different ways to improve the performance of the Walksat algorithm. We call our new algorithm backbone guided Walksat (BGWalksat). On large random SAT and Max-SAT problems as well as instances from the SATLIB, BGWalksat significantly improves Walksat's performance.

1 Introduction

Satisfiability or SAT is an archetypical combinatorial problem. A SAT involves a set of Boolean variables and a set of clauses in the form of a disjunction of literals (variables or their negations). A clause is satisfied if one of its literals is set to true, and the problem is satisfied if all clauses are satisfied. The problem is to decide if a variable assignment exists that satisfies all the clauses. When not all clauses are satisfiable, the goal is to maximize the satisfied clauses, and the problem becomes *maximum SAT* or *Max-SAT*, an optimization problem. A SAT (Max-SAT) with k literals per clause is denoted as k -SAT (Max- k -SAT). It is known that k -SAT with k greater than two is NP-complete and Max- k -SAT with k being at least two is NP-hard [7]. Max-SAT is more general and difficult to solve than SAT. Many real-world problems, such as scheduling, pattern recognition, and multi-agent cooperation and coordination [3; 6], can be formulated and solved as SAT or Max-SAT.

Recent years have witnessed significant progress made on SAT and Max-SAT along two directions. The first is the understanding of properties such as phase transitions and backbones [2; 12; 13; 17; 19]. Phase transitions refer to dramatic changes in a problem property when a critical parameter of a problem changes. The most important phase transition results on random 3-SAT are that when the ratio of the number

of clauses to the number of variables (C/V ratio) increases beyond a critical value, around 4.3, random 3-SATs begin to become unsatisfiable, and the computational cost of the problem increases abruptly [2; 12; 13]. The backbone of a problem is a set of variables that have fixed values in all optimal solutions to the problem. The hardness of a SAT or Max-SAT is determined by the backbone size of the problem [13]. In addition, the backbones of random SAT and Max-SAT also have small to large phase transitions [19]. The second direction is focused on developing efficient SAT solvers. The most noticeable results along this line are the Walksat local search algorithm [16] and its variants [11], which are among the best SAT solvers used in practice for problems such as planning.

A problem of fundamental interest and practical importance in algorithm design is how to utilize problem structure properties such as phase transitions and backbones to cope with high complexity and improve algorithm performance. The current research on this problem is limited. [15] developed a transformation method to exploit phase transitions of tree search. [5] designed a method to incorporate estimated backbone variables in a systematic search for SAT. [18] proposed a heuristic backbone sampling method for generating initial assignments for a local search using an estimated backbone. This sampling scheme will also be studied in this research.

In this paper, we develop a novel method to exploit the structure of local minima reached by an effective local search algorithm, and an effective approach to utilize the structure information to improve the performance of a local search algorithm. We focus on Max-SAT in this research.

To develop our new method, we first study the performance of Walksat, which was designed for SAT, on Max-SAT (Section 3). We show that Walksat is effective on random Max-SAT, finding many high quality local minima close to optimal. In addition, the local minima reached by Walksat appear to form large clusters around optimal solutions. Our results also show that the performance of Walksat degrades when the constrainedness of Max-SAT increases. Although we carry out this analysis for the purpose of developing a new method, the result is of interest and significance on its own. To our knowledge, Walksat has only been analyzed on some overconstrained Steiner tree problems [10].

The main contribution of this paper is an innovative method that exploits the solution structure of a combinatorial problem to improve the performance of a local search algorithm such as Walksat (Section 4). It was inspired by the recent research on phase transitions and backbones of SAT and Max-SAT [13;

*This research was supported in part by NSF grants IIS-0196057 and ITR/EIA-0113618, and in part by DARPA Cooperative Agreements F30602-00-2-0531 and F33615-01-C-1897.

19] and the performance results of Walksat in the first part of the paper. We directly apply the new method to Walksat. Nevertheless, the ideas and techniques developed here are general and applicable to other combinatorial problems and local search algorithms.

Briefly, our main ideas are to use backbone information to make biased moves in a local search, and use local minima to approximate optimal solutions for extracting backbone information. The frequencies of literals appearing in all local minima are called *pseudo-backbone frequencies*. Our new approach applies pseudo-backbone frequencies as a heuristic for selecting a variable to flip in each step of the Walksat algorithm. Such biased moves guide the search toward regions possibly having optimal solutions. The effectiveness of this method is partially due to the effectiveness of Walksat on finding optimal and high quality approximate solutions. We call the resulting algorithm *backbone guided Walksat* or BGWalksat. We demonstrate and evaluate experimentally the effectiveness of BGWalksat on large random Max-SAT instances and real problems from SATLIB [9] (Section 5).

2 The Walksat Algorithm

Walksat is a randomized algorithm. It starts by creating an initial random variable assignment (which we call **assignment generation**). It then repeatedly makes a move by selecting a variable and flipping its value from True to False or vice versa, until it finds a satisfying assignment or reaches a predefined maximal number of flips. Each such attempt is called a *try* or *restart*. The procedure repeats until a maximal number of tries has been executed.

To select a variable, the effect of flipping a variable is assessed. Flipping a variable may make some unsatisfied clauses satisfied. The number of clauses being made unsatisfied by flipping a variable is called the *break-count* of the variable at the current assignment. Walksat attempts to flip a variable with zero break-count, trying not to make the next assignment worse than the current. To find a variable with zero break-count, Walksat first selects an unsatisfied clause C , randomly, from all unsatisfied clauses. This is called **clause pick**. If C has a variable of zero break-count, Walksat then picks such a variable, randomly, from the ones that qualify. If no zero break-count variable exists in C , Walksat then makes a random choice. With probability p it chooses, randomly, a variable from the ones involved in C (called **noise pick**); and with probability $1 - p$ it picks a variable with the least break-count, breaking the tie arbitrarily if multiple choices exist (called **greedy pick**).

A flow chart of one try of Walksat is in Figure 1. The four shaded rectangles contain the random choices we will analyze. Walksat takes three parameters to run, the number of tries, the maximal number of flips in each try or *cutoff parameter*, and a probability or *noise ratio* for noise pick.

3 Performance of Walksat

We now turn to the effectiveness of Walksat on Max-SAT, particularly Max-3-SAT. We measure a local minimum in two ways. The first is the cost difference between a local minimum and an optimal solution (which may have non-zero cost). The second measure is the Hamming distance between a local minimum and the optimal solution nearest to it. We introduce the nearest Hamming distance to capture the minimal number of flips required to turn a local minimum into an optimal solution. Furthermore, to make these two quality measures on

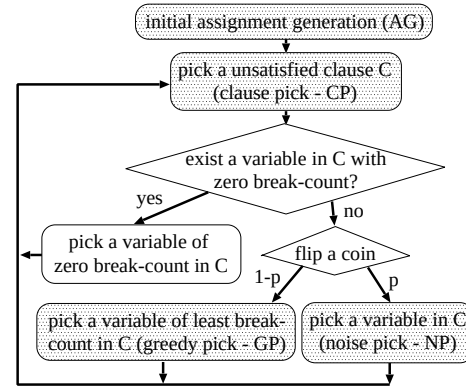


Figure 1: Main operations in a try of Walksat.

two Max-SAT instances of different sizes comparable, we normalize them. For a given problem, we divide the Hamming distance by the number of variables, resulting in the Hamming distance per variable, and divide the cost difference by the number of clauses, giving the cost difference per clause. We then use these two normalized quality measures to construct the landscape of local minima reached by Walksat.

To find the nearest optimal assignment, all optimal solutions are required, which are found by an extended DPLL algorithm [4] for SAT. Since finding all optimal solutions is expensive, we restricted ourselves to relatively small Max-SAT problems. In our experiments, we used problems of 100 variables and C/V ratios of 2.0, 4.3, 6.0 and 8.0 to capture problems in different constrainedness regions. The problems were randomly generated by uniformly picking three literals without replacement for a clause, discarding duplicate clauses. For Walksat, we set the noise ratio at 0.5, the number of tries per problem at 100, and the cutoff parameter at 10,000. We generated 1,000 problems at each C/V ratio.

The quality of local minima from Walksat are summarized in Figure 2. The X-Y planes show the correlation between the normalized Hamming distance and the normalized cost difference. The origins of the figures correspond to global minima. Each point on the X-Y plane represents a possible local minimum that may be visited by Walksat. The vertical Z axis measures the number of local minima reached by Walksat, averaged over 1,000 instances on each of which 100 tries were run.

As shown in Figures 2(a) and (b), Walksat performs well on underconstrained and critically constrained problems, in that it can find global minima very often. This is shown by the points on Z axes indicated by the arrows in the figures. Therefore, Walksat is effective at finding optimal solutions on underconstrained and critically constrained problems. However, the number of local minima which are also global minima decreases from 66,677 to 8,616, on average, as the C/V ratio increases from 2.0 to 4.3, indicating that the effectiveness of Walksat decreases. This number decreases further to 201 and 0 on overconstrained problems with C/V ratios equal to 6.0 and 8.0 (Figures 2(c) and (d)). This result indicates that Walksat becomes less effective in finding optimal solutions as problem constraints increase.

The distribution of local minima of Walksat exhibits a *bell surface* on overconstrained problems (Figures 2(c) and (d)). More importantly, the summit of such a bell surface shifts away

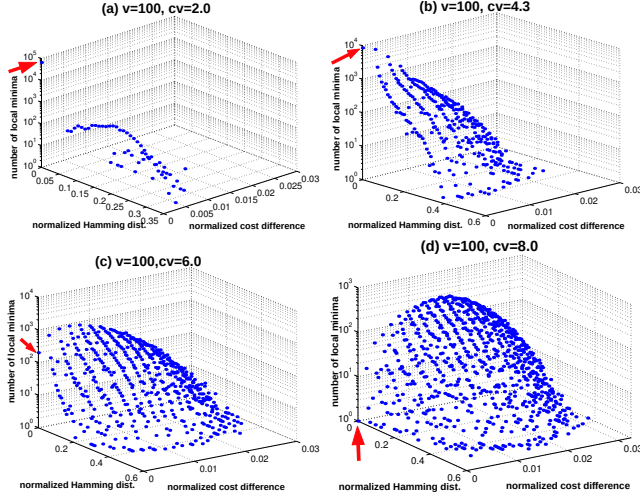


Figure 2: Landscape of local minima from Walksat.

from optimal solutions, the (0,0) point on the X-Y plane, as the problems are more constrained. As the C/V ratio increases, the number of optimal solutions decreases [19], which can become indistinguishable from a large number of suboptimal solutions. As a result, the search space becomes rugged, with more local minima in which Walksat may be trapped.

Despite the difficulty of overconstrained problems, Walksat is still fairly effective in that it is able to reach local minima close to optimal solutions. For a C/V ratio of 8.0 (Figure 2(d)), for instance, a large number of local minima landed on by Walksat (the peak point of the bell surface) have a normalized cost difference to optimal solutions of 0.014 for 100 variable problems, which is equivalent to approximately eleven more constraints violated than an optimal solution on such overconstrained problems. Since Walksat is normally executed in multiple tries, the best local minimum it can reach is usually much better than such most frequent local minima.

A closer examination of the results in Figures 2(c) and (d) indicates a nearly linear correlation between the cost difference and the Hamming distance. This observation implies that a local minimum with a small cost is more likely to share a larger common structure with an optimal solution. A similar phenomenon on the Traveling Salesman Problem was called the “big valley” [1]. We will exploit this in our new search algorithm described in the next section.

4 Backbone Guided Local Search

4.1 The main idea

If all optimal solutions to a problem were known, they could provide a useful clue to how a variable should be set. For example, if a variable is a part of the backbone, i.e., it has a fixed value in all optimal solutions, obviously the variable should be set to that value. In fact, we can extend the concept of backbone to backbone frequency. The frequency of a variable-value pair (or literal in SAT) in all optimal solutions is an indication of how often the variable should take that particular value. This can be exploited as a heuristic for selecting variables and values in a local search such as Walksat. In general, a variable-value pair will be less likely to be swapped out of the current state if it has a higher backbone frequency than another pair.

Unfortunately, exact backbone frequencies are hard to come by without finding at least one optimal solution. The idea to bypass this problem was inspired by the effectiveness of high performance local search algorithms, such as Walksat. Our analysis on the performance of Walksat (Section 3) showed that Walksat can find optimal solutions very often on under-constrained and critically constrained Max-SAT, and can also reach near optimal solutions very close to the optimal. Such near optimal local minima also form large clusters around optimal solutions. Thus, the local minima of Walksat must have common structures, which may also be shared by optimal solutions. Therefore, we can use the local minima as if they were optimal solutions to compute *pseudo-backbone frequencies* as an estimation of the true backbone frequencies. We call this general approach *backbone guided local search* or BGLS.

The above ideas can be applied to almost all local search algorithms. However, actual application is problem and algorithm specific, since local search algorithms operate very differently on different problems. In this research, we consider Max-SAT and the Walksat algorithm.

4.2 Backbone guided Walksat: BGWalksat

The application of BGLS to Max-SAT and Walksat leads to the backbone guided Walksat (BGWalksat) algorithm. Similar to Walksat, BGWalksat runs a total K tries. In addition, BGWalksat splits the total of K tries into two phases. The first, estimation phase, runs K_1 tries of original Walksat to collect local minima and compute pseudo-backbone frequencies (see Section 4.3). The second, backbone guided search phase runs $K_2 = K - K_1$ tries, in which variable selection is guided by pseudo-backbone frequencies. The newly discovered local minima can also be added to the pool of all local minima to update the pseudo-backbone frequencies.

We now consider backbone guided moves or biased moves. As discussed in Section 2 and shown in Figure 1, Walksat makes five uniformly random choices. The first is the random initial assignment generation. Instead of uniformly randomly choosing a variable and setting it to an arbitrary value, we can view the pseudo-backbone frequencies as an approximate probability distribution of variable assignments and take samples of variable assignments as initial assignments from this distribution. This scheme was called heuristic backbone sampling in [18].

The second random choice in Walksat is the random clause pick for choosing an unsatisfied clause. To apply the idea of backbone frequencies, we extend it to the concept of *pseudo-backbone clause frequencies*, which measure the frequencies that clauses are satisfied in all local minima. When picking a clause from the set of unsatisfied clauses, we choose one based on its pseudo-backbone clause frequency. For example, given three unsatisfied clauses C_1, C_2 and C_3 with pseudo-backbone frequencies 0.3, 0.05 and 0.15, respectively, we choose C_1 with probability $0.3/(0.3+0.05+0.15) = 0.6$, C_2 with probability $0.05/0.5 = 0.1$ and C_3 with probability $0.15/0.5 = 0.3$.

The remaining three random choices involved selecting a variable, uniformly, from a particular set. The first choice picks a variable from the set of zero break-count variables in the chosen clause C (the unshaded rectangle in Figure 1), the second directly from the variables involved in C , and the third from the set of variables in C with the least break-count. Rather than picking one variable uniformly from a particular set, we make a biased selection based on variables’ pseudo-backbone

frequencies. Consider an example of three literals $x_1, \neg x_2$ and x_3 in the chosen clause C with backbone frequencies 0.1, 0.3 and 0.6, respectively. We want to flip x_1 more than x_3 because literal $\neg x_1$ appears more often than $\neg x_3$ in all local minima (frequency 0.9 versus 0.4). Therefore, we pick $x_1, \neg x_2$ and x_3 with frequencies $0.9/(0.9+0.7+0.4) = 0.45, 0.7/2.0 = 0.35$ and $0.4/2.0 = 0.2$, respectively.

4.3 Pseudo-backbone frequencies

The performance of backbone guided local search depends largely upon the quality of pseudo-backbone frequencies. There are at least two ways to compute pseudo-backbone frequencies. The first and simplest method is to consider all the available local minima as if they were global optima, and take the frequency of a variable-value pair $l = (x_i, v_i)$ that appears in all local minima S as its pseudo-backbone frequency. Specifically, we have frequency $p(l) = \sum_{s_i \in S, l \in s_i} 1/|S|$. We call this method *average counting*.

It is imperative to note that not all local minima are of equal quality. A lower quality local minimum usually contains less backbone variables than one of a higher quality. Therefore, the former is less reliable and should contribute less to the pseudo-backbone frequencies than the latter. Thus, we must discount the contribution of a local minimum based on its cost. If a local minimum s_i has cost c_i , we can compute the pseudo-backbone frequency of $l = (x_i, v_i)$ as $p(l) = (\sum_{s_i \in S, l \in s_i} 1/c_i) / (\sum_{s_i \in S} 1/c_i)$. In other words, the pseudo-backbone frequency of l is reciprocally weighted by the costs of the local minima that l was involved with. We call this method *cost reciprocal average counting*.

4.4 Walksat and BGWalksat with Dynamic Noise

One limitation of the Walksat family of algorithms is their dependence on a manually set noise parameter. So far, two mechanisms have been proposed to resolve this issue in SAT. Auto-Walksat [14] uses a preliminary probing stage to estimate the optimal value for the noise parameter, while Walksat with dynamic noise [8] automatically adjusts the noise level as the search progresses. We have successfully combined our method with the dynamic noise strategy, for SAT and Max-SAT.

The idea of dynamic noise is simple: start search with the noise parameter equal to zero, and examine the number of violations in the current state every $\theta \cdot m$ flips, where m is the number of clauses in the instance. If the number of violations has not decreased since the last time we checked (n flips ago), the search is assumed to have stagnated, and the noise level is increased to $wp + (1 - wp) \cdot \phi$, where wp is the current noise level. Otherwise, the noise level is decreased to $wp - wp \cdot 2\phi$. The discrepancy between the formulas for increasing and decreasing the noise level are based on the observations of how Walksat behaves when the noise parameter is too high, compared with how it behaves when the parameter is too low [8]. Following [8], we have set $\theta = 1/6$ and $\phi = 0.2$, which have been found to be effective over a wide range of SAT and Max-SAT instances.

Dynamic noise was designed and tested with Walksat's cutoff parameter set to infinity; i.e., no random restarts. This is the setting we use for Walksat with dynamic noise for all of our experiments in the next section. Unfortunately, this setting is incompatible with backbone guided Walksat, which requires random restarts in order to construct the pseudo-backbone. To overcome this, we have devised "compromise" parameter settings, which allow dynamic noise to function effectively, while

still constructing the pseudo-backbone. Specifically, we run Walksat with dynamic noise for 30 short runs to construct the pseudo-backbone, followed by 7 long runs of backbone guided local search, each of which is 10 times as long as a short run.

5 Experimental Results

We now analyze the performance of BGWalksat on Max-SAT. The benchmark suite for empirical evaluation consists of two different types of problems: randomly generated problems and problems converted from real-world applications in SATLIB [9]. We included satisfiable and unsatisfiable problem instances. We used Walksat implementation from Henry Kautz and developed BGWalksat on top of it.

5.1 Random ensembles

In these experiments, we investigate the effects of biased moves on random choices in Walksat (Section 2). There are five places where Walksat makes uniformly random choices (the five rectangles in Figure 1). Our experimental results showed that the biased random choice of picking a variable of zero break-count in a selected clause C (the unshaded rectangle in Figure 1) has almost no effect. The reason is that when a zero break-count variable exists, there is usually one available, meaning that a very few such biased moves occur. We will not consider this biased random choice in the rest of the paper.

In our experiments, we generated random MAX-3-SAT instances with 2,000 variables and three different C/V ratios of 2.0, 4.3 and 8.0. We generated 100 instances for each of the C/V ratios. We also tested various noise ratios, from 0 to 0.9 with an increment of 0.1, as well as Walksat with dynamic noise. Each run, regardless of its configuration, was allowed a total of one million flips.

For Walksat with static noise, the cutoff parameter was set to 10,000 flips (so the algorithm was run from 100 different starting points). We tested different ratios of the lengths of the backbone estimation phase and the actual backbone search phase in BGWalksat. We found that a ratio of about 0.3 seems to provide the best results. In the results shown below, Walksat was run on the first 30 tries to collect the initial set of local minima and obtain the pseudo-backbone frequencies. BGWalksat was run on the remaining 70 tries. New local minima were added to update pseudo-backbone frequencies every 5 tries.

We experimentally compared the two methods for computing pseudo-backbone frequencies (Section 4.3). The average counting (AC) method is generally worse than the cost reciprocal average counting (CRAC) method on random problems. Due to space limitations here, we will only report the results from the CRAC method below.

There is little impact of biased moves when the C/V ratio is 2.0, because Walksat is able to find optimal solutions in the first 30 tries in almost all problems. Figure 3 shows the results on random Max-SAT with 2,000 variable and C/V ratios of 4.3 and 8.0, comparing Walksat and BGWalksat with biased noise pick, biased greedy pick, and bias moves combining biased noise pick, greedy pick, clause pick, and assignment generation. The error bars in the figures correspond to 95% confidence intervals. As shown, biased greedy pick is effective at low noise levels, but leads to degraded performance on higher noise levels. Biased noise pick, on the other hand, has a greater effect as the noise level increases, due to the direct relationship between the noise level and the frequency of noise picking. Biased assignment generation and clause pick on their own, have

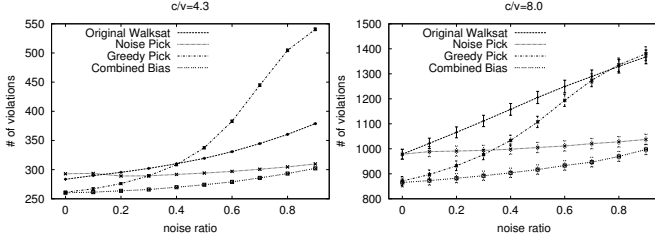


Figure 3: Quality improvement on random Max-SAT using biased noise pick, biased greedy pick, and the combination of biased noise pick, greedy pick, clause pick, and assignment generation (combined bias).

little effect of Walksat’s performance (data not shown). However, when combined with other strategies, they can provide improvements, as shown.

We also tested Walksat and BGWalksat with dynamic noise on the same instances, with good results. This is fairly unsurprising, since on this class of instances, Walksat performs best with a low noise ratio, and dynamic noise begins with the noise ratio set to zero, and only raises it as needed. We found that the noise parameter fluctuated at low levels, with an average of 0.2 for both $C/V=8$ and $C/V=4.3$. The noise level never exceeded 0.6 at any time. The solutions found by Walksat with dynamic noise were as good as the best solutions found by Walksat with static noise for $C/V=8$, and only about 3.5% worse for $C/V=4.3$. When combining dynamic noise with BGWalksat, the average noise level was slightly lower, giving 0.13 for $C/V=8$ and 0.15 for $C/V=4.3$, due to the increased number of random restarts. This is because after every restart there is a period of rapid improvement, when the noise level remains at zero. The performance of BGWalksat with dynamic noise is similar to that of BGWalksat with best static noise for $C/V=8$ and slightly worse than that for $C/V=4.3$.

5.2 Real problem instances

We now investigate how BGWalksat improves upon Walksat on the problems from SATLIB [9], when used in conjunction with dynamic noise. The test problems include SAT-encoding instances from a random hard graph coloring problems and SAT-encoded blocks world planning, bounded model checking, and the all interval series problem. The details of these problems can be found on the website. We only chose problems with more than 350 variables, and discarded those that can be easily solved by Walksat and BGWalksat.

We considered satisfiable and unsatisfiable problems. We ran both Walksat and BGWalksat, both using dynamic noise, with a total of 10 million flips (compared with 1 million for our results for random instances). Interestingly, BGWalksat exhibited superior performance, over a wide range of real instances, with some of the methods of biased-moves utilizing the pseudo-backbone disabled. For the results presented here, we use backbone-guided noise pick and backbone-guided clause pick, with the average-counting (AC) method. Results are for twenty runs for each algorithm and instance. Experiments with the best static noise (not shown) produced similar results.

In viewing the results, we found it useful to divide the satisfiable instances into two categories, the easier instances, which were solved at least once (Table 1), and the harder ones, which

Table 1: Walksat vs. BGWalksat on easier satisfiable problems. Walksat and BGWalksat are the runs resulting in a satisfying solutions (out of 20) by these algorithms. The better results are underlined and in bold.

problem	#Var	#Clause	Walksat	BGWalksat
bwJarge.c	3016	50457	1	<u>2</u>
bwJarge.d	6325	131973	<u>1</u>	0
par8-1	350	1149	6	<u>19</u>
par8-2	350	1157	6	<u>19</u>
par8-3	350	1171	7	<u>17</u>
par8-4	350	1155	0	<u>16</u>
par8-5	350	1171	1	<u>15</u>
qg1-08	512	148957	8	<u>12</u>
qg2-08	512	148957	1	<u>4</u>
qg3-08	512	10469	11	<u>20</u>
qg6-09	729	21844	0	<u>5</u>
qg7-09	729	22060	4	<u>5</u>
g125.17	2125	66272	5	0
g250.29	7250	454622	<u>4</u>	2

were not solved by either method, in any of their runs (Table 2). Results for unsatisfiable instances are presented in Table 3.

As the results show, BGWalksat significantly outperforms Walksat in most cases. On easier satisfiable instances (Table 1), BGWalksat finds more satisfying solutions than Walksat for all parity (*par*) and quasigroup (*qg*) classes, and produces similar results to Walksat on blocksworld instances. On harder satisfiable instances (Table 2), BGWalksat outperforms Walksat in all but two of 34 instances, where it is less than half a percent worse. In contrast, the overall average gain is 30%, and is over 50% in 11 of them. On unsatisfiable instances (Table 3), BGWalksat produces impressive gains on longmult instances, and on ssa6288-047, with an overall average gain of 20%. On unsatisfiable quasigroup instances (not shown), performance was similar to that of Walksat. Performance of BGWalksat is never more than 10% worse than Walksat on any of the unsatisfiable instances we have studied.

The most glaring failure of BGWalksat is on the instances *g125.17* and *g250.29*. These instances are SAT-encoded graph coloring problems, and serve to illustrate an important point. As described in Section 3, we believe that our method is effective because it exploits the “big valley” structure of the solution space. However, graph coloring problems exhibit a particular type of symmetry in their solution structures which is opaque to local search methods such as Walksat. For example, coloring all red nodes green, and all green nodes red, results in an identical solution, with a radically different encoding. Thus, there is not a single “big valley”, but several, which can bury the true backbone information and thus lead to degraded performance. Presumably, BGWalksat’s performance will suffer on all instances with this type of symmetry.

6 Conclusions

We developed a novel and general method that exploits backbone information for improving the performance of a local search algorithm. We demonstrated and analyzed the new method, called BGWalksat, on Max-SAT using the Walksat algorithm, with both static and dynamic noise strategies. The main ideas are to extract backbone information from local minima and use it directly to fix the discrepancy between the current state and optimal solutions, so as to guide Walksat toward the regions of the search space more likely to contain optimal solutions. In comparison, almost all existing local search methods focus on the costs of the states in the search space. There-

Table 2: Walksat vs. BGWalksat on harder satisfiable problems. Walksat and BGWalksat are the average numbers of violations in the best solutions found by the algorithms for a given problem, averaged over 20 runs. Gain is the percentage improvement of BGWalksat over Walksat. The better results are underlined and in bold.

problem	#Var	#Clause	Walksat	BGWalksat	gain (%)
bmc-ibm-1	9685	55870	25.3	<u>4.15</u>	83.60
bmc-ibm-2	3628	14468	5.4	<u>1.2</u>	77.78
bmc-ibm-3	14930	72106	115.25	<u>19.7</u>	82.91
bmc-ibm-4	28161	139716	118.15	<u>38.9</u>	67.08
bmc-ibm-5	9396	41207	12.95	<u>1.25</u>	90.35
bmc-ibm-6	51654	368367	358.25	<u>103.6</u>	71.08
bmc-ibm-7	8710	39774	17.4	<u>6.4</u>	63.22
bmc-galileo-8	58074	294821	65.65	<u>15.5</u>	76.39
bmc-galileo-9	63624	326999	95.95	<u>17.3</u>	81.97
bmc-ibm-10	61088	334861	406.15	<u>162.45</u>	60.00
bmc-ibm-11	32109	150027	439.8	<u>358.45</u>	18.50
bmc-ibm-12	39598	19477	554.65	<u>445.25</u>	19.72
bmc-ibm-13	13215	6572	88.05	<u>2.7</u>	96.93
f2000	2000	8500	2.2	<u>2.05</u>	6.82
par16-1-c	317	1264	5.45	<u>5.35</u>	1.83
par16-1	1015	3310	10.45	<u>9.45</u>	9.57
par16-2-c	349	1392	6.2	<u>5.9</u>	4.84
par16-2	1015	3374	10.6	<u>10.4</u>	1.89
par16-3-c	334	1332	6	<u>5.65</u>	5.83
par16-3	1015	3344	10.45	<u>9.75</u>	6.70
par16-4-c	324	1292	6.15	<u>5.5</u>	10.57
par16-4	1015	3324	10.4	<u>9.55</u>	8.17
par16-5-c	341	1360	6.25	<u>6.05</u>	3.20
par16-5	1015	3358	10.45	<u>9.85</u>	5.74
par32-1-c	1315	5254	21.7	<u>20.85</u>	3.92
par32-1	3176	10277	30.95	<u>30.25</u>	2.26
par32-2-c	1303	5206	21.15	<u>21.2</u>	-0.24
par32-2	3176	10253	32.1	<u>28.35</u>	11.68
par32-3-c	1325	5294	22.05	<u>21.3</u>	3.40
par32-3	3176	10297	32.95	<u>28.55</u>	13.35
par32-4-c	1333	5326	21.3	<u>21.4</u>	-0.47
par32-4	3176	10313	33.65	<u>29.4</u>	12.63
par32-5-c	1339	5350	23.15	<u>22.05</u>	4.75
par32-5	3176	10325	32.9	<u>30.3</u>	7.90
Average					29.82

fore, the new algorithm focuses more on where the current state is within the search space and tries to fix possible problems in the structures of the state directly.

BGWalksat can significantly outperforms Walksat on SAT and Max-SAT, with both static and dynamic noise. On random Max-3-SAT problems, the more constrained the problems, the more improvement BGWalksat can provide. Specifically, on random Max-3-SAT with 2,000 variables and a C/V ratio of 8.0, BGWalksat can satisfy approximately 10% more clauses on average than Walksat (using dynamic noise). On satisfiable problems from the SATLIB, BGWalksat almost always significantly improves on Walksat's results, with the exception of graph coloring problems which contains a type of symmetry (see Section 5). On unsatisfiable SATLIB problems, BGWalksat substantially reduces the number of unsatisfied clauses across several instance classes.

In the process of developing BGWalksat, we also carried out a systematic experimental analysis of Walksat on Max-SAT (again with both static and dynamic noise). Our results show that although designed originally for SAT, Walksat is also effective on Max-SAT, even though its effectiveness degrades as the problem becomes more constrained.

References

- [1] K. D. Boese. *Models for Iterative Global Optimization*. PhD thesis, UCLA/Computer Science Department, 1996.

Table 3: Walksat vs. BGWalksat on unsatisfiable problems. The legend is the same as that in Table 2.

problem	#Var	#Clause	Walksat	BGWalksat	gain (%)
longmult06	2848	8853	<u>1.5</u>	1.65	-10.00
longmult07	3319	10335	<u>2.05</u>	2.2	-7.32
longmult08	3810	11877	<u>3.65</u>	<u>2.65</u>	27.40
longmult09	4321	13479	6.75	<u>2.9</u>	57.04
longmult10	4852	15141	10.25	<u>5.6</u>	45.37
longmult11	5403	16863	15.05	<u>9.2</u>	38.87
longmult12	5974	18645	17.8	<u>16.2</u>	8.99
longmult13	6565	20487	23.25	<u>21.4</u>	7.96
longmult14	7176	22389	32.6	<u>24.6</u>	24.54
longmult15	7807	24351	41.5	<u>30</u>	27.71
ssa6288-047	10410	34238	100.25	<u>89.7</u>	10.52
Average					20.01

- [2] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proc. IJCAI-91*, pages 331–337.
- [3] P. Codognet and F. Rossi. Notes for the ECAI2000 tutorial on Solving and Programming with Soft Constraints: Theory and Practice. available at <http://www.math.unipd.it/~frossi/papers.html>.
- [4] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *CACM*, 5:394–397, 1962.
- [5] O. Dubois and G. Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formula. In *Proc. IJCAI-01*, pages 248–253, 2001.
- [6] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
- [7] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, 1979.
- [8] H. H. Hoos. An adaptive noise mechanism for walksat. In *Proc. AAAI-02*.
- [9] H. H. Hoos and T. Stuzle. SATLIB - the satisfiability library. <http://www.informatik.tu-darmstadt.de/AI/SATLIB>, 1999.
- [10] Y. Jiang, H. Kautz, and B. Selman. Solving problems with hard and soft constraints using a stochastic algorithm for MAX-SAT. In *Proc. 1st Workshop on AI and OR*, Timberline, OR, 1995.
- [11] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Proc. AAAI-97*, pages 321–326, 1997.
- [12] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proc. AAAI-92*, 1992.
- [13] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400:133–137, 1999.
- [14] D. J. Patterson and H. Kautz. Auto-Walksat: A self-tuning implementation of Walksat. *Electronic Notes in Discrete Mathematics*, 9, 2001.
- [15] J. C. Pemberton and W. Zhang. Epsilon-transformation: Exploiting phase transitions to solve combinatorial optimization problems. *Artificial Intelligence*, 81:297–325, 1996.
- [16] B. Selman, H. Kautz, and B. Cohen. Noise strategies for local search. In *Proc. AAAI-94*.
- [17] J. Slaney and T. Walsh. Backbones in optimization and approximation. In *Proc. IJCAI-01*, 2001.
- [18] O. Telelis and P. Stamatopoulos. Heuristic backbone sampling for maximum satisfiability. In *Proc. 2nd Hellenic Conf. on AI*, pages 129–139, 2002.
- [19] W. Zhang. Phase transitions and backbones of 3-SAT and Maximum 3-SAT. In *Proc. 7th Intern. Conf. on Principles and Practice of Constraint Programming (CP-2001)*, pages 153–167, 2001.

SATISFIABILITY

SATISFIABILITY AND PHASE TRANSITIONS

Phase Transitions of Bounded Satisfiability Problems*

Delbert D. Bailey, Phokion G. Kolaitis

Computer Science Department

University of California, Santa Cruz

Santa Cruz, CA 95064, U.S.A

{dbailey, kolaitis}@cse.ucsc.edu

Abstract

We investigate phase transitions for the family of bounded satisfiability problems $3SAT(b)$, recently introduced by Zhang, that ask: given a 3CNF-formula, is there a truth assignment that violates no more than b of its clauses. Zhang's results were experimental and for a fixed number of variables ($n = 25$), and suggested that the locations of the phase transitions for $3SAT(b)$ are separated and move significantly as b increases. Analysis of these locations was posed as an open question.

We analytically show that the phase transitions of all $3SAT(b)$ problems must occur within a narrow region, regardless of how large the value of b is. Moreover, our experiments reveal that the phase transitions for these problems occur in a remarkable way. Specifically, unlike $3SAT$, the probability curves for $3SAT(b)$ do not have a quasi-common intersection point about which they rotate as they become steeper with increasing n . Instead, they move rapidly to the left toward the narrow region that the analysis predicts.

1 Introduction and Summary of Results

A phase transition of a “system” can be described as an abrupt change in the behavior of the system that occurs when a certain “control parameter” is at or near a certain critical value. Traditionally, phase transitions have been studied by physicists working in the area of statistical mechanics. During the past decade, however, computer scientists have carried out an intensive study of phase transitions of algorithmic problems, first of NP-complete decision problems and, more recently, of decision problems that are complete for higher computational complexity classes. These investigations have shed new light on the “structure” of presumably intractable decision problems by examining them from a perspective that had been hitherto unexplored in computer science; moreover, for certain fundamental algorithmic problems, the location of the phase transition has been correlated to the peak average cost of natural algorithms for solving these problems.

*Research of the authors was partially supported by NSF Grant No. IIS-9907419

The Boolean satisfiability problems $kSAT$, $k \geq 3$, constitute the most thoroughly investigated collection of NP-complete problems from the perspective of phase transitions. An instance of $kSAT$ is a $kCNF$ formula; the “control parameter” of such a formula is the ratio of the number of clauses over the number of variables occurring in the formula. Each fixed value r of this ratio gives rise to the family of probability spaces $F_k(n, r)$, $n \geq 1$, where $F_k(n, r)$ is the collection of all $kCNF$ -formulas with rn clauses generated by selecting k variables from n variables without replacement and then negating each variable with probability $1/2$. Franco and Paul [Franco and Paul, 1983] were the first to focus on this fixed clauses-to-variables ratio model and to initiate a study of the asymptotic behavior of the probability $p_k(n, r)$ that a random formula in $F_k(n, r)$ is satisfiable. During the past decade, much of the work in this area has been motivated from the conjecture of Chvátal and Reed [Chvátal and Reed, 1992] to the effect that, for every $k \geq 3$, there is a positive real number r_k such that if $r < r_k$, then $\lim_{n \rightarrow \infty} p_k(n, r) = 0$, whereas if $r > r_k$, then $\lim_{n \rightarrow \infty} p_k(n, r) = 1$.

The above conjecture asserts that, for every $k \geq 3$, a phase transition occurs in the probability $p_k(n, r)$ of a random formula in $F_k(n, r)$ being satisfiable, as $n \rightarrow \infty$. In spite of intensive efforts by several researchers, this conjecture has not been settled thus far. Nonetheless, progress toward establishing this conjecture has been made in two different fronts. On the experimental front, large-scale experiments with random Boolean formulas have provided evidence for the existence of a critical ratio r_k and have yielded estimates of its actual value. In particular, experiments by Selman, Mitchell and Levesque [Selman *et al.*, 1996] with random 3CNF-formulas and analysis of these experiments by Kirkpatrick and Selman [Kirkpatrick and Selman, 1994] indicate that r_3 is about 4.2. On the analytical front, there has been continuous progress toward establishing progressively tighter upper and lower bounds for the value of r_3 . The best analytical results obtained to date assert that if r_3 exists, then $3.42 < r_3 < 4.507$ [Kaporis *et al.*, 2002; Dubois *et al.*, 2000]. The experiments carried out by Selman, Mitchell and Levesque [Selman *et al.*, 1996] for random 3CNF-formulas also revealed that the critical ratio 4.2 appears to be the location at which the average cost of the Davis-Putnam-Logemann-Loveland (DPLL) procedure for Boolean satisfiability peaks.

Many fundamental algorithmic problems are optimization problems and not mere decision problems. In particular, if a Boolean formula happens to be over-constrained and, thus, unsatisfiable, it is natural to ask for the maximum number of clauses in the formula that can be simultaneously satisfied. By focusing on 3CNF-formulas, we obtain the optimization problem MAX 3SAT: given a 3CNF-formula φ , find the maximum number of clauses of φ that can be simultaneously satisfied. MAX 3SAT is a prototypical constraint optimization problem that is known to play a prominent role in the study of the approximability properties of NP-optimization problems. Indeed, as shown by Papadimitriou and Yannakakis [Papadimitriou and Yannakakis, 1991], MAX SAT is a MAX SNP-complete problem; this means that MAX SAT is a constant-approximable optimization problem and that every NP-optimization problem in the class MAX SNP can be reduced to it via an approximation-preserving polynomial-time reduction (see also [Papadimitriou, 1994]).

Its importance in complexity and approximability notwithstanding, MAX 3SAT had not been investigated from the perspective of phase transitions until recently. Zhang [Zhang, 2001] was the first to investigate phase transitions for MAX 3SAT by carrying out an initial set of experiments for the family of *bounded satisfiability* problems 3SAT(b), where b is a non-negative integer. An instance of 3SAT(b) is a 3CNF-formula φ and the question is: does there exist a truth assignment that violates no more than b clauses of φ ? Equivalently, 3SAT(b) asks whether the optimum value of MAX SAT on an instance φ is at least $m - B$, where m is the number of clauses of φ . Thus, each 3SAT(b) is a decision problem obtained from MAX SAT by imposing a “quality bound” on the optimum value. The control parameter of each 3SAT(b) is the ratio of clauses to variables, that is, the same control parameter as the one used for 3SAT.

Zhang [Zhang, 2001] explored phase transitions for the bounded satisfiability problems 3SAT(b) by running experiments on random 3CNF-formulas with $n = 25$ variables and for $b = 5, 10, 15, 20$. His findings suggest that there is a series of separated phase transitions corresponding to different quality bounds. The location of each phase transition appears to increase with b (see Figure 7); moreover, the average cost for solving the optimization problem MAX SAT appears to be the envelope of the peak average cost for solving the decision problems 3SAT(b), as b increases. Zhang [Zhang, 2001] did not report on the asymptotic behavior of 3SAT(b) as the number of variables increases, because he ran experiments for just a single value ($n = 25$) of the number of variables. He posed the analysis of phase transitions for 3SAT(b) as an open problem and, in particular, raised the question of finding the exact location of the phase transitions for these decision problems.

In this paper, we report on a systematic investigation of phase transitions for the family of bounded satisfiability problems 3SAT(b). Our investigation has produced both analytical and experimental results that yield a more complete and, at the same time, rather surprising picture of these phase transitions. As stated above, Zhang’s [Zhang, 2001] initial experiments suggest that the location of the phase transition for each 3SAT(b) increases with b . Here, using the first-

moment method, we show analytically that the phase transitions for 3SAT(b) must occur (if they occur at all) within a rather narrow region, regardless of the value of b . Moreover, the same behavior is exhibited by the families k SAT(b) of decision problems underlying the optimization problems MAX k SAT, $k \geq 3$. In particular, the locations of the phase transitions for 3SAT(b) are bounded from above by $1/(3 - \lg(7)) \approx 5.19$ and from below by the greatest known lower bound 3.42 for the location of the phase transition for 3SAT. At the experimental front, we investigated the asymptotic behavior of 3SAT(b) for the values $b = 3, 4$ and 5 by running experiments for several different values of the number n of variables of random 3CNF-formulas ($n = 10, 15, 20, 25, 30, 35$). Our most striking experimental finding is the discovery that, as n increases, the phase transition for each 3SAT(b) emerges in a manner that is qualitatively different from that for 3SAT, as we describe next.

In the case of 3SAT, the experiments in [Selman *et al.*, 1996] revealed that the family of curves for the probability of satisfiability of a random 3CNF-formula (one curve for each number n of variables) has the property that the curves become progressively steeper and every two of them intersect at a single point whose abscissa is near the location of the phase transition for 3SAT (see Figure 1). In contrast, our experiments reveal that, for each bounded satisfiability problem 3SAT(b), the family of curves for the probability of a “yes” answer of a random 3CNF-formula (one curve for each number n of variables) become progressively steeper but they do *not* intersect; instead, they are separated by a distance that is getting smaller as n increases. Figures 2, 3, and 4 depict these findings for 3SAT(3), 3SAT(4) and 3SAT(5), respectively. This is a qualitatively different pattern of emergence of a phase transition that does not seem to have been encountered in earlier investigations of phase transitions of other NP-complete problems in which the probability curves appear to have a quasi-common intersection point.

2 Bounded Satisfiability Problems

For every positive integer $k \geq 3$, MAX k SAT is the following optimization problem: given a k CNF-formula φ , find the maximum number of simultaneously satisfied clauses of φ . The study of phase transitions for an optimization problem begins by focusing on one or more decision problems that underlie that optimization problem. The most natural way to derive a decision problem from an optimization problem is to consider as input both an instance of the optimization problem and an arbitrary quality bound, and to ask whether the optimum value on this instance is bigger (or smaller) than the given quality bound. In the case of MAX k SAT, this gives rise to the following NP-complete decision problem: given a k CNF-formula φ and an integer c , is there a truth assignment to the variables of φ that satisfies at least c clauses of φ ? It is also possible to derive a family of decision problems that underlie MAX k SAT and are such that the inputs to each of them are k CNF-formulas only. For each non-negative integer b , let k SAT(b) be the following *bounded satisfiability* problem: given a k CNF-formula φ , is there a truth assignment to the variables of φ that violates no more than b clauses of

φ ? Clearly, $k\text{SAT}(0)$ is $k\text{SAT}$ itself; moreover, it is easy to show that every bounded satisfiability problem $k\text{SAT}(b)$, where $k \geq 3$ and $b \geq 1$, is NP-complete. Since the input to $k\text{SAT}(b)$ is a $k\text{CNF}$ -formula, phase transitions for the family $k\text{SAT}(b)$ can be studied using the ratio of clauses to variables as the control parameter, that is, the same control parameter as the one used in the study of phase transitions for $k\text{SAT}$.

There have been earlier investigations of phase transitions for other fundamental optimization problems, including the TRAVELING SALESMAN PROBLEM [Gent and Walsh, 1996b] and NUMBER PARTITIONING [Gent and Walsh, 1996a]. It should be noted, however, that for each of these two optimization problems a single decision problem was considered (instead of a family of decision problems) because the quality bound was part of the input to the decision problem. Moreover, the control parameter for these problems was chosen in such a way that it encoded the quality bound in some direct or indirect way, and this choice was criticized in subsequent investigations [Slaney and Thiébaux, 1998]. Clearly, this criticism does not apply to the bounded satisfiability problems $k\text{SAT}(b)$.

For every integers $k \geq 3$, $n \geq 1$, and $b \geq 0$, let $\Pr_k[n, r, b]$ be the probability that a random formula φ in the space $F_k(n, r)$ is a “yes” instance of $k\text{SAT}(b)$, that is, there is a truth assignment that violates no more than b clauses of φ . We now have all the notation in place to formulate the following conjecture concerning phase transitions for the bounded satisfiability problems $k\text{SAT}(b)$.

Conjecture 2.1: For every integer $k \geq 3$ and every integer $b \geq 0$, there is a positive real number $r_{k,b}$ such that

- If $r < r_{k,b}$, then $\lim_{n \rightarrow \infty} \Pr_k[n, r, b] = 1$;
- If $r > r_{k,b}$, then $\lim_{n \rightarrow \infty} \Pr_k[n, r, b] = 0$.

Note that Chvátal and Reed’s conjecture [Chvátal and Reed, 1992] concerning phase transitions for $k\text{SAT}$, $k \geq 3$, is the special case of Conjecture 2.1 in which $b = 0$. As mentioned in Section 1, Zhang [Zhang, 2001] raised the problem of finding the exact location of the phase transition for $3\text{SAT}(b)$, which, in terms of the notation introduced here, amounts to first establishing Conjecture 2.1 for $k = 3$ and then determining the critical ratio $r_{3,b}$, for each b . Although far from solving these problems, the next result yields analytical upper bounds for the values of $r_{k,b}$; in particular, it demonstrates that for each $k \geq 3$, all critical ratios $r_{k,b}$ are bounded by a quantity that depends only on k .

Proposition 2.2: Fix two integers $k \geq 3$ and $b \geq 0$.

- If $r > \frac{1}{k - \lg(2^k - 1)}$, then $\lim_{n \rightarrow \infty} \Pr_k[n, r, b] = 0$.
- Consequently, if $r_{k,b}$ exists, then $r_{k,b} \leq \frac{1}{k - \lg(2^k - 1)}$, regardless of the value of b .

Proof: If X is a random variable taking non-negative values and having finite expectation $E(X)$, then Markov’s inequality asserts that $\Pr[X \geq 1] \leq E(X)$. This inequality has been used in the past to obtain a coarse upper bound for the critical ratio r_k for the phase transition of $k\text{SAT}$. Here, we will apply the same method to obtain upper bounds for the critical ratios $r_{k,b}$ for the phase transitions of $k\text{SAT}(b)$. In this

case, however, bounding the expectation of a suitable random variable X is not as trivial as it was for $k\text{SAT}$, but still quite straightforward.

Let $X_k^{n,r,b}$ be a random variable on $F_k(n, r)$ such that if φ is a formula in $F_k(n, r)$, then $X_k^{n,r,b}(\varphi)$ is the number of truth assignments on n variables that violate no more than b clauses of φ . Clearly, $\Pr_k[n, r, b] = \Pr[X_k^{n,r,b} \geq 1]$, thus our goal now is to bound the expectation $E(X_k^{n,r,b})$.

For every truth assignment α on n variables, let I_α^b be the Bernoulli random variable on $F_k(n, r)$ such that $I_\alpha^b(\varphi) = 1$, if α violates no more than b clauses of φ , and $I_\alpha^b(\varphi) = 0$, otherwise. It is easy to see that the number of formulas in $F_k(n, r)$ for which α violates no more than b clauses is $\sum_{i=0}^b \binom{rn}{i} \binom{n}{k}^i (2^k - 1)^{rn-i} \binom{n}{k}^{rn-i}$, which implies that

$$E(I_\alpha^b) = \frac{\sum_{i=0}^b \binom{rn}{i} \binom{n}{k}^i (2^k - 1)^{rn-i} \binom{n}{k}^{rn-i}}{2^{krn} \binom{n}{k}^{rn}}.$$

Since $X_k^{n,r,b} = \sum_\alpha I_\alpha^b$, the linearity of expectation implies that $E(X_k^{n,r,b}) = \sum_\alpha E(I_\alpha^b) = 2^n E(I_\alpha^b)$. Thus,

$$\begin{aligned} E(X_k^{n,r,b}) &= 2^n \left(\frac{\sum_{i=0}^b \binom{rn}{i} \binom{n}{k}^i (2^k - 1)^{rn-i} \binom{n}{k}^{rn-i}}{2^{krn} \binom{n}{k}^{rn}} \right) \\ &= \left(2 \left(\frac{2^k - 1}{2^k} \right)^r \right)^n \sum_{i=0}^b \frac{\binom{rn}{i}}{(2^k - 1)^i} \end{aligned}$$

Consequently,

$$\lim_{n \rightarrow \infty} E(X_k^{n,r,b}) \leq \lim_{n \rightarrow \infty} \frac{\mathcal{O}(n^b)}{\left(\frac{1}{2} \left(\frac{2^k}{2^k - 1} \right)^r \right)^n}.$$

Thus, if $r > \frac{1}{k - \lg(2^k - 1)}$, then $\lim_{n \rightarrow \infty} E(X_k^{n,r,b}) = 0$, regardless of the value of b . Now an application of Markov’s inequality to $X_k^{n,r,b}$ gives the result. ■

Concerning lower bounds for $r_{k,b}$, it is obvious that if r is a ratio such that almost all $k\text{CNF}$ -formulas are satisfiable, then $\lim_{n \rightarrow \infty} \Pr_k[n, r, b] = 1$, regardless of the value of b . Consequently, if r_k is the critical ratio for $k\text{SAT}$, then $r_k \leq r_{k,b}$, regardless of the value of b . Of course, this assumes that the critical ratios r_k and $r_{k,b}$ actually exist. Until this is established, we can use the analytically derived lower bounds for r_k as lower bounds for every $r_{k,b}$. In particular, the best currently known such lower bound for r_3 is 3.42 [Kaporis *et al.*, 2002]. By combining these remarks with Proposition 2.2, we obtain the following bounds for the phase transitions of the bounded satisfiability problems $3\text{SAT}(b)$.

Corollary 2.3: Let $b \geq 0$ be an integer.

- If $r < 3.42$, then $\lim_{n \rightarrow \infty} \Pr_3[n, r, b] = 1$
- If $r > \frac{1}{3 - \lg(7)} \approx 5.19$, then $\lim_{n \rightarrow \infty} \Pr_3[n, r, b] = 0$.

Consequently, if $r_{3,b}$ exists, then $3.42 \leq r_{3,b} \leq 5.19$, regardless of the value of b .

3 Experimental Results

We ran experiments for random 3CNF-formulas drawn from $n = 10, 15, 20, 25, 30$ and 35 variable spaces. We implemented a suitably modified version of the Davis-Logemann-Loveland-Putnam (DPLL) procedure and tested each random formula drawn to determine whether it is a “yes” instance of the bounded satisfiability problems $3SAT(b)$, for $b = 3, 4$ and 5 . For each (n, b) -pair, probability curves were determined by recording the average number of “yes” instances in samples of 1200 formulas, one sample for each possible ratio of clauses to variables up to a ratio of 15.

The probability curves for $3SAT(b)$ with $b = 3, 4$ and 5 are depicted in Figures 2, 3 and 4, respectively.

In these figures, we see that, as r increases, the probability curves start out in a region where the probability is 1 or close to 1 and then they transition to a region where the probability is 0 or close to 0. To make the terminology more precise, let us define the *location* of the transition for the probability of $3SAT(b)$ to be the ratio r at which the probability is 0.5, and the *width* of the transition for the probability of $3SAT(b)$ to be the difference in the r values between the ratios at which the probability falls from 0.9 to 0.1. As n increases, the probability curves move from right to left; moreover, both the location and the width of the transition appear to change with different values of n . For each fixed b , the width becomes smaller with increasing n just as is the case for the probability curves for 3SAT in Figure 1; the location, however, moves dramatically to the left in sharp contrast with the behavior for 3SAT, where the location moves very slightly and converges to a value in the transition region where all the curves appear to intersect at one point. The magnitude of the leftward movement of the transition location with increasing n appears to become greater with increasing values of b . Thus, the phase transitions for the bounded satisfiability problems $3SAT(b)$ emerge in a pattern that is novel and qualitatively different from that of the phase transitions for other NP-complete problems.

We also recorded the average cost of the modified DPLL-procedure for solving $3SAT(b)$, where $b = 3, 4, 5$. Each of these problems is much harder on average than 3SAT; moreover, these problems are getting harder on average as b increases. As a matter of fact, the experiments for $b = 5$ and for $n = 35$ took several weeks to complete. Figure 5 depicts the average cost of solving 3SAT, while Figures 6 depicts the average cost of solving $3SAT(5)$.

In Figure 6, the performance curves for $3SAT(5)$ move from lower to higher with increasing n and, as has been observed with 3SAT, the peaks in cost appear to correspond to the location of the transitions. The peaks move from right to left in synchrony with the movement of the location of the transition for $3SAT(b)$; this movement is more dramatic than in the case of 3SAT.

4 Finite Size Scaling

The locations of the transition in the probability curves for $3SAT(b)$ appear to approach a limiting critical ratio $r_{3,b}$ that is within the analytically derived upper and lower bounds in Corollary 2.3. Nonetheless, it is not clear what the critical ratio $r_{3,b}$ actually is, nor can it be estimated by visual inspection.

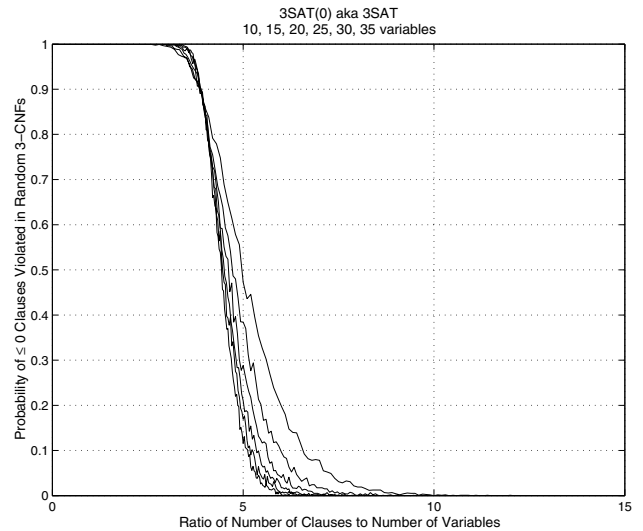


Figure 1: The probability curves for bounded satisfiability with $b = 0$. There is a curve for each of six different values of n . They show the well known 3SAT behavior. As n increases the transition regions of the curves become steeper and appear to have a common intersection point about which they rotate. The common hypothesis is that in the limit the curves approach a step function through this point and its ratio is the critical phase transition value for 3SAT, approximately 4.2.

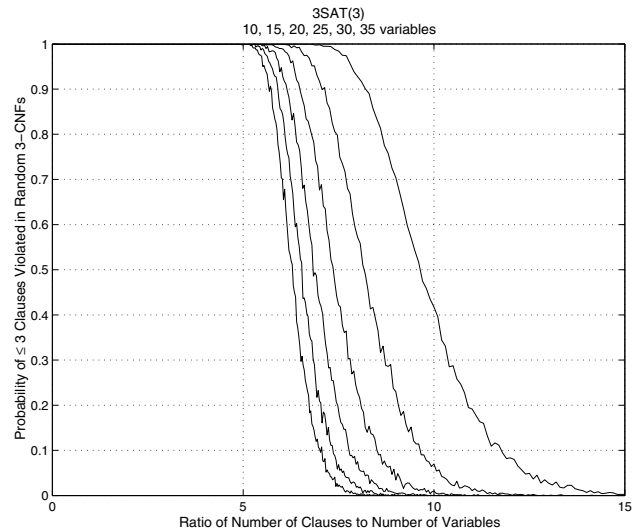


Figure 2: These are the probability curves for $3SAT(3)$. There is a curve for each of the same n values as in Figure 1. As n increases, the transition regions appear to steepen and distinctly move to the left. The steepening is similar to 3SAT behavior but here there is no apparent common rotation point. The transitions appear to be approaching a step function in the limit but it is not possible to visually determine the critical ratio. We used a form of finite size scaling to estimate it as approximately 4.4

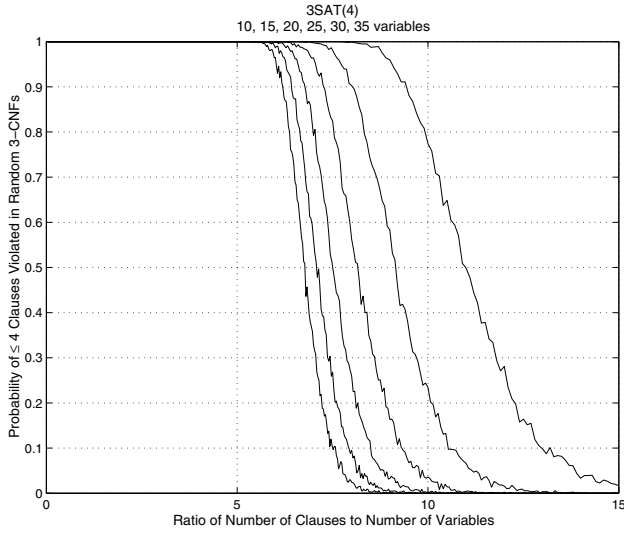


Figure 3: These are the probability curves for 3SAT(4). There is a curve for each of the same n values as in the previous figures. Again, as n increases, the transition regions appear to steepen and distinctly move to the left. The steepening is similar to 3SAT behavior but here there is no apparent common rotation point. The transitions appear to be approaching a step function in the limit but it is not possible to visually determine the critical ratio. We used a form of finite size scaling to estimate it as approximately 4.5

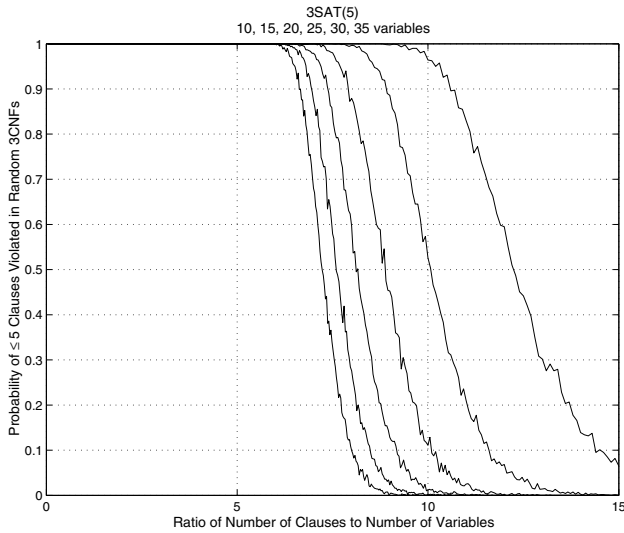


Figure 4: These are the probability curves for 3SAT(5). There is a curve for each of the same n values as in the previous figures. Again, as n increases, the transition regions appear to steepen and distinctly move to the left. The steepening is similar to 3SAT behavior but here there is no apparent common rotation point. The transitions appear to be approaching a step function in the limit but it is not possible to visually determine the critical ratio. We used a form of finite size scaling to estimate it as approximately 4.8. Note the analytical upper bound is approximately 5.19.

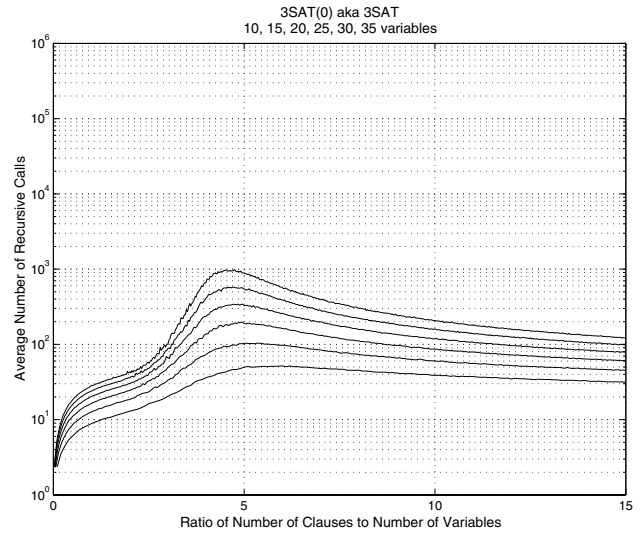


Figure 5: This set of curves plots the well known performance behavior of the DPLL procedure for 3SAT. This figure is here for comparison with the following figure for 3SAT(5).

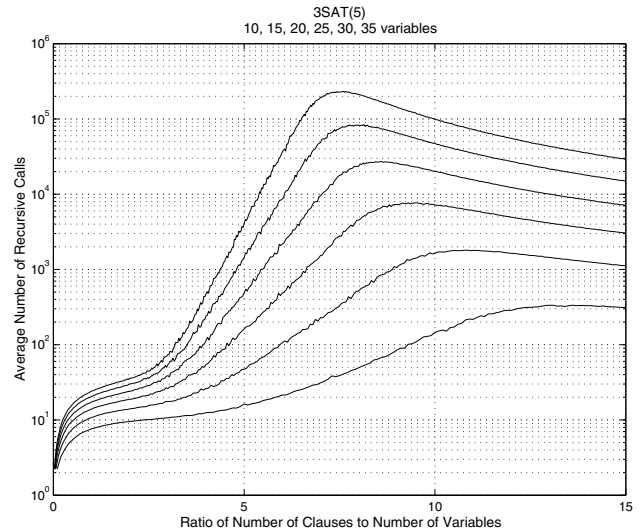


Figure 6: This set of curves plots the algorithmic costs for 3SAT(5). The behavior is similar to 3SAT in that cost increases with increasing n ; moreover, cost exhibits the so-called “easy/hard/easy” pattern as the ratio increases. As would be expected the costs for bounded satisfiability are much higher than for satisfiability. Note that the movement of the peak locations, as n increases, is quite marked and corresponds to the leftward movement of the transition regions in the probability curves.

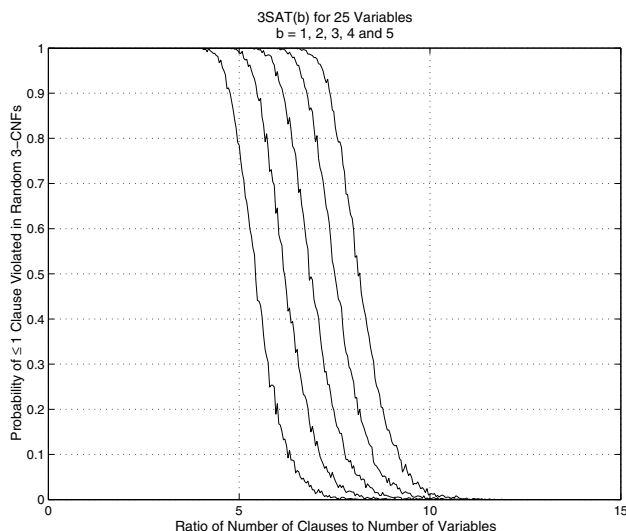


Figure 7: This figure shows the apparently separate “phase transitions” for different values of b when looking at probability curves for a fixed n , in this case 25 variables. The transition region moves from left to right with increasing b . This type of behavior was previously reported for $b = 5, 10, 15$, and 20 [Zhang, 2001].

tion since no two probability curves for $3SAT(b)$ intersect. A finite-size scaling analysis of the data, assuming a power law of the form $\frac{(r-r_{3,b})n^{\nu_{3,b}}}{r_{3,b}}$, allowed us to obtain estimates for both the critical ratio $r_{3,b}$ and for the exponent $\nu_{3,b}$, where $b = 3, 4$ and 5 .

In [Kirkpatrick and Selman, 1994], an estimate for the value of the critical ratio r_k was obtained by visually estimating the point at which the probability curves for $kSAT$ appear to intersect, and then regression techniques were applied to determine the best exponent ν_k in the power law for $kSAT$. This cannot be done with the bounded satisfiability problems $3SAT(b)$, since the probability curves do not intersect, but appear to be moving to the left as the number n of variables increases. Nevertheless the finite-size scaling hypothesis can still be tested with a more elaborate procedure. Our goal was to test whether it is possible find values for $r_{3,b}$ and $\nu_{3,b}$ such that when the ratio r is rescaled with the above power law, the probability curves for the different values n of the number of variables collapse to a single curve. Using routines for interpolation and regression from Matlab, we stepped through small increments of putative $r_{3,b}$ values and measured how well the curves collapsed to a single curve by calculating the sum of their pairwise squared differences when accordingly transformed. The estimated $r_{3,b}$ was taken to be the value that minimized this difference measure. We found this technique to be well behaved, in that it exhibited a clear minimum and gave much more precision than visually judging how well the curves appeared to collapse to a single curve. To further validate this technique, we checked it on the phase transition of $2SAT$, which has been analytically determined to occur at the critical ratio $r_2 = 1$ [Chvátal and Reed, 1992; Fernandez de la Vega, 1992; Goerdts, 1996]. The agreement was very good, namely, the estimate for r_2 was 0.98 .

Table 1: Finite-size Scaling Results

b	$r_{3,b}$	$\nu_{3,b}$
3	4.40+	0.64+
4	4.48+	0.68+
5	4.84	0.77

The results of the finite-size scaling analysis for $3SAT(b)$, where $b = 3, 4$ and 5 , are shown in Table 1. Note that these results are consistent with the analytical upper and lower bounds for $r_{3,b}$ in Corollary 2.3, that the $r_{b,k}$ are in the region between r_k for $k-SAT$ and the Markov upper bound for $k-SAT$. This consistency is the most that can be reasonably concluded from the finite-size scaling analysis. The technique provides a more principled way to make the estimates than eyeballing, but no great accuracy should be attributed to the estimates in spite of their apparent precision. Although the finite-size scaling results suggest that the locations of the phase transitions of $3SAT(b)$ are different for different b ’s, it may still be the case that they all coincide with the location for $3SAT$.

5 Concluding Remarks

The results reported here advance the understanding of the phase transitions for the family of bounded satisfiability problems $3SAT(b)$, introduced by Zhang [Zhang, 2001].

Our main analytical finding is that the phase transitions of all $3SAT(b)$ problems must occur within a narrow region, regardless of how large the value of b is. Moreover, our experiments revealed that the phase transitions for these problems occur in a remarkable way. Specifically, unlike $3SAT$, the probability curves for $3SAT(b)$ do not have a quasi-common intersection point about which they rotate as they become steeper with increasing n . Instead, they move rapidly to the left toward the narrow region that the analysis predicts.

Identifying the exact locations of the phase transitions for $3SAT(b)$ remains an open problem, which is at least as hard as identifying the location of the phase transition for $3SAT$. Equally hard appears to be the problem of analytically establishing whether these locations are separated or coincide. Nonetheless, it may be possible to analytically obtain tighter bounds for these locations using some of the sophisticated techniques that have been quite successful in finding tighter bounds for the phase transition of $3SAT$.

Acknowledgments: We are grateful to Víctor Dalmau for stimulating and insightful discussions. We also wish to thank the reviewers for their many useful comments and suggestions.

References

- [Chvátal and Reed, 1992] V. Chvátal and B. Reed. Mick gets some (the odds are on his side). In *33th Annual Symposium on Foundations of Computer Science*, pages 620–627, 1992.
- [Dubois *et al.*, 2000] O. Dubois, Y. Boufkhad, and J. Mandler. Typical random 3-SAT formulae and the satisfiability threshold. In

- Proc. 11th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 126–127, 2000.
- [Fernandez de la Vega, 1992] W. Fernandez de la Vega. On random 2-SAT. Manuscript, 1992.
- [Franco and Paul, 1983] J. V. Franco and M. Paul. Probabilistic analysis of the Davis-Putnam Procedure for solving the satisfiability problem”. *Discrete Applied Math.*, 5:77–87, 1983.
- [Gent and Walsh, 1996a] Ian Gent and Toby Walsh. Phase transitions and annealed theories: number partitioning. *Proc. of 12th European Conf. on Artificial Intelligence, ECAI-96*, pages 170–174, 1996.
- [Gent and Walsh, 1996b] Ian Gent and Toby Walsh. The TSP phase transition. *Artificial Intelligence*, 88(1–2):349–358, 1996.
- [Goerdt, 1996] A. Goerdt. A threshold for unsatisfiability. *J. Comput. System Sci.*, 53(3):469–486, 1996.
- [Kaporis *et al.*, 2002] A. C. Kaporis, L. M. Kirousis, and E. G. Lalas. The probabilistic analysis of a greedy satisfiability algorithm. In R. Möhring and R. Ramen, editors, *Algorithms-ESA2002, Proceedings of the Tenth Annual European Symposium, LNCS 2461*, pages 574–585. Springer-Verlag, 2002.
- [Kirkpatrick and Selman, 1994] S. Kirkpatrick and B. Selman. Critical behavior in the satisfiability of random formulas. *Science*, 264:1297–1301, 1994.
- [Papadimitriou and Yannakakis, 1991] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Science*, 43:425–440, 1991.
- [Papadimitriou, 1994] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
- [Selman *et al.*, 1996] B. Selman, D. G. Mitchell, and H. J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):17–29, 1996.
- [Slaney and Thiébaux, 1998] John Slaney and Sylvie Thiébaux. On the hardness of decision and optimisation problems. In Henri Prade, editor, *13th European Conference on Artificial Intelligence, ECAI 98*, pages 224–248. John Wiley & Sons, Ltd., 1998.
- [Zhang, 2001] Weixiong Zhang. Phase transitions and backbones of 3-SAT and maximum 3-SAT. In T. Walsh, editor, *Principles and Practice of Constraint Programming - CP 2001*, volume 2239 of *LNCS*, pages 153–167. Springer-Verlag, 2001.
-

Understanding the Power of Clause Learning

Paul Beame*

Henry Kautz*

Ashish Sabharwal*

Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350

{beame, kautz, ashish}@cs.washington.edu

Abstract

Efficient implementations of DPLL with the addition of clause learning are the fastest complete satisfiability solvers and can handle many significant real-world problems, such as verification, planning, and design. Despite its importance, little is known of the ultimate strengths and limitations of the technique. This paper presents the first precise characterization of clause learning as a proof system, and begins the task of understanding its power. In particular, we show that clause learning using any non-redundant scheme and unlimited restarts is equivalent to general resolution. We also show that without restarts but with a new learning scheme, clause learning can provide exponentially smaller proofs than regular resolution, which itself is known to be much stronger than ordinary DPLL.

1 Introduction

In recent years the task of deciding whether a CNF propositional logic formula is satisfiable has gone from a problem of theoretical interest to a practical approach for solving real-world problems. SAT procedures are now a standard tool for hardware verification, including verification of super-scalar processors [Velev and Bryant, 2001; Biere *et al.*, 1999]. Open problems in group theory have been encoded and solved using satisfiability [Zhang and Hsiang, 1994]. Other applications of SAT include circuit diagnosis and experiment design [Konuk and Larrabee, 1993; Gomes *et al.*, 1998b].

The most surprising aspect of such relatively recent practical progress is that the best complete satisfiability testing algorithms remain variants of the DPLL procedure [Davis and Putnam, 1960; Davis *et al.*, 1962] for backtrack search in the space of partial truth assignments. The main improvements to DPLL have been better branch selection heuristics (*e.g.*, [Li and Anbulagan, 1997]), and extensions such as randomized restarts [Gomes *et al.*, 1998a] and clause learning. One can argue that clause learning has been the most significant of these in scaling DPLL to realistic problems.

Clause learning grew out of work in AI on explanation-based learning (EBL), which sought to improve the perfor-

mance of backtrack search algorithms by generating explanations for failure (backtrack) points, and then adding the explanations as new constraints on the original problem [de Kleer and Williams, 1987; Stallman and Sussman, 1977; Genesereth, 1984; Davis, 1984]. For general constraint satisfaction problems the explanations are called “conflicts” or “no goods”; in the case of Boolean CNF satisfiability, the technique becomes clause learning. A series of researchers [Bayardo Jr. and Schrag, 1997; Marques-Silva and Sakallah, 1996; Zhang, 1997; Moskewicz *et al.*, 2001; Zhang *et al.*, 2001] showed that clause learning can be efficiently implemented and used to solve hard problems that cannot be approached by any other technique.

Despite its importance there has been little work on formal properties of clause learning, with the goal of understanding its fundamental strengths and limitations. A likely reason for such inattention is that clause learning is a rather complex rule of inference – in fact, as we describe below, a complex family of rules of inference. A contribution of this paper is that we provide a precise specification of clause learning.

Another problem in characterizing clause learning is defining a formal notion of the strength or power of a reasoning method. This paper uses the notion of proof complexity [Cook and Reckhow, 1977], which compares inference systems in terms of the sizes of the shortest proofs they sanction. A family of formulas C provides an *exponential separation* between systems S_1 and S_2 if the shortest proofs of formulas in C in system S_1 are exponentially smaller than the corresponding shortest proofs in S_2 . From this basic propositional proof complexity point of view, only families of unsatisfiable formulas are of interest, because only proofs of unsatisfiability can be large; minimum proofs of satisfiability are linear in the number of variables of the formula. Nevertheless, Achlioptas *et al.* [2001] have shown how negative proof complexity results for unsatisfiable formulas can be used to derive time lower bounds for specific inference algorithms running on satisfiable formulas as well.

Proof complexity does not capture everything we intuitively mean by the power of a reasoning system, because it says nothing about how difficult it is to find shortest proofs. However, it is a good notion with which to begin our analysis, because the size of proofs provides a lower-bound on the running time of any implementation of the system. In the systems we consider, a branching function, which determines

* Research supported by NSF Grant ITR-0219468

which variable to split upon or which pair of clauses to resolve, guides the search. A negative proof complexity result for a system tells us that a family of formulas is intractable even with a perfect branching function; likewise, a positive result gives us hope of finding a branching function.

A basic result in proof complexity is that general resolution is exponentially stronger than the DPLL procedure [Bonet *et al.*, 2000; Ben-Sasson *et al.*, 2000]. This is because the trace of DPLL running on an unsatisfiable formula can be converted to a tree-like resolution proof of the same size, and tree-like proofs must sometimes be exponentially larger than the DAG-like proofs generated by general resolution. Although resolution can yield shorter proofs, in practice DPLL is better because it provides a more efficient way to search for proofs. The weakness of the tree-resolution proofs that DPLL finds is that they do not reuse derived clauses. The conflict clauses found when DPLL is augmented by clause learning correspond to reuse of derived clauses in the associated resolution proofs and thus to more general forms of resolution proofs. An intuition behind the results in this paper is that the addition of clause learning moves DPLL closer to general resolution while retaining its practical efficiency.

It has been previously observed that clause learning can be viewed as adding resolvents to a tree-like proof [Marques-Silva, 1998]. However, this paper provides the first mathematical *proof* that clause learning is exponentially stronger than tree-like resolution. Further, we provide a family of formulas that exponentially separates clause learning from regular resolution, a system that is known to be intermediate in strength between general and tree resolution. The proof uses a new clause learning scheme called FirstNewCut that we introduce. We also show that combining clause learning with restarts is as strong as general resolution.

Although this paper focuses on basic proof complexity, we briefly indicate how the understanding we gain through this kind of analysis may lead to practical applications. For example, our proofs describe an improvement to the clause learning rules previously suggested in the literature, and suggest an approach to leveraging the structure of a problem encoded as a CNF formula in order to create a branching heuristic that takes the greatest advantage of clause learning. As an example, we apply these ideas to certain natural satisfiable and unsatisfiable formulas where we obtain significant speed-ups over existing methods.

2 Preliminaries

A CNF formula F is an AND ($\wedge$) of *clauses*, where each clause is an OR ($\vee$) of *literals*, and a literal is a variable or its negation ($\neg$). It is natural to think of F as a set of clauses and each clause as a set of literals. A clause that is a subset of another is called its *subclause*.

Let ρ be a partial assignment to the variables of F . The *restricted formula* $F|\rho$ is obtained from F by replacing variables in ρ with their assigned values. F is said to be *simplified* if all clauses with at least one TRUE literal are deleted, all occurrences of FALSE literals are removed from clauses, and the resulting formula, if different, is simplified recursively.

2.1 The DPLL Procedure

The basic idea of the Davis-Putnam-Logemann-Loveland (DPLL) procedure [Davis and Putnam, 1960; Davis *et al.*, 1962] for testing satisfiability of CNF formulas is to *branch* on variables, setting them to TRUE or FALSE, until either an initial clause is *violated* (i.e. has all literals set to FALSE) or all variables have been set. In the former case, we backtrack to the last branching decision whose other branch has not been tried yet, reverse the decision, and proceed recursively. In the latter, we terminate with a satisfying assignment. If all possible branches have been unsuccessfully tried, the formula is declared unsatisfiable. To increase efficiency, *pure literals* (those whose negation does not appear) and *unit clauses* (those with only one unset literal) are immediately set to true.

Definition 1. A *branching sequence* for a CNF formula F is a sequence $\sigma = (l_1, l_2, \dots, l_k)$ of literals of F , possibly with repetitions. DPLL on F *branches according to* σ if it always picks the next variable v to branch on in the literal order given by σ , skips it if v is currently assigned a value, and branches further by setting the chosen literal to FALSE otherwise.

In this paper, we will use the term DPLL to denote the basic branching and backtracking procedure described above. It will, for instance, not include extensions such as learning conflict clauses or restarting, but will allow intelligent branching heuristics. Note that this is in contrast with the occasional use of the term DPLL to encompass practically all branching and backtracking approaches, including those involving learning.

2.2 Resolution

Resolution is a simple proof system that can be used to prove unsatisfiability of CNF formulas. The *resolution rule* states that given clauses $(A \vee x)$ and $(B \vee \neg x)$, we can derive clause $(A \vee B)$ by *resolving on* x . A *resolution derivation* of C from a CNF formula F is a sequence $\pi = (C_1, C_2, \dots, C_s \equiv C)$ where each clause C_i is either a clause of F (an *initial clause*) or derived by applying the resolution rule to C_j and C_k , $j, k < i$ (a *derived clause*). The *size* of π is s , the number of clauses occurring in it. We will assume that each $C_j \neq C$ in π is used to derive at least one other clause C_i , $i > j$. Any derivation of the empty clause Λ from F , also called a *refutation* or *proof* of F , shows that F is unsatisfiable.

Despite its simplicity, resolution is not efficiently implementable due to the difficulty of finding good choices of clauses to resolve; natural choices typically yield huge storage requirements. Various restrictions on the structure of resolution proofs lead to less powerful but easier to implement variants such as tree-like, regular, linear and positive resolution. *Tree-like* resolution uses non-empty derived clauses exactly once in the proof and is equivalent to an optimal DPLL procedure. *Regular* resolution allows any variable to be resolved upon at most once along any “path” in the proof from an initial clause to Λ . Both these variants are sound and complete but differ in efficiency – regular resolution is known to be exponentially stronger than tree-like [Bonet *et al.*, 2000; Ben-Sasson *et al.*, 2000], and general resolution is exponentially stronger than regular [Alekhnovich *et al.*, 2002].

Definition 2. A resolution derivation $(C_1, C_2, \dots, C_k)$ is *trivial* iff all variables resolved upon are distinct and each

$C_i, i \geq 3$, is either an initial clause or is derived by resolving C_{i-1} with an initial clause.

A trivial derivation is tree-like as well as regular. Moreover, the condition that each derived clause C_i use C_{i-1} in its derivation makes it *linear*. As we will see, trivial derivations correspond to conflicts in clause learning algorithms.

3 Clause Learning

Clause learning proceeds by following the normal branching process of DPLL until there is a “conflict” after unit propagation. If this conflict occurs without any branches, the formula is declared unsatisfiable. Otherwise, the “conflict graph” is analyzed and the “cause” of the conflict is learned in the form of a “conflict clause.” We now backtrack and continue as in ordinary DPLL, treating the learned clause just like initial ones. A clause is said to be *known* at a stage if it is either an initial clause or has already been learned. The learning process is expected to save us from redoing the same computation when we later have an assignment that causes conflict due in part to the same reason.

If a given CNF formula F is unsatisfiable, clause learning terminates with a conflict without any branches. Since all clauses used in this conflict themselves follow directly or indirectly from F , this failure of clause learning in finding a satisfying assignment constitutes a logical proof of unsatisfiability of F . Our bounds compare the size of such a proof with the size of a (possibly restricted) resolution proof of unsatisfiability of F .

Variations of such conflict driven learning [Bayardo Jr. and Schrag, 1997; Marques-Silva and Sakallah, 1996; Zhang, 1997; Moskewicz *et al.*, 2001; Zhang *et al.*, 2001] include different ways of choosing the clause to learn and possibly allowing multiple clauses to be learned from a single conflict. Although many such algorithms have been proposed and demonstrated to be empirically successful, a theoretical discussion of the underlying concepts and structures needed for our bounds is lacking. The rest of this section focuses on this formal framework.

Definition 3. A *clause learning proof* π of an unsatisfiable CNF formula F under scheme $\mathcal{S}$ and induced by branching sequence σ is the result of applying DPLL with unit propagation on F , branching according to σ , and using scheme $\mathcal{S}$ to learn conflict clauses such that at the end of this process, there is a conflict without any branches. The *size* of the proof, $size(\pi)$, is $|\sigma|$.

All clause learning algorithms discussed in this paper are based on *unit propagation*, which is the process of repeatedly applying the unit clause rule followed by formula simplification until no clause with exactly one unassigned literal remains. In this context, it is convenient to work with residual formulas at different stages of DPLL. Let ρ be the partial assignment at some stage of DPLL on formula F . The *residual formula* at this stage is obtained by simplifying $F|\rho$ and applying unit propagation.

When using unit propagation, variables assigned values through the actual branching process are called *decision* variables and those assigned values as a result of unit propagation

are called *implied* variables. *Decision and implied literals* are analogously defined. Upon backtracking, the last decision variable no longer remains a decision variable and might instead become an implied variable depending on the clauses learned so far. The *decision level* of a decision variable x is one more than the number of current decision variables at the time of branching on x . The *decision level* of an implied variable is the maximum of the decision levels of decision variables used to imply it. The *decision level* at any step of the underlying DPLL procedure is the maximum of the decision levels of all current decision variables.

3.1 Implication Graph and Conflicts

Unit propagation can be naturally associated with an *implication graph* that captures all possible ways of deriving all implied literals from decision literals.

Definition 4. The *implication graph* G at a given stage of DPLL is a directed acyclic graph with edges labeled with sets of clauses. It is constructed as follows:

1. Create a node for each decision literal, labeled with that literal. These will be the indegree zero root nodes of G .
2. While there exists a known clause $C = (l_1 \vee \dots \vee l_k \vee l)$ such that $\neg l_1, \dots, \neg l_k$ label nodes in G ,
 - (a) Add a node labeled l if not already present in G .
 - (b) Add edges $(l_i, l), 1 \leq i \leq k$, if not already present.
 - (c) Add C to the label set of these edges. These edges are thought of as grouped together and associated with clause C .
3. Add to G a special node Λ . For any variable x which occurs both positively and negatively in G , add directed edges from x and $\neg x$ to Λ .

Since all node labels in G are distinct, we identify nodes with the literals labeling them. Any variable x occurring both positively and negatively in G is a *conflict variable*, and x as well as $\neg x$ are *conflict literals*. G contains a *conflict* if it has at least one conflict variable. DPLL at a given stage has a *conflict* if the implication graph at that stage contains a conflict. A conflict can equivalently be thought of as occurring when the residual formula contains the empty clause Λ .

By definition, an implication graph may contain many conflict variables and several ways of deriving any single literal. To better understand and analyze a conflict, we work with a subgraph, called the *conflict graph* (see Figure 1), that captures only one among possibly many ways of reaching a conflict from the decision variables. The choice of the conflict graph is part of the strategy of the solver. It can also be thought of as giving power to clause learning by adding non-determinism.

Definition 5. A *conflict graph* H is any subgraph of an implication graph with the following properties:

1. H contains Λ and exactly one conflict variable.
2. All nodes in H have a path to Λ .
3. Every node l in H other than Λ either corresponds to a decision literal or has precisely the nodes $\neg l_1, \neg l_2, \dots, \neg l_k$ as predecessors where $(l_1 \vee l_2 \vee \dots \vee l_k \vee l)$ is a known clause.

Consider the implication graph at a stage where there is a conflict and fix a conflict graph contained in that implication graph. Pick any cut in the conflict graph that has all decision variables on one side, called the *reason side*, and Λ as well as at least one conflict literal on the other side, called the *conflict side*. All nodes on the reason side that have at least one edge going to the conflict side form a *cause* of the conflict. The negations of the corresponding literals forms the *conflict clause associated with this cut*.

Proposition 1. *Every conflict clause corresponds to a cut in a conflict graph that separates decision variables from Λ and a conflict literal.*

Proof. Let S denote the set containing the negations of the literals of a given conflict clause C and $pred(S)$ be the set of all predecessors of these literals in the underlying implication graph. Let T denote the set containing all literals obtained by unit propagation after setting literals in S to TRUE. Since C is a conflict clause, T must contain a conflict literal. Consider the subgraph $G_{S,T}$ of the implication graph induced by Λ and the literals in $S \cup pred(S) \cup T$, but having no edges going from $pred(S)$ to T . Fix any conflict graph that is a subgraph of $G_{S,T}$. The cut in this conflict graph with T as the conflict side has C as the conflict clause. $\square$

Proposition 2. *If there is a trivial resolution derivation of a clause C from a set of clauses F , then setting all literals of C to FALSE leads to a conflict.*

Proof. Let $\pi = (C_1, C_2, \dots, C_k \equiv C)$ be a trivial resolution derivation of C from F . Assume without loss of generality that clauses in π are ordered so that all initial clauses precede any derived clause. We give a proof by induction on the number of derived clauses in π .

For the base case, π does not have any derived clauses. Consequently, $C_k \in F$. Let $C_k = (l_1 \vee l_2 \vee \dots \vee l_q)$ and ρ be the partial assignment that sets all $l_i, 1 \leq i \leq q$, to FALSE. Unit propagation using clause C_k with $l_i, 1 \leq i \leq q - 1$, set to FALSE derives the literal l_q . Since $\neg l_q$ is a decision literal of ρ , l_q serves as a conflict variable and we have a conflict.

When π does have derived clauses, C_k , by triviality of π , must be derived by resolving C_{k-1} with a clause in F . Assume without loss of generality that $C_{k-1} \equiv (A \vee x)$ and the clause from F used in this resolution step is $(B \vee \neg x)$, where both A and B are subclauses of C_k . ρ falsifies all literals of B , implying $x = \text{FALSE}$ by unit propagation. This in turn results in falsifying all literals of C_{k-1} because all literals of A are also set to FALSE by ρ . Now $(C_1, \dots, C_{k-1})$ is a trivial resolution derivation of C_{k-1} from F with one less derived clause than π , and all literals of C_{k-1} are falsified. By induction, this must lead to a conflict. $\square$

Proposition 3. *Any conflict clause can be derived from known clauses using a trivial resolution derivation.*

Proof. In the light of Proposition 1, assume that we have a conflict clause associated with a cut σ in a fixed conflict graph. Let $V_{\text{conflict}}(\sigma)$ denote the set of variables on the conflict side of σ , but including the conflict variable only if it occurs both positively and negatively on the conflict side.

We will prove by induction on $|V_{\text{conflict}}(\sigma)|$ the stronger statement that the conflict clause associated with a cut σ has a trivial derivation resolving precisely on the variables in $V_{\text{conflict}}(\sigma)$.

For the base case, $V_{\text{conflict}}(\sigma) = \phi$ and the conflict side contains only Λ and a conflict literal, say x . The cause associated with this cut consists of node $\neg x$ that has an edge to Λ , and nodes $\neg l_1, \neg l_2, \dots, \neg l_k$ corresponding to a known clause $C_x = (l_1 \vee l_2 \vee \dots \vee l_k \vee x)$ that each have an edge to x . The conflict clause for this cut is simply the known clause C_x itself, having a length zero trivial derivation.

When $V_{\text{conflict}}(\sigma) \neq \phi$, pick a node y on the conflict side all whose predecessors are on the reason side. Let the conflict clause be $C = (l_1 \vee l_2 \vee \dots \vee l_p)$ and assume without loss of generality that the predecessors of y are $\neg l_1, \neg l_2, \dots, \neg l_k$ for some $k \leq p$. By definition of unit propagation, $C_y = (l_1 \vee l_2 \vee \dots \vee l_k \vee y)$ must be a known clause. Obtain a new cut σ' from σ by pulling node y to the reason side. The new associated conflict clause must be of the form $C' = (\neg y \vee D)$, where D is a subclause of C . Now $V_{\text{conflict}}(\sigma') \subset V_{\text{conflict}}(\sigma)$. Consequently, by induction, C' must have a trivial resolution derivation from known clauses resolving precisely upon the variables in $V_{\text{conflict}}(\sigma')$. Recall that no variable occurs twice in a conflict graph except the conflict variable. Hence $V_{\text{conflict}}(\sigma')$ has exactly all variables of $V_{\text{conflict}}(\sigma)$ except y . Using this trivial derivation of C' and finally resolving C' with the known clause C_y on variable y gives us a trivial derivation of C from known clauses. This completes the inductive step. $\square$

3.2 Different Learning Schemes

Different cuts separating decision variables from Λ and a conflict literal correspond to different learning schemes (see Figure 1). One can also create learning schemes based on cuts not involving conflict literals at all [Zhang *et al.*, 2001], but their effectiveness is not clear. These will not be considered here.

It is insightful to think of the *non-deterministic* scheme as the most general learning scheme. Here we pick the cut non-deterministically, choosing, whenever possible, one whose associated clause is not already known. Since we can repeatedly branch on the same last variable, non-deterministic learning subsumes learning multiple clauses from a single conflict as long as the sets of nodes on the reason side of the corresponding cuts form a (set-wise) decreasing sequence. For simplicity, we will assume that only one clause is learned from any conflict.

In practice, however, we employ deterministic schemes. The *decision* scheme [Zhang *et al.*, 2001], for example, uses the cut whose reason side comprises all decision variables. *rel-sat* [Bayardo Jr. and Schrag, 1997] uses the cut whose conflict side consists of all implied variables at the current decision level. This scheme allows the conflict clause to have exactly one variable from the current decision level, causing an automatic flip in its assignment upon backtracking.

This nice flipping property holds in general for all *unique implication points* (UIPs) [Marques-Silva and Sakallah, 1996]. A UIP of an implication graph is a node at the current decision level d such that any path from the decision variable at level d to the conflict variable as well as its negation

must go through it. Intuitively, it is a *single* reason at level d that causes the conflict. Whereas **rel-sat** uses the decision variable as the obvious UIP, **GRASP** [Marques-Silva and Sakallah, 1996] and **zChaff** [Moskewicz *et al.*, 2001] use *FirstUIP*, the one that is “closest” to the conflict variable. **GRASP** also learns multiple clauses when faced with a conflict. This makes it typically require fewer branching steps but possibly slower because of the time lost in learning and unit propagation.

The concept of UIP can be generalized to decision levels other than the current one. The *1UIP scheme* corresponds to learning the *FirstUIP* clause of the current decision level, the *2UIP scheme* to learning the *FirstUIP* clauses of both the current level and the one before, and so on. Zhang *et al* [2001] present a comparison of all these and other learning schemes and conclude that 1UIP is quite robust and outperforms all other schemes they consider on most of the benchmarks.

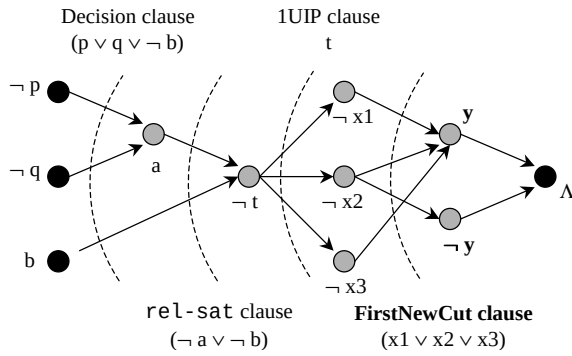


Figure 1: A conflict graph depicting various learning schemes

The FirstNewCut Scheme

We propose a new learning scheme called *FirstNewCut* whose ease of analysis helps us demonstrate the power of clause learning. We would like to point out that we use this scheme here only to prove our theoretical bounds. Its effectiveness on other formulas has not been studied yet.

The key idea behind *FirstNewCut* is to make the conflict clause as relevant to the current conflict as possible by choosing a cut close to the conflict literals. This is what the *FirstUIP* scheme also tries to achieve in a slightly different manner. For the following definitions, fix a cut in a conflict graph and let S be the set of nodes on the reason side that have an edge to some node on the conflict side (S is the reason side *frontier* of the cut). Let C_S be the conflict clause associated with this cut.

Definition 6. *Minimization* of conflict clause C_S is the process of repeatedly identifying, if one exists, a node $v \in S$, all of whose predecessors are also in S , moving it to the conflict side, and removing it from S .

Definition 7. *FirstNewCut scheme*: Start with a cut whose conflict side consists of Λ and a conflict literal. If necessary, repeat the following until the associated conflict clause, after minimization, is not already known: pick a node on the conflict side, pull all its predecessors except those that correspond to decision variables into the conflict side. Finally, learn the resulting new minimized conflict clause.

This scheme starts with the cut that is closest to the conflict literals and iteratively moves it back toward the decision variables until a new associated conflict clause is found. This backward search always halts because the cut with all decision variables on the reason side is certainly a new cut. Note that there are potentially several ways of choosing a literal to move the cut back, leading to different conflict clauses. The *FirstNewCut* scheme, by definition, always learns a clause not already known. This motivates the following:

Definition 8. A clause learning scheme is *non-redundant* if on a conflict, it always learns a clause not already known.

3.3 Fast Backtracking and Restarts

Most clause learning algorithms use *fast backtracking* where one uses the conflict graph to backtrack not only the last branching decision but also all other recent decisions that did not contribute to the conflict [Stallman and Sussman, 1977]. This adds power to clause learning because the current conflict might use clauses learned earlier as a result of branching on the apparently redundant variables. Hence, fast backtracking in general cannot be replaced by a “good” branching sequence that does not produce redundant branches. For the same reason, fast backtracking cannot either be replaced by simply learning the decision scheme clause. However, the results we present in this paper are independent of whether or not fast backtracking is used.

Restarts allow clause learning algorithms to arbitrarily restart their branching process. All clauses learned so far are however retained and now treated as additional initial clauses [Baptista and Silva, 2000]. As we will show, unlimited restarts can make clause learning very powerful at the cost of adding non-determinism. Unless otherwise stated, clause learning proofs will be assumed to allow no restarts.

4 Clause Learning and General Resolution

Lemma 1. Let F be a CNF formula over n variables. If F has a general resolution proof of size s , then it also has a clause learning proof of size at most ns using any non-redundant learning scheme and at most s restarts.

Proof. Let $\pi = (C_1, C_2, \dots, C_s \equiv \Lambda)$ be a general resolution proof of F where each C_i is either an initial clause or derived by resolving two clauses C_j and C_k , $j, k < i$, occurring earlier in π . If π contains a derived clause C_i whose strict subclause C'_i can be derived by resolving two previously occurring clauses, then we can replace C_i with C'_i , do trivial simplifications on further derivations that used C_i and obtain another proof π' of F of size at most s . Doing this repeatedly will remove all such redundant clauses and leave us with a simplified proof no larger in size. Hence we will assume without loss of generality that π has no such clause.

A clause learning proof of F can be constructed by choosing derived clauses of π in order, learning each of them, and restarting. Suppose every clause C_i , $i < p$, is already known and we are at decision level zero. This is trivially true when $C_1, \dots, C_{p-1}$ are initial clauses. If $p = s$, there are two known clauses x and $\neg x$ whose resolution generates $C_s \equiv \Lambda$. In this case we have a conflict from the known clauses at decision level zero and our clause learning proof is complete.

Otherwise, let $C_p = (l_1 \vee l_2 \vee \dots \vee l_k)$ and assume without loss of generality that C_p is derived by resolving two known clauses $(A \vee x)$ and $(B \vee \neg x)$, where both A and B are subclauses of C_p . Our clause learning proof will choose to branch on and set all of $l_1, l_2, \dots, l_k$ to FALSE. This will falsify both A and B and thus imply both x and $\neg x$ after unit propagation, resulting in a conflict. It is easy to see that the only new clause that can be learned from this conflict is C_p . Accordingly, we will use any non-redundant learning scheme, learn C_p and restart to get back to decision level zero. Due to the non-redundancy of C_p assumed earlier in the proof, there is no premature conflict until all of $l_1, l_2, \dots, l_k$ have been branched upon. This makes sure that our clause learning proof proceeds as described above and we learn precisely the clause C_p .

We learn at most s clauses and each learning stage requires branching on at most n variables and exactly one restart. This gives the desired bounds on the size of the constructed clause learning proof and the number of restarts it uses. $\square$

Lemma 2. *Let F be a CNF formula over n variables. If F has a clause learning proof of size s using any learning scheme and any number of restarts, then F also has a general resolution proof of size at most ns .*

Proof. Given a clause learning proof π of F , a resolution proof can be constructed by sequentially deriving all clauses that π learns, which includes the empty clause Λ . From Proposition 3, all these derivations are trivial and hence require at most n steps each. Consequently, the size of the resulting clause learning proof is at most ns . Note that since we derive clauses of π individually, restarts in π do not change the construction. $\square$

Combining Lemmas 1 and 2, we immediately get

Theorem 1. *Clause learning with any non-redundant scheme and unlimited restarts is equivalent to general resolution.*

Note that this theorem strengthens the result from [Baptista and Silva, 2000] that clause learning together with restarts is complete. Our theorem makes the stronger claim that clause learning with restarts can find proofs that are as short as those of general resolution.

5 Clause Learning and Regular Resolution

Here we prove that clause learning even without restarts is exponentially stronger than regular resolution on some formulas. We do this by first introducing a way of extending any CNF formula based on a given resolution proof of it. We then show that if a formula exponentially separates general resolution from regular resolution, its extension exponentially separates clause learning from regular resolution. Finally, we cite specific formulas called GT'_n that satisfy this property.

5.1 The Proof Trace Extension

Definition 9. Let F be a CNF formula and π be a resolution refutation of it whose last step resolves v with $\neg v$. Let $S = \pi \setminus (F \cup \{\neg v, \Lambda\})$. The *proof trace extension* $PT(F, \pi)$ of F is a CNF formula over variables of F and new trace variables t_C for clauses $C \in S$. The clauses of $PT(F, \pi)$ are all initial

clauses of F together with a trace clause $(\neg x \vee t_C)$ for each clause $C \in S$ and each literal $x \in C$.

We first show that if a formula has a short general resolution refutation, then the corresponding proof trace extension has a short clause learning proof. Intuitively, the new trace variables allow us to handle every resolution step of the original proof individually, effectively letting us *restart* after learning each derived clause.

Lemma 3. *Let F be any CNF formula and π be a resolution refutation of it. Then $PT(F, \pi)$ has a clause learning proof of size less than $size(\pi)$ using the FirstNewCut scheme and no restarts.*

Proof. Let v be the variable resolved upon in the last step of the resolution proof π and $S = \pi \setminus (F \cup \{\neg v, \Lambda\})$. Let $(C_1, C_2, \dots, C_k \equiv v)$ be the subsequence of π that has precisely the clauses in S . Assume without loss of generality as in the proof of Lemma 1 that π does not contain a derived clause C_i whose strict subclause C'_i can be derived by resolving two previously occurring clauses. We claim that the branching sequence $(t_{C_1}, t_{C_2}, \dots, t_{C_k})$ induces a clause learning proof of F using the FirstNewCut scheme.

Suppose $C_1 = (x_1 \vee x_2 \vee \dots \vee x_l)$ and assume without loss of generality that it was derived by resolving two previous clauses $(A \vee y)$ and $(B \vee \neg y)$, where both A and B are subclauses of C_1 . The first branch $t_{C_1} = \text{FALSE}$ followed by unit propagation results in implied literals $\neg x_1, \neg x_2, \dots, \neg x_l$ using trace clauses $(\neg x_i \vee t_{C_1})$. Further unit propagation using A and B implies y as well as $\neg y$ and we have a conflict. The cut in the conflict graph containing y and $\neg y$ on the conflict side and everything else on the reason side makes us learn C_1 as the FirstNewCut conflict clause. We now backtrack the branch on t_{C_1} and continue. Subsequent branches on $t_{C_2}, t_{C_3}, \dots, t_{C_k}$ similarly make us learn clauses $C_2, C_3, \dots, C_k$.

We have now learned (or have as initial clauses) the clause v as well as the two clauses used to derive $\neg v$ in π . These generate Λ in the residual formula, leading to a conflict without any branches and finishing the clause learning proof. Note that the maximum decision level in this proof is 1. $\square$

Lemma 4. *Let F be a CNF formula over n variables that has a polynomial (in n) size general resolution proof π but requires exponential size regular resolution proofs. Then $PT(F, \pi)$ has a polynomial size clause learning proof using the FirstNewCut scheme and no restarts, but requires exponential size regular resolution proofs.*

Proof. Lemma 3 immediately implies that $PT(F, \pi)$ has a polynomial size FirstNewCut clause learning proof.

For the other part, we use a simple reduction argument. Suppose $PT(F, \pi)$ has a regular resolution refutation π' of size s . Consider the restriction ρ that sets every trace variable of this formula to TRUE. ρ keeps original clauses of F intact and trivially satisfies all trace clauses, thereby reducing the initial clauses of $PT(F, \pi)$ to precisely F . Recall that regularity of resolution proofs is preserved under arbitrary restrictions. Consequently, applying ρ to π' gives us a regular resolution refutation of F of size at most s . By the

assumption in the Lemma, s must be exponential in n . Note that $PT(F, \pi)$ itself is of size polynomial in n because of our choice of π . Hence s is also exponential in the size of $PT(F, \pi)$ itself. $\square$

5.2 The GT'_n Formulas

We use the proof trace extension of an explicit family of unsatisfiable CNF formulas called GT'_n to obtain an exponential separation between regular resolution and clause learning. Note that in place of GT'_n , we could also have used any other formulas satisfying the conditions on F in Lemma 4, such as the modified pebbling formulas of Alekhovich *et al* [2002].

The GT'_n formulas are based on the ordering principle that any partial linear ordering on the set $\{1, 2, \dots, n\}$ must have a maximal element. The original formulas, called GT_n , were first considered by Krishnamurthy [1985] and later used by Bonet and Galesi [1999] to show the optimality of the size-width relationship of resolution proofs. Recently, Alekhovich *et al* [2002] used GT'_n to show an exponential separation between general and regular resolution. We refer the reader to this paper for exact specification of the GT'_n formulas. For our bound, we only need the following result:

Lemma 5 (Alekhovich *et al.*, 2002). *GT'_n has a polynomial size general resolution refutation but requires exponential size regular resolution proofs.*

Let $\pi_{GT'}$ be the polynomial size resolution refutation of GT'_n described in [Alekhovich *et al.*, 2002]. It follows from Lemmas 4 and 5 that $PT(GT'_n, \pi_{GT'})$ has a polynomial size clause learning proof using the FirstNewCut scheme but requires exponential size regular resolution proofs. Hence,

Theorem 2. *There exist CNF formulas on which clause learning using the FirstNewCut scheme and no restarts provides exponentially smaller proofs than regular resolution.*

6 Experimental Results

Table 1 reports the performance of variants of zChaff on grid pebbling formulas. We conducted experiments on a 1600 MHz Linux machine with memory limit set to 512MB. Base code of zChaff was extended to allow a (partial) branching sequence to be specified as part of the input. We used the more popular 1UIP learning scheme of zChaff instead of FirstNewCut because for these formulas, both schemes provide small proofs.

Pebbling formulas based on pebbling graphs are known to be hard for tree-like resolution but easy for regular resolution [Ben-Sasson and Wigderson, 1999]. We refer the reader to this paper for exact specification of the formulas. For our experiments, we worked with a uniform, grid like version, where every non-leaf node has precisely two predecessors. For a k layer graph, the corresponding formula has $O(k^2)$ variables and clauses. These formulas are minimally unsatisfiable. We also used a satisfiable version obtained by deleting a randomly chosen clause. The branching sequence was generated based on depth first traversal of the underlying pebbling graph. Results are reported for zChaff with no learning or specified branching sequence (DPLL), with specified branching sequence only, with clause learning only (original zChaff), and both.

Solver	Formula		Runtime (seconds)	
	layers	variables	unsat.	satisfiable
DPLL	5	30	0.24	0.12
	6	42	110	0.02
	7	56	> 24 hrs	0.07
	8	72	> 24 hrs	> 24 hrs
Branch sequence only	5	30	0.20	0.00
	6	42	105	0.00
	7	56	> 24 hrs	0.00
	9	90	> 24 hrs	> 24 hrs
Clause learning only	20	420	0.12	0.05
	40	1,640	59	36
	60	3,660	65	14
	65	4,290	‡	47
(original zChaff)	70	4,970	‡	‡
Clause learning and branch sequence	40	1,640	0.04	0.04
	100	10,100	0.59	0.62
	500	250,500	254	288
	1,000	1,001,000	4,251	5,335
	1,500	2,551,500	21,103	‡

Table 1: zChaff on pebbling formulas. ‡ denotes out of memory

7 Discussion and Open Problems

This paper has begun the task of formally understanding the power of clause learning from a proof complexity perspective. We have seen that clause learning can be more powerful than even regular resolution, and that learning with restarts yields general resolution.

In practice, a solver must employ good branching heuristics as well as implement a powerful proof system. Our result that modified pebbling formulas have small clause learning proofs depends critically upon the solver choosing a branching sequence that solves the formula in “bottom up” fashion, so that the learned clauses have maximal reuse. As we describe in a subsequent paper [Sabharwal *et al.*, 2003], this branching sequence can be efficiently generated by a combination of breadth-first and depth-first traversals of the original pebbling graph even for more general classes of pebbling formulas. As shown in Table 1, one needs both clause learning as well as a good branching sequence to efficiently solve large problem instances.

Of course, pebbling graphs, which correspond to problems involving precedence of tasks, represent a narrow domain of applicability. However, logic encodings of many kinds of real-world problems, such as planning graphs [Kautz and Selman, 1996], exhibit layered structure not unlike pebbling graphs. An important direction of our current research is to generate branching sequences that allow clause learning to work well on the general classes of structures that arise in encodings of particular problem domains.

Different learning schemes are likely to vary in their effectiveness when used on formulas from different domains. We introduced FirstNewCut as a new scheme and used it in this paper to derive our theoretical results. How well it performs on encodings of real-world problems is still open. It would be interesting to know if there is a class of practical problems

on which FirstNewCut works better than other schemes.

This paper inspires but leaves open several interesting questions of proof complexity. We have shown that with arbitrary restarts, clause learning is as powerful as general resolution. However, judging when to restart and deciding what branching sequence to use after restarting adds more non-determinism to the process, making it harder for practical implementations. Can clause learning with no or limited restarts also simulate general resolution efficiently? We showed that there are formulas on which clause learning is much more efficient than regular resolution. In general, can every small regular refutation be converted into a small clause learning proof? Or are regular resolution and clause learning incomparable?

References

- [Achlioptas *et al.*, 2001] D. Achlioptas, P. Beame, and M. Molloy. A sharp threshold in proof complexity. In *33rd STOC*, 337–346, 2001.
- [Alekhovich *et al.*, 2002] M. Alekhovich, J. Johannsen, T. Pitassi, and A. Urquhart. An exponential separation between regular and general resolution. In *34th STOC*, 448–456, 2002.
- [Baptista and Silva, 2000] L. Baptista and J. P. M. Silva. Using randomization and learning to solve hard real-world instances of satisfiability. In *Prin. and Prac. of Const. Prog.*, 489–494, 2000.
- [Bayardo Jr. and Schrag, 1997] R. J. Bayardo Jr. and R. C. Schrag. Using CST look-back techniques to solve real-world SAT instances. In *14th AAAI*, 203–208, 1997.
- [Ben-Sasson and Wigderson, 1999] E. Ben-Sasson and A. Wigderson. Short proofs are narrow – resolution made simple. In *31st STOC*, 517–526, 1999.
- [Ben-Sasson *et al.*, 2000] E. Ben-Sasson, R. Impagliazzo, and A. Wigderson. Near-optimal separation of treelike and general resolution. Tech. Rep. TR00-005, Elec. Colloq. in Comput. Compl., <http://www.eccc.uni-trier.de/eccc/>, 2000.
- [Biere *et al.*, 1999] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *36th DAC*, 317–320, 1999.
- [Bonet and Galesi, 1999] M. L. Bonet and N. Galesi. A study of proof search algorithms for resolution and polynomial calculus. In *40th FOCS*, 422–432, 1999.
- [Bonet *et al.*, 2000] M. L. Bonet, J. L. Esteban, N. Galesi, and J. Johansen, On the relative complexity of resolution refinements and cutting planes proof systems, *SIAM J. Comput.*, 30 (2000), 1462–1484.
- [Cook and Reckhow, 1977] S. A. Cook and R. A. Reckhow, The relative efficiency of propositional proof systems, *J. Symb. Logic*, 44 (1977), 36–50.
- [Davis and Putnam, 1960] M. Davis and H. Putnam, A computing procedure for quantification theory, *CACM*, 7 (1960), 201–215.
- [Davis *et al.*, 1962] M. Davis, G. Logemann, and D. Loveland, A machine program for theorem proving, *CACM*, 5 (1962), 394–397.
- [Davis, 1984] R. Davis, Diagnostic reasoning based on structure and behavior, *J. AI*, 24 (1984), 347–410.
- [de Kleer and Williams, 1987] J. de Kleer and B. C. Williams, Diagnosing multiple faults, *J. AI*, 32 (1987), 97–130.
- [Genesereth, 1984] R. Genesereth, The use of design descriptions in automated diagnosis, *J. AI*, 24 (1984), 411–436.
- [Gomes *et al.*, 1998a] C. P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *15th AAAI*, 431–437, 1998.
- [Gomes *et al.*, 1998b] C. P. Gomes, B. Selman, K. McAloon, and C. Trethoff. Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems. In *4th Int. Conf. Art. Intel. Planning Syst.*, 1998.
- [Kautz and Selman, 1996] H. A. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *13th AAAI*, 1194–1201, 1996.
- [Konuk and Larrabee, 1993] H. Konuk and T. Larrabee. Explorations of sequential ATPG using boolean satisfiability. In *11th VLSI Test Symposium*, 85–90, 1993.
- [Krishnamurthy, 1985] B. Krishnamurthy, Short proofs for tricky formulas, *Acta Inf.*, 22 (1985), 253–274.
- [Li and Anbulagan, 1997] C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *IJCAI (1)*, 366–371, 1997.
- [Marques-Silva and Sakallah, 1996] J. P. Marques-Silva and K. A. Sakallah. Grasp – a new search algorithm for satisfiability. In *ICCAD*, 220–227, 1996.
- [Marques-Silva, 1998] J. Marques-Silva. An overview of backtrack search satisfiability algorithms. In *5th Symp. on AI and Math.*, 1998.
- [Moskewicz *et al.*, 2001] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *38th DAC*, 530–535, 2001.
- [Sabharwal *et al.*, 2003] A. Sabharwal, P. Beame, and H. Kautz. Using problem structure for efficient clause learning. To appear in SAT, 2003.
- [Stallman and Sussman, 1977] R. Stallman and G. J. Sussman, Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis, *J. AI*, 9 (1977), 135–196.
- [Velev and Bryant, 2001] M. Velev and R. Bryant. Effective use of boolean satisfiability procedures in the formal verification of superscalar and vliw microprocessors. In *38th DAC*, 226–231, 2001.
- [Zhang and Hsiang, 1994] H. Zhang and J. Hsiang. Solving open quasigroup problems by propositional reasoning. In *Proceedings of the International Computer Symp.*, 1994.
- [Zhang *et al.*, 2001] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *ICCAD*, 279–285, 2001.
- [Zhang, 1997] H. Zhang. SATO: An efficient propositional prover. In *Int. Conf. on Auto. Deduc., LNAI*, vol. 1249, 272–275, 1997.

Phase Transitions of the Asymmetric Traveling Salesman*

Weixiong Zhang

Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO 63130, USA
Email: zhang@cse.wustl.edu

Abstract

We empirically study phase transitions of the asymmetric Traveling Salesman. Using random instances of up to 1,500 cities, we show that many properties of the problem, including the backbone and optimal tour cost, experience sharp transitions as the precision of intercity distances increases across a critical value. We also show that the average computational cost of the well-known branch-and-bound subtour elimination algorithm for the problem also exhibits a threshold behavior, transitioning from easy to difficult as the distance precision increases. These results provide strong positive evidences to a decade-long open question regarding the existence of phase transitions of the Traveling Salesman.

1 Introduction

Phase transition refers to such a phenomenon of a system in which some global properties change rapidly and dramatically when a control parameter crosses a critical value. A simple example of phase transition is water changing from the liquid phase to the solid phase when the temperature drops below the freezing point. Phase transitions of combinatorial problems and threshold behavior of combinatorial algorithms have drawn much attention in recent years [12]. It has been shown that many combinatorial decision problems have phase transitions, such as Boolean satisfiability [7; 21; 22], graph coloring [7] and number partitioning [4].

Another useful concept for characterizing combinatorial problems is that of backbones [17; 22]. A backbone variable refers to such a variable that has a fixed value in all solutions of a problem. All such backbone variables are collectively referred to as the backbone of the problem. The fraction of backbone, the percentage of variables in the backbone, reflects the constrainedness of the problem and directly affects an algorithm searching for a solution. The larger the backbone, the more tightly constrained the problem becomes.

*This research was supported in part by NSF grants IIS-0196057 and ITR/EIA-0113618, and in part by DARPA Cooperative Agreements F30602-00-2-0531 and F33615-01-C-1897. Thanks to Sharlee Climer for the joint work on an algorithm for finding backbones and Alan Frieze for a question on a bin-ball problem. Thanks also to Sharlee Climer, David Johnson, Scott Kirkpatrick and the anonymous reviewers for very helpful comments and suggestions to early versions of the paper.

In contrast to numerous phase-transition studies on decision problems, the research on the phase transitions and backbones of *optimization problems* is limited. An early work on the symmetric Traveling Salesman Problem, which is an optimization problem, introduced the concept of backbones and left an open question of whether there exists a phase transition of the TSP [17]. However, this question has not been addressed since 1985. One of the most rigorous phase-transition results was obtained on number partitioning [4], an optimization problem. However, the phase transition analyzed was the existence of a perfect partition of a set of integers, which is in essence a decision problem. In our early work, we studied the relationship between the phase transitions of satisfiability, a decision problem, and maximum satisfiability, an optimization problem [24]. In addition, the relationship between backbones and average-case algorithmic complexity has also been considered [23].

In this paper, we study the phase transitions of the asymmetric Traveling Salesman Problem (TSP), which has many real-world applications such as scheduling and routing. The TSP [13; 18] is an archetypical combinatorial optimization problem, and also very often a touchstone for combinatorial algorithms. In this paper, we consider the asymmetric TSP (ATSP), where the distance from one city to another may not be necessarily the same as the distance in the reverse direction. The ATSP is more general and difficult than the symmetric TSP (STSP). Our results provide strong positive evidences to the long-standing open question of [17] using the more general form of the problem. Another reason that we choose the ATSP rather than the symmetric TSP is that many properties of the assignment problem [19], a problem closely related to the ATSP, are known (Section 3.2) and useful to our analysis.

Specifically, using random problem instances of up to 1,500 cities, we empirically reveal that the average optimal tour length, the accuracy of the best cost function (the assignment problem), and the backbone of the ATSP undergo sharp phase transitions. The control parameter is the precision of intercity distances which is typically represented by the number of digits for the distances. Note that these results are algorithm independent and properties of the problem. Furthermore, we show that the average computational cost of the well-known branch-and-bound subtour elimination algorithm [1; 3] for the ATSP exhibits a threshold behavior, in which the computational cost grows abruptly and dramatically as the distance precision increases.

Two related results are worth mentioning. The research in [26] revealed that the average complexity of the subtour elim-

ination algorithm for the ATSP is controlled by the number of distinct intercity distances. We will further extend this result in Section 5. However, these results are algorithm specific, which may not necessarily reflect intrinsic features of the underlying problem. The research in [11] studied the decision version of the symmetric TSP. Specifically, it analyzed the probability that a tour of length less than a specific value exists for a random symmetric euclidean TSP, showing that the probability has a one-to-zero phase transition as the length of the targeting tour increases. Note that the phase-transition result in [11] does not address the open question of [17] which is on the optimization version of the problem.

2 The Problem and Algorithms

Given n cities and a matrix $D = (d_{i,j})$ that defines the distances or costs of pairs of cities, the *Traveling Salesman Problem* (TSP) is to find a minimum-cost complete tour visiting each city once. When the cost matrix is asymmetric, i.e., $d_{i,j}$ is not necessarily equal to $d_{j,i}$, the problem is the asymmetric TSP (ATSP), which is more difficult than the STSP, for both optimization and approximation [14].

The branch-and-bound (BnB) subtour elimination algorithm [1; 3] solves the ATSP using the assignment problem (AP) as a lower-bound cost function. The AP, which can be solved in $O(n^3)$, is to assign to each city i another city j with cost equal to distance $d_{i,j}$, so that the total cost of all such assignments is minimized [19]. If the AP solution happens to be a complete tour, it is also an ATSP solution.

The BnB search takes the ATSP as the root of the state space and solves the AP to the root node. If the AP solution is not a complete tour, decompose it into subproblems by selecting a subtour from the AP solution and generate subproblems by excluding some edges in the subtour to eliminate the subtour. We used the Carpaneto-Toth subtour-elimination heuristic [6] in our implementation, which generates no duplicate subproblem, so that the search space is a tree. Next, select as the current problem a new subproblem that has been generated but not yet expanded. This process continues until there is no unexpanded problem, or all unexpanded problems have costs greater than or equal to the cost of the best complete tour found so far.

Our algorithm is, in principle, the same as that of [5], which is probably the best known complete algorithm for the ATSP. The main difference between the two is that we use depth-first branch-and-bound (DFBnB), due to its low space requirement, while [5] used best-first search. We further extended our DFBnB algorithm to finding all optimal solutions and backbones.

3 Previous, Related Results

Two previous lines of work influenced this research and helped reveal the properties of the ATSP.

3.1 Phase transitions in tree search

To capture the tree search of the DFBnB algorithm for the ATSP, we first introduce an abstract tree model. An *incremental tree* [15; 20; 25], $T(m, d)$, is a tree with depth d , and independent and identically distributed (i.i.d.) random branching factors with mean m . Edges are assigned costs that are finite and nonnegative i.i.d. variables. The cost of a node is the sum of the edge costs along the path from the root to that node. An edge cost is nothing but the difference between a node cost and

the cost of its parent. An *optimal goal* is a node of minimum cost at depth d . The overall goal is to find an optimal goal node.

Let p_0 be the probability of a node having the same cost as its parent. The expected number of child nodes of a node that have the same cost as their parent is mp_0 . The following results have been proven [15; 20; 25]. Let C^* be the expected optimal goal cost of $T(m, d)$ with $m > 1$. Let N_{BFS} and N_{DFS} be the expected numbers of nodes expanded by best-first search and DFBnB, respectively. As $d \rightarrow \infty$, (1) $C^*/d \rightarrow \alpha_1$ and $N_{BFS} = N_{DFS} = \theta(\alpha_2^d)$ almost surely when $mp_0 < 1$, where α_1 and α_2 are constants; (2) $C^*/(\log \log d) \rightarrow 1$ and $N_{BFS} = \theta(d^2)$ and $N_{DFS} = O(d^3)$ almost surely when $mp_0 = 1$; (3) C^* remains bounded by a constant and $N_{BFS} = \theta(d)$ and $N_{DFS} = O(d^2)$ almost surely when $mp_0 > 1$.

The above results mean that the cost of an optimal goal node almost surely undergoes a phase transition from a linear function of depth d to a constant when mp_0 increases beyond one. Meanwhile, the expected complexity of best-first search and DFBnB changes dramatically from exponential to polynomial in d as mp_0 is reduced below one.

3.2 Relationship between AP and ATSP

The assignment problem (AP) cost function and its relationship with the ATSP have been a research interest for a long time [8; 9; 10; 16]. The relationship between the AP cost, $AP(D)$, and the ATSP cost, $ATSP(D)$, has remarkably different characteristics under different distance distributions of a random matrix D . The $AP(D)$ and $ATSP(D)$ can be the same with a high probability, or they can differ from each other, with a high probability, by a function of n . If the expected number of zeros in a row of D approaches infinity when $n \rightarrow \infty$, $AP(D) = ATSP(D)$ with a probability tending to one [9]. However, if the distances are uniform over $[0, 1, \dots, \lfloor c_n n \rfloor]$, $AP(D) = ATSP(D)$ with a probability going to zero, where c_n grows to infinity with n [9]. Indeed, when the entities of D are uniform over $[0, 1]$, $E(ATSP(D) - AP(D)) \geq c_0/n$, where c_0 is a positive constant [10].

4 Phase Transitions of the ATSP

The results in the previous section indicate that the quality of the AP function varies significantly, depending on the underlying distance distribution. Precisely, the difference between the AP cost, $AP(D)$, and the ATSP cost, $ATSP(D)$, has two phases, controlled by the number of zero distances in the distance matrix D . In one phase, the difference is zero with high probability, while in the other phase, the expectation of the difference is a function of the problem size n .

How does the difference between $ATSP(D)$ and $AP(D)$ change phases? Does it have a sharp phase transition, or does it follow a slow process? Do other properties of the ATSP, such as the backbone, also have phase transitions?

The two-phase result on the accuracy of the AP cost function discussed in Section 3.2 is in principle consistent with the phase-transition result of incremental random trees in Section 3.1. The root of the search tree has a cost equal to the AP cost $AP(D)$ to the problem and an optimal goal node has the ATSP tour cost $ATSP(D)$. If we subtract the AP cost to the root from every node in the ATSP search tree, the root node has cost zero and an optimal goal node has cost equal to $ATSP(D) - AP(D)$. When there are a large number of zero distances in D , the AP cost of a child node in a search tree is

more likely to be the same as the AP cost of its parent, since AP will tend to use the zero distances. Therefore, it is expected that more nodes in the search tree will have more than one child node having the same cost as their parents. In fact, the concept of zero distances can be extended to minimal-cost distances. If the cost of the minimal distance is known, we can simply subtract it from every distance without affecting the accuracy of the AP function.

4.1 The control parameter

Furthermore, the concept and role of zero and minimal distances can be extended to that of equal distances. The AP to a child node is obtained by excluding one arc, and possibly including some arcs, of a subtour of the AP solution to the parent node. With this new restriction, the child AP is then computed by constructing a new augmenting path from the starting point to the end point of the excluded arc [19]. The child and parent AP solutions usually differ by only a relatively small number of arcs. In other words, the child AP solution is derived by replacing some arcs in the parent AP solution. If the numbers of distances of equal values are large, it is more likely that the child AP cost is equal to the parent AP cost. Conversely, when the number of distinct distances is large, it is unlikely the costs of two APs will be the same. When the distances are uniform over $[0, 1, \dots, R - 1]$, the probability that the child AP cost is equal to the parent AP cost will depend on the range R . If R is small, relative to the problem size n , this probability will be high. In short, for a given problem size, the accuracy of the AP cost function is controlled by the number of distinct distances in matrix D . More precisely, the accuracy of the AP is determined by the fraction, denoted as $\rho(n)$, of distinct distances.

In practice, however, we do not directly control the number or the fraction of distinct distances. In addition to the actual structures of the “layouts” of the cities, the precision of the distances also affects the number of distinct distances. The precision of a number is usually represented by the maximal number of digits allowed for the number. As a result, the number of digits for distances is naturally a good choice for the control parameter.

The effect of a given number of digits on the fraction of distinct distances is relative to the problem size n . Consider a matrix D with distances uniformly over $\{0, 1, 2, \dots, R - 1\}$, where the range R is determined by the number of digits b . For a fixed b , the fraction of distinct distances of a larger matrix D is obviously smaller than that of a smaller D . Thus, the control parameter for the fraction $\rho(n, b)$ of distinct distances must be in the form of $b \cdot f(n)$, where $f(n)$ is a rescaling function on the effect of the number of digits.

To find the scaling function $f(n)$, consider the number of distinct distances of matrix D for a given integer range R . The problem of finding the number of distinct distances is equivalent to the bin-ball problem as follows. We are given M balls and R bins, and asked to place the balls into the bins. Each ball is independently put into one of the bins with an equal probability. We are interested in the fraction of bins that are not empty after all the placements. Here, for asymmetric TSP $M = n^2 - n$ balls correspond to the total number of nondiagonal distances of matrix D , and R bins represent the possible integers to be selected from. Since each ball (distance) is thrown independently and uniformly into one of R bins (integers), the probability that one bin is not empty after throwing

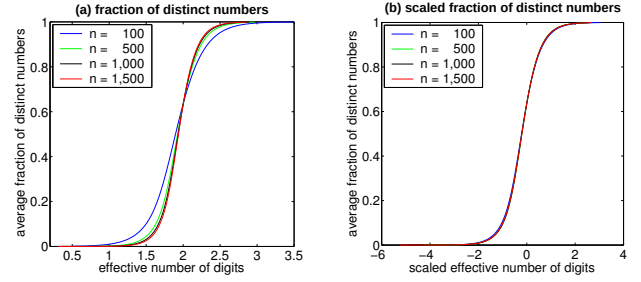


Figure 1: (a) Average fraction of distinct distances in matrix D , $\rho(n, b)$, controlled by the effective number of digits, $\beta = b \log_{10}^{-1}(n)$. (b) Average $\rho(n, b)$ after finite-size scaling. $\beta_0 = 2.00195 \pm 0.00025$ and $v = 6.4350 \pm 0.0003$.

M balls is $1 - (1 - 1/R)^M$. The expected number of occupied bins (distinct distances) is simply $R(1 - (1 - 1/R)^M)$. Thus, the expected fraction of distinct distances in matrix D is $E[\rho(n, b)] = R(1 - (1 - 1/R)^M) / M$. Note that if M or n is fixed, $E[\rho(n, b)] \rightarrow 1$ as $R \rightarrow \infty$, since in this case the expectation of the number of distinct distances approaches M . On the other hand, when R is fixed, $E[\rho(n, b)] \rightarrow 0$ when M or n goes to infinity, since all of a finite number of R bins will be occupied by an infinite number of balls in this case. We use decimal values for distances in this paper. Thus $R = 10^b$, where b is the number of digits for distances, and we can rewrite $E[\rho(n, b)]$ as $E[\rho(n, b)] = 10^b(1 - (1 - 1/10^b)^M) / M$.

It turns out that if we plot $\rho(n, b)$ against $b \log_{10}^{-1}(n)$, it will have relatively the same scale for different problem sizes n , as shown in Figure 1(a). This means that the scaling function for the effective number of digits is $f(n) = \log_{10}^{-1}(n)$. Function $b \log_{10}^{-1}(n)$ is thus the *effective number of digits* that controls the fraction of distinct distances in matrix D , which we denote as $\beta(n, b)$. To have the same effective number of digits β on two different problem sizes, say n_1 and n_2 with $n_1 < n_2$, the range R should be different. On these two problems, R needs to be n_1^β and n_2^β , respectively, giving $n_1^\beta < n_2^\beta$.

Note that R can also be represented by a number in other bases, such as binary. Which base to use will not affect the results quantitatively, but introduces a constant factor to the results. In fact, since $b = \log_{10}(R)$, the control parameter $\beta(n, b) = b \log_{10}^{-1}(n) = \log_n(R)$, which is independent of the base for intercity distances.

Controlled by the effective number of digits $b \log_{10}^{-1}(n)$, the fraction of distinct distances ρ has a phase transition of its own, also shown in Figure 1(a). The larger the problem, the sharper the transition, and there exists a crossover point among the transitions of problems with different sizes. We can examine the phase transitions more closely using finite-size scaling. Finite-size scaling [2] is a method that has been successfully applied to phase transitions in similar systems of different sizes. Based on finite-size scaling, around a critical parameter (temperature), problems of different sizes tend to be indistinguishable except for a change of scale given by a power law in a characteristic length. Thus, finite-size scaling can help to characterize a phase transition precisely around a critical point of the control parameter as the problem scales to infinity. For the problem at hand, the effective

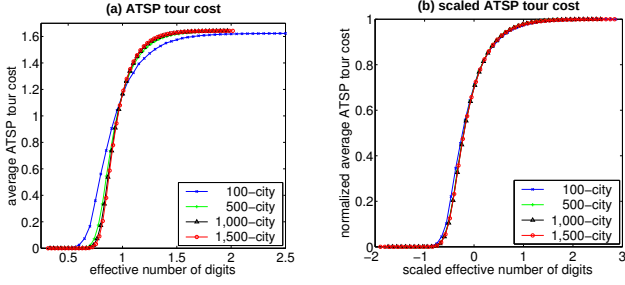


Figure 2: (a) Average optimal ATSP tour cost. (b) Scaled and normalized average optimal tour cost. $\beta_0 = 0.995 \pm 0.002$ and $v = 7.143 \pm 0.005$.

number of distinct distances $\beta = b \log_{10}^{-1}(n)$ and the problem size n play the roles of the temperature and the characteristic length, respectively. This means that as n goes to infinity, the control parameter becomes $(\beta - \beta_0)n^{1/v}$, where β_0 is the critical point and $n^{1/v}$ specifies the change of scale. Using $E[\rho(n, b)] = R(1 - (1 - 1/R)^M)/M$ and numerical methods, we obtained the critical value $\beta_0 = 2.00195 \pm 0.00025$ and rescaling component $v = 6.4350 \pm 0.0003$, where the error bounds represent the 95% confidence intervals. The rescaled phase transitions are shown in Figure 1(b).

Note that the number of digits used for intercity distances is nothing but a measurement of the precision of the distances. The larger the number of digits, the higher the precision becomes. This agrees with the common practice of using more effective digits to gain precision. Therefore, the phase transition of the control parameter is in turn determined by the precision of intercity distances.

4.2 Phase transitions

With the control parameter, the effective number of digits $\beta(n, b)$ for intercity distances, identified, we are now in a position to investigate possible phase transitions in (1) the ATSP cost, (2) the probability that an AP cost is equal to the corresponding ATSP cost, (3) the relative error or accuracy of the AP lower-bound function, and finally (4) the backbone of the ATSP. The answers to the first three problems provides a detailed picture on the accuracy of the AP cost function, and the answer to the last problem reveals the intrinsic constrainedness among the cities as the precision of distances changes. We examine these four problems in turn.

We generated uniformly random problem instances of 100-, 200-, 300- to 1,000-cities and 1,500-cities. Intercity distances are independently and uniformly chosen from $\{0, 1, 2, \dots, R-1\}$ for a given range R , which is controlled by the number of digits b . We varied b , with an increment of 0.1, from 1.0 to 6.0 for instances with up to 1,000-cities and from 1.0 to 6.5 for instances with 1,500-cities. For each combination of n and b , we generated 10,000 instances for problems (1)-(3) listed above, and 1,000 instances for problem (4) due to its high computational cost. To make the result figures readable, we only use the curves from 100-, 500-, 1,000- and 1,500-city problems.

There is a phase transition in the ATSP tour cost, $ATSP(D)$, shown in Figure 2(a). The reported tour costs are obtained by dividing the integer tour costs from the DFBnB algorithm by $n \times (R - 1)$, where n is the number of cities and R

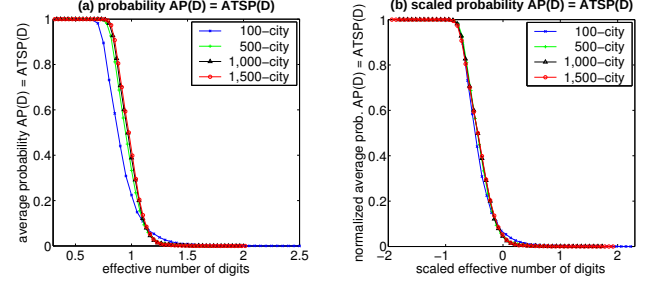


Figure 3: (a) Average probability that $AP(D) = ATSP(D)$. (b) Average probability after finite-size scaling. $\beta_0 = 1.169 \pm 0.02$, and $v = 8.929 \pm 0.004$.

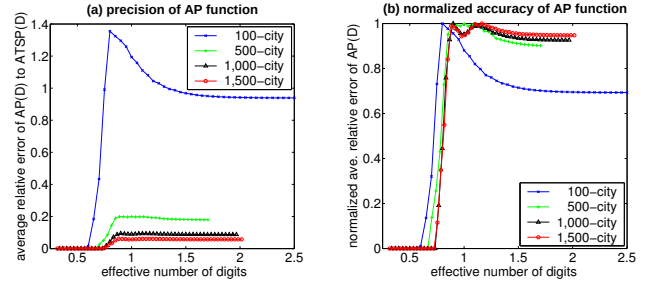


Figure 4: (a) Average accuracy of AP lower-bound function, measured by the error of AP cost relative to ATSP cost. (b) Normalized average accuracy.

the range of intercity costs. Equivalently, an intercity distance was virtually converted to a real value in $[0, 1]$. By doing this, we can verify the existing analytical result on the AP and ATSP costs. This will be discussed in detail in the next subsection.

As shown, the ATSP tour cost increases abruptly and dramatically as the effective number of digits increases, exhibiting phase transitions. The transitions become sharper as the problem becomes larger, and there exist crossover points among curves from different problem sizes. By finite-size scaling, we further determine the critical value of the control parameter at which the phase transitions occur. The scaled result is shown in Figure 2(b). It is worthwhile to mention that the AP cost (not shown) follows almost the same phase-transition pattern.

Our numerical results show that when the number of digits for intercity distances is very small, for example, less than 1.9 digits or $R \approx 80$ for $n = 1,500$, the AP and ATSP costs are equal to zero, meaning that these two costs are the same as well. Given a random distance matrix D , how likely is it that an AP cost will differ from the ATSP tour cost as the effective number of digits β increases? We address this question by examining the probability that an AP cost $AP(D)$ is equal to the corresponding ATSP cost $ATSP(D)$ as β increases. Figure 3(a) shows the results, averaged over the same set of instances for Figure 2. As shown, the probability that $AP(D) = ATSP(D)$ also experiences phase transitions. Figure 3(b) shows the phase transitions after finite-size scaling.

The results in Figure 3 also imply that the quality of the AP function degrades as the effective number of digits β increases. The degradation also follows a phase-transition process. This is verified by Figure 4, using the data from the same set of

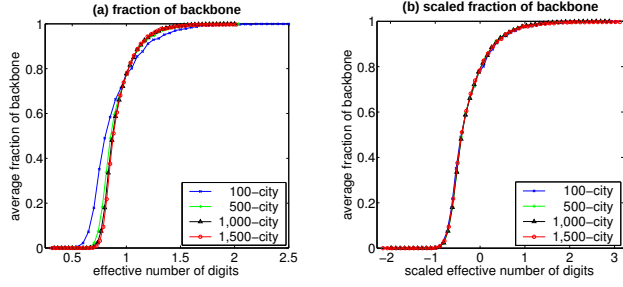


Figure 5: (a) Average fraction of backbone. (b) Scaled backbone fraction. $\beta_0 = 1.012 \pm 0.007$, $v = 6.452 \pm 0.002$.

n	digits	AP cost	ATSP cost	relative AP error (%)
2100	8.6810	1.6435 ± 0.0008	1.6444 ± 0.0008	0.0402 ± 0.0009
2300	8.7555	1.6439 ± 0.0007	1.6445 ± 0.0008	0.0368 ± 0.0008
2500	8.8235	1.6441 ± 0.0007	1.6442 ± 0.0007	0.0336 ± 0.0008
2700	8.8861	1.6438 ± 0.0007	1.6444 ± 0.0007	0.0309 ± 0.0007
2900	8.9440	1.6439 ± 0.0007	1.6444 ± 0.0007	0.0286 ± 0.0007

Table 1: Results on AP cost, the ATSP cost and AP error relative to the ATSP cost, on random ATSP. Each datum is averaged of 10,000 instances. All error bounds represent 95 percent confidence intervals.

problem instances for the previous two figures.

We now turn to the backbone of the ATSP, which is the fraction of directed arcs that appear in all optimal solutions. The backbones also exhibit phase transitions as the effective number of digits for distances increases. The result is included in Figure 5. Interestingly, the phase-transition pattern of the backbone is almost identical to that of the fraction of distinct entities in the distance matrix, shown in Figure 1, and that of the ATSP tour cost, shown in Figure 2.

The fraction of backbone captures, in essence, the tightness of the constraints among the cities. As more intercity distances become distinct, the number of tours of unique lengths increases. Consequently, the number of optimal solutions decreases and the fraction of backbone grows inversely. When more arcs are part of the backbone, optimal solutions become more restricted and the number of optimal solutions decreases, making it more difficult to find an optimal solution.

4.3 Asymptotic AP precision

As a by-product of the phase-transition results, we now provide asymptotic values of the ATSP cost, the AP cost and its accuracy. We attempt to extend the previous theoretical results on the AP cost, which is known to be within $(1.51, 1.94)$ [8], and the observation that the accuracy of the AP lower bounds increases as the problem size increases [1].

We need to be cautious in selecting the number of digits for intercity distances. As discussed earlier, the same number of digits for distances gives rise to different effective numbers of digits on problems of different sizes. Therefore, the number of digits must be scaled properly to have the same effect on problems of different sizes when we exam an asymptotic feature.

Therefore, in our experiments, we fixed the scaled effective number of digits β to a constant. Based on the phase-transition of the control parameter in Figure 1, we took $\beta = (b \log_{10}^{-1}(n) - 2.002)n^{1/6.435}$ a constant of 2. $\beta = 2$ is suf-

ficiently large so that all distances are distinct, regardless of problem size, and the quantities to be examined do not change substantially after finite-size scaling. $\beta = 2$ is also relatively small so that we can experiment on large problems. In our implementation of the DFBnB algorithm, distances are integers of 4 bytes. Thus the number of digits must be less than 9.4 without causing an overflow in the worst case. Using $\beta = 2$, we can go up to roughly 2,500-city ATSPs in the worst case.

Table 1 shows the results, with up to 2,900 cities. The AP cost approaches to 1.6439 and the ATSP cost to 1.6444. The accuracy of AP function indeed improves as the problem size increases, with relative error reduced to about 0.0286% for 2,900-city problems.

5 Threshold Behavior of Subtour Elimination

The phase-transition results indicate that the ATSP becomes more constrained and difficult as the distance precision becomes higher. We now study how the DFBnB subtour elimination algorithm behaves. We separate this issue from the phase transitions studied before because we now consider the behavior of a particular algorithm, which may not be necessarily a feature of the underlying problem. Nevertheless, this is still an issue of its own interest because this algorithm is among the best known methods for the ATSP, and we hope that a better understanding of this algorithm can shed light on the typical case complexity of the problem.

Figure 6(a) shows the average complexity of the DFBnB algorithm, measured by the number of calls to the AP function. The result is averaged over the same problem instances for each data point as used for the phase transitions in the previous section. Note that the number of AP calls increases significantly from small problems to large ones using the same effective number of digits for distances. Thus, we normalize the result in such a way that for a given problem size, the minimal and maximal AP calls among all problem instances of the same size are mapped to zero and one, respectively, and the other AP calls are proportionally adjusted to a ratio between 0 and 1. This allows us to compare the results from different problem sizes on one figure. The curves in Figure 6(a) follow a pattern similar to that of the phase transitions in the previous section. The complexity of the subtour elimination algorithm increases with the effective number of digits, and exhibits a threshold behavior similar to phase transitions. Indeed, we can use finite-size scaling to capture the behavior as the problem size grows, as illustrated in Figure 6(b). The results in Figure 6 and the results in the previous section indicate that the complexity of the subtour elimination algorithm goes hand-in-hand with the constrainedness of the problem.

Similar results have been reported in [26]. The results of this subsection extend that in [26] to different sizes of problems and by applying finite-size scaling to capture the threshold behavior as problem size increases.

6 Conclusions and Discussions

Our main contributions are twofold. First, we provided strong positive evidences to the long-standing question of whether the Traveling Salesman Problem (TSP) has phase transitions [17]. We studied this issue on the more general asymmetric TSP (ATSP). We empirically showed that many properties, including the ATSP tour cost and the fraction of backbone, have

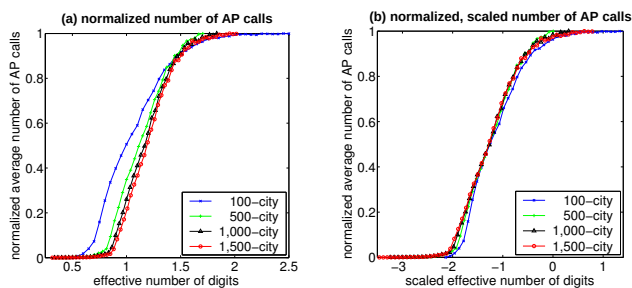


Figure 6: (a) Normalized average number of AP calls of DF-BnB subtour elimination. (b) Scaled average number of AP calls. $\beta_0 = 1.702 \pm 0.045$ and $v = 8.015 \pm 0.032$.

two characteristically different values, and the transitions between them are rather abrupt and dramatic, displaying phase-transition phenomena. The control parameter of the phase transitions is the effective number of digits representing the intercity distances, which in essence measures the precision of distances. Our results revealed the connection between distance precision and phase transition properties in the ATSP. Distance precision is the control parameter for various phase transitions of the ATSP. We believe that the concept of precision determining problem properties such as phase transitions is rather universal and may very well be applicable to other problems, including various scheduling and planning problems.

Second, our phase transitions results provide a practical guidance to how to generate difficult random ATSP problem instances and which instances to use to compare the asymptotic performance of two algorithms. A common practice in comparing algorithms when using a random ensemble is to generate problems with a fixed distance precision. Our results imply that the correct way is to use problem instances of different sizes that have the same or similar features such as the same fraction of backbones. This, in turn, requires to increase the number of digits for intercity distances as the problem size grows. We believe that this guidance is general and can be applied to other optimization problems.

References

- [1] E. Balas and P. Toth. Branch and bound methods. In *The Traveling Salesman Problem*, pages 361–401. John Wiley & Sons, Essex, England, 1985.
- [2] M. N. Barber. Finite-size scaling. In *Phase Transitions and Critical Phenomena*, volume 8, pages 145–266. 1983.
- [3] M. Bellmore and J. C. Malone. Pathology of traveling-salesman subtour-elimination algorithms. *Operations Research*, 19:278–307, 1971.
- [4] C. Borgs, J. T. Chayes, and B. Pittel. Phase transition and finite-size scaling for the integer partitioning problem. *Random Structures and Algorithms*, 19:247–288, 2001.
- [5] G. Carpaneto, M. Dell’Amico, and P. Toth. Exact solution of large-scale, asymmetric Traveling Salesman Problems. *ACM Trans. on Mathematical Software*, 21:394–409, 1995.
- [6] G. Carpaneto and P. Toth. Some new branching and bounding criteria for the asymmetric traveling salesman problem. *Management Science*, 26:736–743, 1980.
- [7] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proc. IJCAI-91*, pages 331–337.
- [8] D. Coppersmith and G. B. Sorkin. Constructive bounds and exact expectations for the random assignment problem. *Random Structures and Algorithms*, 15:113–144, 1999.
- [9] A. Frieze, R. M. Karp, and B. Reed. When is the assignment bound asymptotically tight for the asymmetric traveling-salesman problem? In *Proc. of Integer Programming and Combinatorial Optimization*, pages 453–461, 1992.
- [10] A. Frieze and G. B. Sorkin. The probabilistic relationship between the assignment and asymmetric traveling salesman problems. In *Proc. of SODA-01*, pages 652–660, 2001.
- [11] I. Gent and T. Walsh. The TSP phase transition. *Artificial Intelligence*, 88:349–358, 1996.
- [12] C. P. Gomes, T. Hogg, T. Walsh, and W. Zhang. IJCAI-2001 tutorial: Phase transitions and structure in combinatorial problems. <http://www.cs.wustl.edu/~zhang/links/ijcai-phase-transitions.html>.
- [13] G. Gutin and A. P. Punnen, editors. *The Traveling Salesman Problem and Its Variations*. Kluwer Academic Publishers, 2002.
- [14] D. S. Johnson, G. Gutin, L. A. McGeoch, A. Yeo, W. Zhang, and A. Zverovich. Experimental analysis of heuristics for the ATSP. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 445–488. 2002.
- [15] R. M. Karp and J. Pearl. Searching for an optimal path in a tree with random costs. *Artificial Intelligence*, 21:99–117, 1983.
- [16] R. M. Karp and J. M. Steele. Probabilistic analysis of heuristics. In *The Traveling Salesman Problem*, pages 181–205. 1985.
- [17] S. Kirkpatrick and G. Toulouse. Configuration space analysis of traveling salesman problems. *Journale de Physique*, 46:1277–1292, 1985.
- [18] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem*. John Wiley & Sons, Essex, England, 1985.
- [19] S. Martello and P. Toth. Linear assignment problems. *Annals of Discrete Mathematics*, 31:259–282, 1987.
- [20] C. J. H. McDiarmid. Probabilistic analysis of tree search. In *Disorder in Physical Systems*, pages 249–260. 1990.
- [21] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proc. AAAI-92*, 1992.
- [22] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic ‘phase transitions’. *Nature*, 400:133–137, 1999.
- [23] J. Slaney and T. Walsh. Backbones in optimization and approximation. In *Proc. IJCAI-01*, pages 254–259, 2001.
- [24] W. Zhang. Phase transitions and backbones of 3-SAT and Maximum 3-SAT. In *Proc. CP-2001*, pages 153–167, 2001.
- [25] W. Zhang and R. E. Korf. Performance of linear-space search algorithms. *Artificial Intelligence*, 79:241–292, 1995.
- [26] W. Zhang and R. E. Korf. A study of complexity transitions on the asymmetric Traveling Salesman Problem. *Artificial Intelligence*, 81:223–239, 1996.

SCHEDULING

Contract Algorithms and Robots on Rays: Unifying Two Scheduling Problems

Daniel S. Bernstein

Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
bern@cs.umass.edu

Lev Finkelstein

Computer Science Department
Technion—IIT
Haifa 32000, Israel
lev@cs.technion.ac.il

Shlomo Zilberstein

Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
shlomo@cs.umass.edu

Abstract

We study two apparently different, but formally similar, scheduling problems. The first problem involves contract algorithms, which can trade off run time for solution quality, as long as the amount of available run time is known in advance. The problem is to schedule contract algorithms to run on parallel processors, under the condition that an interruption can occur at any time, and upon interruption a solution to any one of a number of problems can be requested. Schedules are compared in terms of acceleration ratio, which is a worst-case measure of efficiency. We provide a schedule and prove its optimality among a particular class of schedules. Our second problem involves multiple robots searching for a goal on one of multiple rays. Search strategies are compared in terms of time-competitive ratio, the ratio of the total search time to the time it would take for one robot to traverse directly to the goal. We demonstrate that search strategies and contract schedules are formally equivalent. In addition, for our class of schedules, we derive a formula relating the acceleration ratio of a schedule to the time-competitive ratio of the corresponding search strategy.

1 Introduction

In this paper, we demonstrate a connection between two problems that initially seem unrelated. The first involves computing solutions to multiple problems, under the condition that a solution to any one of the problems can be requested at any time. Challenges of this type arise in the design of intelligent user interfaces, information prefetching systems, and medical diagnosis systems. The second problem involves multiple robots searching an unknown environment for a goal. Problems of this nature arise in robotics and space exploration. In the following paragraphs we describe the problems, along with our contribution, in more detail.

The first problem concerns *anytime* algorithms [Horvitz, 1987; Dean and Boddy, 1988; Russell and Zilberstein, 1991], which produce solutions of different qualities depending on available computation time. More specifically, we focus on

contract algorithms, which are anytime algorithms that require the deadline as input prior to the start of execution. With contract algorithms, no assumptions can be made about results produced before the given deadline. This is in contrast to the familiar *interruptible* algorithms, which can be queried at any point during execution. Although less flexible than interruptible algorithms, contract algorithms typically use simpler data structures, making them easier to implement and maintain. An example in AI is game playing programs based on heuristic search. For these programs, the allowed deliberation time is usually known in advance, and is used to set internal parameters. Another example is planning algorithms that perform state-space abstraction. With these algorithms, the run time can be controlled by setting the abstraction level at the start of execution.

Our problem can be stated as follows. We are given n instances of an optimization problem, along with a contract algorithm for the problem, and we have an m -processor machine on which to run the algorithm. An interruption can occur at any time, and a solution can be requested for any one of the problem instances. Given these constraints, we want a good general strategy for scheduling runs of the algorithm on the processors.

In the case of one problem instance and one processor, Russell and Zilberstein [1991] suggested iteratively doubling the contract lengths. With this schedule, for any interruption time t , the last contract completed (if one exists) is always of length at least $t/4$. This factor of four is the *acceleration ratio* of the schedule, a worst-case measure of its efficiency. Zilberstein *et al.* [1999] showed that no schedule can achieve an acceleration ratio less than four.

The generalization to multiple problem instances has been considered [Zilberstein *et al.*, 1999], as has the generalization to scheduling contracts on parallel processors [Bernstein *et al.*, 2002]. Optimal acceleration ratios have been derived in both cases. The more general multi-processor, multi-instance case has not previously been studied. In this paper, we provide a schedule for this case, and we prove that this schedule is optimal among a restricted, though still interesting, class of schedules. The optimality proof is a nontrivial extension of the previous proofs, and contains as a lemma a generalization of the monotone convergence principle.

This work is most closely related to Horvitz's continual computation framework [Horvitz, 2001]. In his framework,

as in ours, computation is performed with limited knowledge about the deadline or desired result. However, the assumptions underlying the two frameworks are different. In the continual computation framework, the limited knowledge comes in the form of probability distributions. In contrast, our use of acceleration ratio does not require probabilistic information. Furthermore, contract algorithms and parallel processing are both not considered in continual computation.

Our contract scheduling results can be directly applied to a robot search problem. In this problem, m robots search for a goal that is located on one of p intersecting rays. The aim is to minimize the *time-competitive ratio*, which is the worst-case ratio of the total time spent searching to the time for a single robot to traverse directly to the goal. The optimal ratio for the one-robot case was derived previously [Baeza-Yates *et al.*, 1993].

We address the general multi-robot case, for which search strategies are formally equivalent to contract schedules. For our class of schedules, we derive a formula relating the acceleration ratio of a schedule to the time-competitive ratio of the corresponding search strategy. The optimal time-competitive ratio is derived as a corollary.

This work is the first to draw a precise connection between contract scheduling and multi-robot search, and the first to provide nontrivial results for the multi-robot case. Kao *et al.* [1998] studied the multi-robot problem, but used a different performance measure. They minimized the *distance-competitive ratio*, which is the worst-case ratio of the total distance traveled during the search to the distance between the origin and the goal.

2 Scheduling a Contract Algorithm

2.1 Problem Description

An anytime algorithm A , when applied to an optimization problem instance i for time t , produces a solution of some real-valued quality $Q_A(t, i)$. The function Q_A is called A 's *performance profile*. In general, one does not know an algorithm's performance profile. Nevertheless the concept of a performance profile is useful in reasoning about anytime algorithms. We assume that the performance profile of an anytime algorithm on any problem instance is defined for all $t \geq 0$ and is a nondecreasing function of t .

The distinctions among different types of anytime algorithms arise from different assumptions about which parameters are known prior to execution. When both t and i are known in advance, the algorithm is called *contract*. When only i is known in advance, the algorithm is called *interruptible*. For the case where both are unknown, we will say that the algorithm is *multi-interruptible*, because it acts like multiple interruptible algorithms running in parallel.

Suppose we have a contract algorithm A , which we can run on a machine with m processors. At some unknown deadline, a solution to one of n problem instances will be requested. This setup requires a multi-interruptible algorithm, which we can create by scheduling contracts in such a way that progress is continually made on each problem instance. Upon interruption and query, the result returned is that of the longest completed contract dedicated to the desired problem instance.

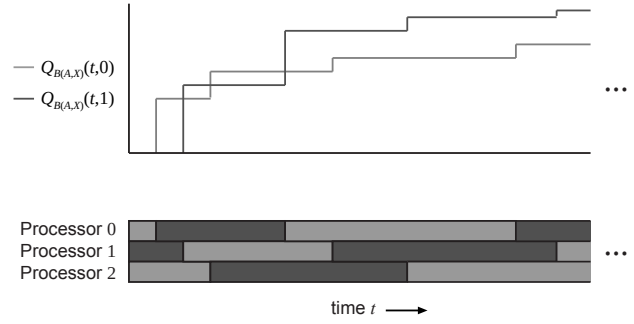


Figure 1: Scheduling a contract algorithm on three processors to create a two-problem multi-interruptible algorithm.

The multi-interruptible algorithm's performance profile depends on A 's profile and the schedule, in a way to be formalized shortly. A particular schedule for the case of $m = 3$ and $n = 2$ is illustrated in Figure 1.

Formally, a *schedule* is a triple $X = \langle \{P_k\}, \{I_k\}, \{L_k\} \rangle$, where $P_k \in \{0, \dots, m-1\}$ is the processor of contract k , $I_k \in \{0, \dots, n-1\}$ is the problem instance worked on by contract k , and $L_k \in \mathbb{R}^+$ is the length of contract k . The contract index will be assumed to start at zero. A schedule must satisfy some minor conditions, which require a couple more definitions to state. First, let δ be defined as

$$\delta(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}.$$

Next, for a given schedule X , the *completion time* of the k th contract is defined as

$$G_k = \sum_{i=0}^k \delta(P_i, P_k) L_i.$$

Note that although G depends on X , we omit the subscript. Dependence on the schedule will be made implicit throughout the paper for ease of notation. For every schedule, the index ordering must correspond to the completion time ordering. Furthermore, no two contracts may complete at the exact same time. The formal statement of these two conditions is that $k < l$ is equivalent to $G_k < G_l$ for all k, l . Also, a schedule must have $I_k = k$ for $0 \leq k \leq n-1$. This ensures that after time $t = G_{n-1}$, a solution is available for each problem instance.

To compare schedules, we use a worst-case metric called *acceleration ratio*. The acceleration ratio tells us how much faster our constructed algorithm would need to run in order to ensure the same quality as if the query time and problem were known, and a dedicated processor was assigned to producing a result. Intuitively, it measures how well a schedule handles the uncertainty about the problem instance and interruption time.

Before formally defining acceleration ratio, we must state some more technical details of the problem and present some more definitions. First, we take the view that when a contract completes at time t , its solution is available to be returned upon interruption at any time $\tau > t$. Second, we assume that

no interruptions occur until after time $t = G_{n-1}$, so that there is always at least one result available.

The length of the longest contract for instance i to complete before some time $t > G_{n-1}$ is

$$s(i, t) = \max\{L_k | G_k < t \text{ and } I_k = i\}.$$

The performance profile for the multi-interruptible algorithm $B(A, X)$ is defined for all i and all $t > G_{n-1}$ as

$$Q_{B(A, X)}(t, i) = Q_A(s(i, t), i).$$

We can now give a precise definition of acceleration ratio.

Definition 1 The acceleration ratio, $R_{m,n}(X)$, is the smallest constant r for which $Q_{B(A, X)}(t, i) \geq Q_A(t/r, i)$ for all contract algorithms A , all problem instances i , and all times $t > G_{n-1}$.

In the next section, we provide a schedule that is optimal within a restricted, though still interesting, class of schedules. We state below in precise terms the properties that delineate the class of schedules under consideration. Schedules having the three properties below will be called *cyclic* schedules.

The first property states that problem instances are completed in a round-robin manner. This seems sensible, as the desired problem instance is unknown. However, we cannot yet prove that for every non-problem-round-robin schedule, there is an equally good problem-round-robin schedule.

Property 1 (Problem-round-robin) $I_k = k \bmod n$ for all k .

The next property states that the lengths of contracts for each problem instance must increase with time. Given that performance profiles are nondecreasing, it seems that it would never be beneficial to use a schedule that doesn't satisfy this property. However, as with the first property, we cannot yet prove this. One difficulty lies in having to satisfy the problem-round-robin property mentioned above. We would like to be able to "remove" useless contracts from a schedule, but we have not found a way to do this while guaranteeing that the resulting schedule will be problem-round-robin.

Property 2 (Length-increasing) For all k, l , if $I_k = I_l$ and $k < l$, then $L_k < L_l$.

The final property states that processors return results in a round-robin manner. This property does not play a part in our lower bound derivation, but it is used in drawing a connection to the robot search problem. We introduce it at this point only for ease of exposition.

Property 3 (Processor-round-robin) $P_k = k \bmod m$ for all k .

We can prove a lemma that allows us to cast acceleration ratio in simpler terms when we are considering only cyclic schedules. In the proof, the following facts are established: acceleration ratio can be stated without reference to performance profiles; the only interruption times that need to be considered are completion times; and upon interruption, the result returned is from the contract with index exactly n less than index of the current contract.

Lemma 1 For all cyclic schedules X ,

$$R_{m,n}(X) = \sup_k \frac{G_{k+n}}{L_k}.$$

Proof: We first argue that

$$R_{m,n}(X) = \max_i \sup_{t > G_{n-1}} \frac{t}{s(i, t)}.$$

From the definitions given above, we have

$$Q_A(s(i, t), i) = Q_{B(A, X)}(t, i) \geq Q_A(t/R_{m,n}(X), i),$$

for all i and $t > G_{n-1}$. Since this holds for any algorithm A , we can suppose an algorithm A with performance profile $Q_A(t, i) = t$ for all i . Thus $s(i, t) \geq t/R_{m,n}(X)$, and hence $R_{m,n}(X) \geq t/s(i, t)$ for all i and $t > G_{n-1}$. This implies that

$$R_{m,n}(X) \geq \max_i \sup_{t > G_{n-1}} \frac{t}{s(i, t)}.$$

To show that equality holds, assume the contrary and derive a contradiction with the fact that $R_{m,n}(X)$ is defined as the smallest constant enforcing the inequality between $Q_{B(A, X)}$ and Q_A .

Next we show that

$$\max_i \sup_{t > G_{n-1}} \frac{t}{s(i, t)} = \max_i \sup_{k \geq n} \frac{G_k}{s(i, G_k)}.$$

For each i , $s(i, t)$ is left-continuous everywhere and piecewise constant, with the pieces delimited by time points G_k . So for all i , $t/s(i, t)$ is left-continuous and piecewise linear and increasing. Thus, the local maxima of $t/s(i, t)$ occur at the points G_k ; no other times may play a role in the supremum.

Finally,

$$\max_i \sup_{k \geq n} \frac{G_k}{s(i, G_k)} = \sup_k \frac{G_{k+n}}{L_k}$$

follows from the problem-round-robin and length-increasing properties, and the fact that no two contracts can finish at the exact same time. $\square$

To conclude this section, we define the minimal acceleration ratio for m processors and n problems to be

$$R_{m,n}^* = \inf_X R_{m,n}(X),$$

where the infimum is taken over the set of cyclic schedules. In the following sections, we provide tight bounds for this ratio.

2.2 An Exponential Schedule

A simple approach to scheduling contract algorithms is to have the contract lengths increase exponentially. We consider the schedule

$$E = \langle k \bmod m, k \bmod n, ((m+n)/n)^{k/m} \rangle.$$

It is easily verified that this is a cyclic schedule. The following theorem gives an expression for this schedule's acceleration ratio.

Theorem 1 The acceleration ratio for the exponential schedule is

$$R_{m,n}(E) = \left(\frac{n}{m}\right) \left(\frac{m+n}{n}\right)^{\frac{m+n}{m}}.$$

Proof: Let $b = ((m+n)/n)^{1/m}$. The following is true for all k :

$$\begin{aligned} \frac{G_{k+n}}{L_k} &= \frac{\sum_{i=0}^{k+n} \delta(P_i, P_k) L_i}{L_k} \\ &= \frac{\sum_{i=0}^{\lfloor (k+n)/m \rfloor} L_{(k+n) \bmod m + mi}}{L_k} \\ &= \frac{\sum_{i=0}^{\lfloor (k+n)/m \rfloor} b^{(k+n) \bmod m + mi}}{b^k} \\ &= \frac{b^{(k+n) \bmod m} \sum_{i=0}^{\lfloor (k+n)/m \rfloor} b^{mi}}{b^k} \\ &= \frac{b^{(k+n) \bmod m} \left(\frac{n}{m}\right) (b^{m(\lfloor (k+n)/m \rfloor + 1)} - 1)}{b^k} \\ &= \left(\frac{n}{m}\right) \frac{b^{k+m+n} - b^{(k+n) \bmod m}}{b^k} \\ &= \left(\frac{n}{m}\right) b^{m+n} - \left(\frac{n}{m}\right) b^{(k+n) \bmod m - k}. \end{aligned}$$

Note that this expression is nondecreasing with k . Thus

$$\begin{aligned} R_{m,n}(E) &= \lim_{k \rightarrow \infty} \left(\frac{n}{m}\right) b^{m+n} - \left(\frac{n}{m}\right) b^{(k+n) \bmod m - k} \\ &= \left(\frac{n}{m}\right) b^{m+n} \\ &= \left(\frac{n}{m}\right) \left(\frac{m+n}{n}\right)^{\frac{m+n}{m}}. \end{aligned}$$

□

There are a few things to note about the ratio we just derived. As the number of processors approaches infinity, it tends to one. This is intuitive; by adding processors, we can get arbitrarily close to the omniscient algorithm. As the number of problems approaches infinity, the ratio tends to infinity. Finally, the ratio depends only on the ratio of problems to processors, and not on the absolute numbers.

We turn now to showing that no cyclic schedule can achieve a smaller ratio.

2.3 Lower Bound

We define a function to represent the sum of the lengths of all the contracts finishing no later than contract k finishes:

$$H_k = \sum_{i=0}^k L_i.$$

We can derive an inequality involving only the acceleration ratio and $\{H_k\}$.

Lemma 2 For all cyclic schedules X and all k ,

$$H_{k+m+n} \leq R_{m,n}(X)(H_{k+m} - H_k).$$

Proof: Consider the contract with index $k+m+n$. We define the set U such that $u \in U$ if and only if u is the index of the last contract on some processor to finish no later than contract $k+m+n$ finishes. It follows that $H_{k+m+n} = \sum_{u \in U} G_u$. Note that U contains at most m distinct integers, each between 1 and $k+m+n$. Since G is increasing,

$$\sum_{u \in U} G_u \leq \sum_{i=1}^m G_{k+i+n}.$$

Using Lemma 1, we get

$$\begin{aligned} \sum_{i=1}^m G_{k+i+n} &\leq R_{m,n}(X) \sum_{i=1}^m L_{k+i} \\ &= R_{m,n}(X) (H_{k+m} - H_k). \end{aligned}$$

□

To derive our lower bound, we also need the following lemma, which is a generalization of the monotone convergence principle. Its proof is deferred to the appendix.

Lemma 3 Let m and n be relatively prime, and let $\{p_k\}$ be a sequence of real numbers that is bounded from below by 1. If $(p_k \cdots p_{k+m-1})^{n/m} \geq p_{k+m} \cdots p_{k+m+n-1}$ is satisfied for all k , then $\{p_k\}$ converges.

We can now prove our lower bound theorem.

Theorem 2 The optimal acceleration ratio for m processors and n problems is

$$R_{m,n}^* = \left(\frac{n}{m}\right) \left(\frac{m+n}{n}\right)^{\frac{m+n}{m}}.$$

Proof: Consider an arbitrary cyclic schedule X . From Lemma 2, we have

$$R_{m,n}(X)(H_{k+m} - H_k) \geq H_{k+m+n},$$

and thus

$$R_{m,n}(X) \left(1 - \frac{H_k}{H_{k+m}}\right) \geq \frac{H_{k+m+n}}{H_{k+m}}$$

for all k .

Now let $v = \gcd(m, n)$. We define $m' = m/v$ and $n' = n/v$. Note that m' and n' are relatively prime. Further, let $p_k = H_{v(k+1)}/H_{vk}$. Note that $p_k > 1$ for all k . Then for all k ,

$$\begin{aligned} R_{m,n}(X) &\left(1 - \frac{1}{p_k \cdots p_{k+m'-1}}\right) \\ &= R_{m,n}(X) \left(1 - \frac{H_{vk} \cdots H_{v(k+m'-1)}}{H_{v(k+1)} \cdots H_{v(k+m')}}\right) \\ &= R_{m,n}(X) \left(1 - \frac{H_{vk}}{H_{v(k+m')}}\right) \\ &\geq \frac{H_{vk+m+n}}{H_{vk+m}} \\ &= \frac{H_{v(k+m'+1)} \cdots H_{v(k+m'+n')}}{H_{v(k+m')} \cdots H_{v(k+m'+n'-1)}} \\ &= p_{k+m'} \cdots p_{k+m'+n'-1}. \end{aligned}$$

This implies that for all k ,

$$R_{m,n}(X) \geq \frac{p_k \cdots p_{k+m'+n'-1}}{p_k \cdots p_{k+m'-1} - 1}.$$

There are two cases to consider.

Case 1: There exists some k' such that

$$(p_{k'} \cdots p_{k'+m'-1})^{\frac{n}{m}} \leq p_{k'+m'} \cdots p_{k'+m'+n'-1}.$$

Then we have

$$R_{m,n}(X) \geq \frac{(p_{k'} \cdots p_{k'+m'-1})^{\frac{m+n}{m}}}{p_{k'} \cdots p_{k'+m'-1} - 1}.$$

We are interested in how small $R_{m,n}(X)$ can be. Let $d = p_{k'} \cdots p_{k'+m'-1}$. Then

$$R_{m,n}(X) \geq \frac{d^{\frac{m+n}{m}}}{d - 1}.$$

The value $d = (m+n)/n$ minimizes the right-hand side over the region $d > 1$. Substituting into the previous inequality, we find

$$R_{m,n}(X) \geq \frac{\left(\frac{m+n}{n}\right)^{\frac{m+n}{m}}}{\left(\frac{m+n}{n}\right) - 1} = \left(\frac{n}{m}\right) \left(\frac{m+n}{n}\right)^{\frac{m+n}{m}}.$$

Case 2: The inequality

$$(p_k \cdots p_{k+m'-1})^{\frac{n}{m}} > p_{k+m'} \cdots p_{k+m'+n'-1}$$

holds for all k .

Because $n/m = n'/m'$, we can apply Lemma 3 to show that $\{p_k\}$ converges, and hence $\lim_{k \rightarrow \infty} p_k$ is well defined. Then we have

$$\begin{aligned} R_{m,n}(X) &\geq \lim_{k \rightarrow \infty} \frac{p_k \cdots p_{k+m'+n'-1}}{p_k \cdots p_{k+m'-1} - 1} \\ &= \frac{\lim_{k \rightarrow \infty} p_k \cdots p_{k+m'+n'-1}}{\lim_{k \rightarrow \infty} p_k \cdots p_{k+m'-1} - 1} \\ &= \frac{(\lim_{k \rightarrow \infty} p_k)^{m'+n'}}{(\lim_{k \rightarrow \infty} p_k)^{m'} - 1}. \end{aligned}$$

We can proceed as in the previous case but with $d = \lim_{k \rightarrow \infty} p_k$. This gives

$$R_{m,n}(X) \geq \frac{d^{m'+n'}}{d^{m'} - 1}.$$

As in the previous case, we find that

$$R_{m,n}(X) \geq \left(\frac{n}{m}\right) \left(\frac{m+n}{n}\right)^{\frac{m+n}{m}}.$$

Combining this inequality with Theorem 1, we get the desired result. $\square$

3 Multi-Robot Search on Rays

The results from the previous section can be directly applied to a formally similar problem involving multiple robots searching for a goal. The problem is described as follows. Initially, m robots stand at the intersection of p rays (with $m < p$). The robots, all moving in a continuous fashion and

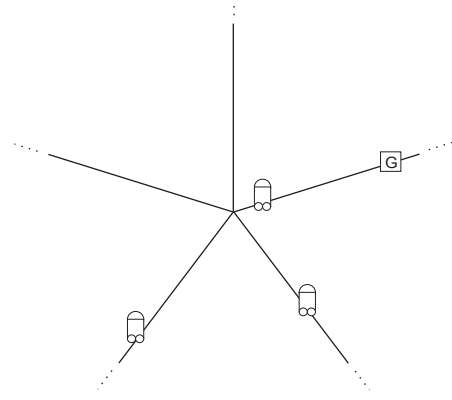


Figure 2: The search problem with three robots and five rays.

at the same speed, search for a goal at an unknown location on one of the rays (see Figure 2). The search ends as soon as one of the robots finds the goal. Because there are not enough robots to cover all of the rays, finding a good search strategy is nontrivial. At this level of detail, the search problem may seem very different from the contract scheduling problem. Below we provide a more precise description of the search problem and show how the contract scheduling results can be applied.

A search is an infinite sequence of search extents, or return trips departing from the origin. Formally, a *search strategy* is a triple $X = \langle \{P_k\}, \{I_k\}, \{L_k\} \rangle$, where $P_k \in \{0, \dots, m-1\}$ is the robot executing search extent k , $I_k \in \{0, \dots, p-1\}$ is the ray on which search extent k takes place, and $L_k \in \mathbb{R}^+$ is the length of search extent k . By adding the same conditions as in the contract scheduling case, we get an exact correspondence between schedules and search strategies. A *cyclic search strategy* is defined in the same way as a cyclic schedule.

A natural metric for the efficiency of a search strategy is the *time-competitive ratio*, the worst-case ratio of the total time spent searching to the time required for a single robot to traverse directly to the goal. We assume that the goal is not considered discovered until a robot actually moves beyond it. We assume also that the location of the goal is such that it cannot be found on the first search extent on any of the rays. This means that for each ray a , we consider only locations $t \geq L_a$.

Before formally defining time-competitive ratio, we need to introduce a new function and explain its use in the definition. Let us define $s(a, t)$ to be the index of the first search extent that goes past point t on ray a . This means that if the goal is located at point t on ray a , then it is found during search extent $s(a, t)$. The total search time up through extent $s(a, t)$ is $2G_{s(a,t)}$. (The factor of two results from search extents going out and back on rays.) However, since the search ends as soon as the goal is found, the last extent does not go to completion, and thus we must subtract out $2L_{s(a,t)} - t$. This leads us to the following formal definition for time-competitive ratio.

Definition 2 The time-competitive ratio,

$$T_{m,p}(X) = \max_a \sup_{t \geq L_a} \frac{2G_{s(a,t)} - (2L_{s(a,t)} - t)}{t}.$$

We can now give a formula for the relationship between acceleration ratio and time-competitive ratio for cyclic schedules and search strategies.

Theorem 3 For all cyclic search strategies (schedules) X ,

$$T_{m,p}(X) = 2R_{m,p-m}(X') + 1,$$

where X' is the same as X , but with $I_k = k \bmod (p - m)$.

Proof: Because of the length-increasing and problem-round-robin properties, we know that the first search extent to pass a point on a ray comes p extents after the last extent on that same ray. Formally, we have that for all a and all $t \geq L_a$,

$$L_{s(a,t)-p} \leq t < L_{s(a,t)}.$$

Thus we have

$$\begin{aligned} T_{m,p}(X) &= \max_a \sup_{t \geq L_a} \frac{2G_{s(a,t)} - (2L_{s(a,t)} - t)}{t} \\ &= \max_a \sup_{t \geq L_a} \frac{2G_{s(a,t)} - 2L_{s(a,t)}}{t} + 1 \\ &= \max_a \sup_{t \geq L_a} \frac{2G_{s(a,t)} - 2L_{s(a,t)}}{L_{s(a,t)-p}} + 1. \end{aligned}$$

The last equation above expresses the intuitive notion that the worst place for the goal to be is just out of reach of one of the robots.

Now we have

$$\begin{aligned} &\max_a \sup_{t \geq L_a} \frac{2G_{s(a,t)} - 2L_{s(a,t)}}{L_{s(a,t)-p}} + 1 \\ &= \sup_k \frac{2G_{k+p} - 2L_{k+p}}{L_k} + 1 \\ &= \sup_k \frac{2(G_{k+p} - L_{k+p})}{L_k} + 1 \\ &= \sup_k \frac{2G_{k+p-m}}{L_k} + 1. \end{aligned}$$

The last equation requires some explanation. Because of the processor-round-robin property, the robot that performs search extent $k + p$ does so immediately after completing search extent $k + p - m$. Thus $G_{k+p} = G_{k+p-m} + L_{k+p}$.

Finally,

$$\begin{aligned} \sup_k \frac{2G_{k+p-m}}{L_k} + 1 &= 2 \sup_k \frac{G_{k+p-m}}{L_k} + 1 \\ &= 2R_{m,p-m}(X') + 1. \end{aligned}$$

□

We define the minimal time-competitive ratio for m robots and p rays as

$$T_{m,p}^* = \inf_X T_{m,p}(X),$$

where the infimum is taken over the set of cyclic search strategies. This has the following immediate consequence.

Corollary 1 $T_{m,p}^* = 2R_{m,p-m}^* + 1$.

4 Conclusion

In this paper, we addressed two apparently different scheduling problems, one involving contract algorithms, and the other involving robots searching on rays. For the contract scheduling problem, we provided a schedule and proved its optimality among the class of cyclic schedules. We further showed how contract scheduling results can be applied to the robot search problem, thus unifying the two problems.

A natural direction for future research is to study less restricted classes of schedules and search strategies. One intriguing question is whether lower acceleration ratios can be achieved with schedules that are not problem-round-robin or length-increasing. It would also be interesting to know whether the contract scheduling and robot search problems have similarities beyond those that result from using cyclic schedules and search strategies.

Acknowledgments

We thank William Hesse and László Babai for providing key insights into the proof of Lemma 3. We also thank Theodore Perkins and Charles Sutton for comments on earlier drafts. Support for this work was provided in part by the NSF under grant IIS-0219606 and by NASA under grants NAG-2-1394 and NAG-2-1463. Daniel Bernstein was supported by a NASA GSRP Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the NSF or NASA.

A Proof of Lemma 3

This section contains a proof of the central lemma used for our lower bound result. It is repeated below.

Lemma 3 Let m and n be relatively prime, and let $\{p_k\}$ be a sequence of real numbers that is bounded from below by 1. If $(p_k \cdots p_{k+m-1})^{n/m} \geq p_{k+m} \cdots p_{k+m+n-1}$ is satisfied for all k , then $\{p_k\}$ converges.

Proof: We first define $q_k = \ln p_k$. Since the convergence of $\{q_k\}$ implies the convergence of $\{p_k\}$, we can hereafter focus our attention on $\{q_k\}$.

We have

$$nq_k + \cdots + nq_{k+m-1} \geq mq_{k+m} + \cdots + mq_{k+m+n-1}.$$

Let us define

$$r_k = nq_k + \cdots + nq_{k+m-1} - mq_{k+m} - \cdots - mq_{k+m+n-1}.$$

Note that $r_k \geq 0$ for all k . Now let

$$\begin{aligned} s_k &= nq_k + 2nq_{k+1} + \cdots + mnq_{k+m-1} + \cdots + \\ &\quad 2mq_{k+m+n-3} + mq_{k+m+n-2}. \end{aligned}$$

A simple calculation shows that for all k , $r_k = s_k - s_{k+1}$. Because the r_k are all nonnegative, we have $s_k \geq s_{k+1}$ for all k . Since $\{p_k\}$ is bounded below by 1, $\{q_k\}$ is bounded from below, and hence $\{s_k\}$ is bounded below. By the monotone convergence principle, $\{s_k\}$ converges.

Thus we have

$$\begin{aligned} v + \epsilon_k &= nq_k + 2nq_{k+1} + \cdots + mnq_{k+m-1} + \cdots + \\ &\quad 2mq_{k+m+n-3} + mq_{k+m+n-2}, \end{aligned}$$

where $v = \lim_{k \rightarrow \infty} s_k$ and $\lim_{k \rightarrow \infty} \epsilon_k = 0$. This equation is a linear nonhomogeneous difference equation [Mickens, 1987]. The behavior of $\{q_k\}$ depends on the roots of the characteristic polynomial,

$$f(x) = nx^{m+n-2} + 2nx^{m+n-3} + \dots + mn x^{n-1} + \dots + 2mx + m.$$

We know that $\{q_k\}$ is bounded below, and since $\{s_k\}$ converges, $\{q_k\}$ is also bounded above. Thus to demonstrate convergence, it suffices to show that f has no roots on the unit circle.

Consider the polynomial

$$g(x) = (x - 1)^2 f(x) = nx^{m+n} - (n + m)x^n + m.$$

Clearly $z = 1$ is a root of g but not of f . To show that f has no roots on the unit circle, we will show that g has no roots on the unit circle other than $z = 1$.

We must show that for all z , if $g(z) = 0$ and $|z| = 1$, then $z = 1$. If $g(z) = 0$, then

$$nz^{m+n} + m = (n + m)z^n.$$

Adding the assumption that $|z| = 1$, we get

$$n + m = |n + m||z|^n = |(n + m)z^n| = |nz^{m+n} + m|.$$

In order for the first and last expressions to be equal, we must have $z^{m+n} = 1$. Substituting back into the original equality, we get

$$n + m = (n + m)z^n,$$

which implies that $z^n = 1$. It is a basic property of complex numbers that if $z^{m+n} = z^n = 1$ and $\gcd(m, n) = 1$, then $z = 1$.

We have thus shown that $\{q_k\}$ converges, and hence $\{p_k\}$ converges. $\square$

References

- [Baeza-Yates *et al.*, 1993] Ricardo Baeza-Yates, Joseph Culberson, and Gregory Rawlins. Searching in the plane. *Information and Computation*, 106:234–252, 1993.
- [Bernstein *et al.*, 2002] Daniel S. Bernstein, Theodore J. Perkins, Shlomo Zilberstein, and Lev Finkelstein. Scheduling contract algorithms on multiple processors. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 2002.
- [Dean and Boddy, 1988] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988.
- [Horvitz, 1987] Eric Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Workshop on Uncertainty in Artificial Intelligence*, 1987.
- [Horvitz, 2001] Eric Horvitz. Principles and applications of continual computation. *Artificial Intelligence Journal*, 126(1-2):159–196, 2001.
- [Kao *et al.*, 1998] Ming-Yang Kao, Yuan Ma, Michael Sipser, and Yiqun Yin. Optimal constructions of hybrid algorithms. *Journal of Algorithms*, 29:142–164, 1998.

[Mickens, 1987] Ronald E. Mickens. *Difference Equations*. Van Nostrand Reinhold Company, Inc., New York, NY, 1987.

[Russell and Zilberstein, 1991] Stuart J. Russell and Shlomo Zilberstein. Composing real-time systems. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, 1991.

[Zilberstein *et al.*, 1999] Shlomo Zilberstein, François Charpillat, and Philippe Chassaing. Real-time problem-solving with contract algorithms. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.

Maximizing Flexibility: A Retraction Heuristic for Oversubscribed Scheduling Problems

Laurence A. Kramer and Stephen F. Smith

The Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh PA 15213
{lkramer,sfs}@cs.cmu.edu

Abstract

In this paper we consider the solution of scheduling problems that are inherently over-subscribed. In such problems, there are always more tasks to execute within a given time frame than available resource capacity will allow, and hence decisions must be made about which tasks should be included in the schedule and which should be excluded. We adopt a controlled, iterative repair search approach, and focus on improving the results of an initial priority-driven solution generation procedure. Central to our approach is a new retraction heuristic, termed *max-flexibility*, which is responsible for identifying which tasks to (temporarily) retract from the schedule for reassignment in an effort to incorporate additional tasks into the schedule. The *max-flexibility* heuristic chooses those tasks that have maximum flexibility for assignment within their feasible windows. We empirically evaluate the performance of *max-flexibility* using problem data and the basic scheduling procedure from a fielded airlift mission scheduling application. We show that it produces better improvement results than two contention-based retraction heuristics, including a variant of min-conflicts [Minton *et al.*, 1992], with significantly less search and computational cost.

1 Introduction

Many scheduling domains present problems that are oversubscribed; problems where there are more tasks to be performed over a given time frame than can be feasibly accommodated by available resources. In such problems, it is inevitably necessary to exclude some tasks from the schedule. Hence, a basic objective is to maximize resource utilization (or somewhat equivalently to accommodate as many tasks as possible). However, in many cases, the situation is further complicated. Input tasks are often differentiated by priority, implying that some tasks are more important than others and if necessary should be included at the expense of others. In the particular domain that motivates this work, for example, task priorities must be rigidly respected. Though it is theoretically

possible to trade one higher priority task for a set of lower priority tasks in some circumstances, this is the exception rather than the rule. Thus, the objective is to accommodate as many tasks as possible within this constraint.

Oversubscribed problems present an interesting challenge for constraint-directed search procedures. Oversubscribed problems are not particularly well suited to formulation within a standard backtracking search framework. Hence the application of constructive approaches depends heavily on the ability of search control heuristics to anticipate resource interactions. Repair-based approaches, on the other hand, tend to operate myopically through infeasible intermediate states in hopes of arriving at a better final feasible state. Their effectiveness in the presence of global constraints (such as enforcement of priority) will similarly rely on the ability of search heuristics to effectively focus the repair process.

In this paper, we take a repair-based search perspective of the problem and focus on locally improving an initial solution. We assume that the initial solution generator is priority-driven, and define a controlled, task swapping search procedure for finding and exploiting opportunities to rearrange currently scheduled tasks and incorporate additional tasks that were excluded from the initial solution. To direct the repair process we introduce a novel retraction heuristic, *max-flexibility*, for choosing which tasks to (temporarily) retract to make room for additional, lower priority tasks. *Max-flexibility* chooses based on a simple measure of the relative temporal flexibility that alternative competing tasks have for feasible re-assignment elsewhere in the schedule.

We evaluate the efficacy of this approach using data and the basic scheduling procedure obtained from a real-world, multi-mission scheduling problem: the day-to-day airlift scheduling problem faced by the USAF Air Mobility Command (AMC). We compare the performance of *max-flexibility* to a variant of the min-conflicts heuristic [Minton *et al.*, 1992] (adapted to serve as a retraction heuristic) and another more-informed contention-based heuristic.

2 Basic Allocation Procedure

Without loss of generality the AMC scheduling problem can be characterized abstractly as follows:

- A set T of tasks (or missions) are submitted for execution. Each task $i \in T$ has an earliest pickup time

est_i , a latest delivery time lft_i , a pickup location $orig_i$, a dropoff location $dest_i$, a duration d_i (determined by $orig_i$ and $dest_i$) and a priority pr_i

- A set Res of resources (or air wings) are available for assignment to missions. Each resource $r \in Res$ has capacity $cap_r \geq 1$ (corresponding to the number of contracted aircraft for that wing).
- Each task i has an associated set Res_i of feasible resources (or air wings), any of which can be assigned to carry out i . Any given task i requires 1 unit of capacity (i.e., one aircraft) of the resource r that is assigned to perform it.
- Each resource r has a designated location $home_r$. For a given task i , each resource $r \in Res_i$ requires a positioning time $pos_{r,i}$ to travel from $home_r$ to $orig_i$, and a de-positioning time $depos_{r,i}$ to travel from $dest_i$ back to $home_r$.

A schedule is a *feasible* assignment of missions to wings. To be feasible, each task i must be scheduled to execute within its $[est_i, lft_i]$ interval, and for each resource r and time point t , $assigned-cap_{r,t} \leq cap_r$. Typically, the problem is over-subscribed and only a subset of tasks in T can be feasibly accommodated. If all tasks cannot be scheduled, preference is given to higher priority tasks. Tasks that cannot be placed in the schedule are designated as *unassignable*. For each unassignable task i , $pr_i \leq pr_j, \forall j \in Scheduled(T) : r_j \in Res_i \wedge [st_j, et_j] \cap [est_i, lft_i] \neq \emptyset$, where r_j is the assigned resource and $[st_j, et_j]$ is the scheduled interval.

Both the scale and continuous, dynamic nature of the AMC scheduling problem effectively preclude the use of systematic solution procedures that can guarantee any sort of maximal accommodation of the tasks in T . The approach adopted within the AMC Allocator application instead focuses on quickly obtaining a good baseline solution, and then providing a number of tools for the end user to selectively relax problem constraints to incorporate as many additional (initially unassignable) tasks as possible [Becker and Smith, 2000; Kramer and Smith, 2002].

The basic allocation procedure used within the AMC Allocator constructs a schedule incrementally; the set of unassigned input missions is first prioritized, and then missions are successively inserted into the current partial scheduling in priority order. The prioritization scheme utilized in the initial step considers the assigned priority $pr(i)$ of each mission i as its dominant ordering criterion. In case of missions of equal priority, secondary criteria give preference to missions with earlier possible start times and smaller overall slack. A given mission i is inserted into the schedule via a search of alternative options. More specifically, for each $r \in R_i$, a set of candidate execution intervals is generated and evaluated, and the highest ranked alternative is selected as the assignment. In the search procedure's basic configuration, only feasible intervals are generated and the evaluation function emphasizes choices that minimize $pos_{r,i}$, $depos_{r,i}$ and execute i as early as possible. If a mission cannot be feasibly assigned, it is marked as unassignable.

This scheduling process allows for assignment of individual missions on the order of milliseconds, with a two to

three week window of approximately a thousand missions assignable in a few seconds.

Since it is generally expected that several or more missions will be unable to be assigned during the first pass allocation process, the user can direct the system to automatically explore the space of constraint relaxation options on a given mission, or can do this on a more interactive basis. Available relaxation options include delaying a mission beyond its due date, over-allocating beyond contracted capacity on a wing, bumping a lower priority mission, or some combination of these basic options. Specifically tailored evaluation functions are used when searching in the space of relaxed constraints. For instance, when searching for delay options, the dominant optimization criterion is minimizing tardiness.¹

3 Improving on the Initial Solution

The quality of any schedule produced by the above greedy procedure will be a function in large part of its prioritization heuristic. This heuristic dictates the assignment possibilities that will be available to a given task i at the time it is inserted into the schedule, and consequently which tasks will ultimately end up as unassignable. As just indicated, this heuristic gives over-riding preference to higher priority tasks. This bias in fact reflects the basic scheduling policy in the AMC application domain.

At the same time, strict reliance on this ordering heuristic can obviously lead to sub-optimal solutions. It is quite possible that some "rearrangement" of the assignments of higher priority tasks could enable the feasible insertion of additional, lower priority tasks. Since, in practice, the end user may spend non-trivial amounts of time analyzing and negotiating constraint relaxation options to enable incorporation of additional unassignable missions, it makes sense to consider techniques for productively broadening the search performed by this basic procedure.

Two broad classes of approaches have been pursued in the literature. One set of approaches (e.g., [Joslin and Clements, 1998; Bresina, 1996; Cicirello and Smith, 2002]) focuses on exploring a "neighborhood" around the trajectory of the base heuristic. In our context, this would correspond to repeatedly perturbing the task order in some manner and reapplying the basic search procedure. Another set of approaches have been termed *iterative repair* (e.g., [Minton *et al.*, 1992; Zweben *et al.*, 1994; Rabideau *et al.*, 1999]), wherein an initial base solution is progressively revised (and hopefully improved) over time. In our context, this would correspond to repeated retraction and (re)assertion of subsets of task assignments. Both sets of approaches have natural anytime properties. One interesting requirement in the current context, however, is that we would like to guarantee that higher priority missions won't be supplanted by lower priority missions. In the case of iterative re-solving approaches, enforcement of this constraint would seem possible only in a rather

¹A separate search procedure designed to identify and exploit opportunities for reclaiming resource capacity by combining (or "merging") two or more roundtrip missions into one is also provided, but this capability is orthogonal to the techniques discussed in the current paper.

indirect way (through global filtering of the solutions generated). Iterative repair approaches typically involve generation of infeasible solutions and are thus potentially well-suited to controlled exploration of solution improvement opportunities. However, they are generally not designed to enforce global solution constraints.

Accordingly, the approach we propose takes the solution improvement perspective of iterative repair methods as a starting point, but manages solution change in a more systematic, globally constrained manner. Starting with an initial baseline solution and a set U of unassignable tasks, the basic idea is to spend some amount of iterative repair search around the “footprint” of each unassignable task’s feasible execution window in the schedule. Within the repair search for a given $u \in U$, criteria other than task priority are used to determine which task(s) to retract next, and higher priority tasks can be displaced by a lower priority task. If the repair search carried out for a given task u can find a feasible rearrangement of currently scheduled tasks that allows u to be incorporated, then this solution is accepted, and we move on to the next unconsidered task $u' \in U$. If, alternatively, the repair search for a given task u is not able to feasibly reassign all tasks displaced by the insertion of u into the schedule, then the state of the schedule prior to consideration of u is restored, and u remains unassignable. Conceptually, the approach can be seen as successively relaxing and reasserting the global constraint that higher priority missions must take precedence over lower priority missions, temporarily creating “infeasible” solutions in hopes of arriving at a better feasible solution.

In the subsections below, we describe this task swapping procedure, and the heuristics that drive it, in more detail.

3.1 Task Swapping

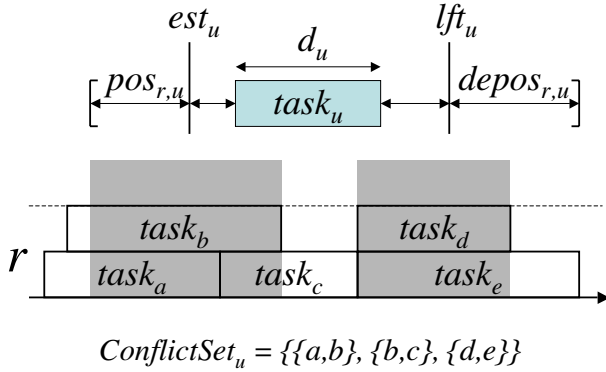


Figure 1: An unassignable task u

Figure 1 depicts a simple example of a task u that is unassignable due to prior scheduling commitments. In this case, u requires capacity on a particular resource r , and the time interval $ReqInt_{r,u} = [est_u - pos_{r,u}, lft_u + depos_{r,u}]$ defines the “footprint” of u ’s allocation requirement. Within $ReqInt_{r,u}$, an allocation duration $alloc-dur_{r,u} = pos_{r,u} + d_u + depos_{r,u}$ is required. Thus, to accommodate u , a subinterval of capacity within $ReqInt_{r,u}$ of at least $alloc-dur_{r,u}$

must be freed up.

To free up capacity for u , one or more currently scheduled tasks must be retracted. We define a conflict $Conflict_{r,int}$ on a resource r as a set of tasks of size Cap_r that simultaneously use capacity over interval int . Intuitively, this is an interval where resource r is currently booked to capacity. We define the conflict set $ConflictSet_u$ of an unassignable task u to be the set of all distinct conflicts over $ReqInt_{r,u}$ on all $r \in R_u$. In Figure 1, for example, $ConflictSet_u = \{\{a,b\}, \{b,c\}, \{d,e\}\}$.

MissionSwap(task, Protected)

1. $Protected \leftarrow Protected \cup \{task\}$
2. $ConflictSet \leftarrow \text{ComputeTaskConflicts}(task)$
3. $Retracted \leftarrow \text{RetractTasks}(ConflictSet, Protected)$
4. **if** $Retracted = \emptyset$ **then** $\text{Return}(\emptyset)$; failure
5. $\text{ScheduleTask}(task)$
6. $\text{ScheduleInPriorityOrder}(Retracted, \text{least-flexible-first})$
7. **loop for** $(i \in Retracted \wedge status_i = \text{unassigned})$ **do**
8. $Protected \leftarrow \text{MissionSwap}(i, Protected)$
9. **if** $Protected = \emptyset$ **then** $\text{Return}(\emptyset)$; failure
10. **end-loop**
11. $\text{Return}(Protected)$; success
12. **end**

RetractTasks(Conflicts, Protected)

1. $Retracted \leftarrow \emptyset$
2. **loop for** $(OpSet \in Conflicts)$ **do**
3. **if** $(OpSet - Protected) = \emptyset$ **then** $\text{Return}(\emptyset)$
4. $t \leftarrow \text{ChooseTaskToRetract}(OpSet - Protected)$
5. $\text{UnscheduleTask}(t)$
6. $Retracted \leftarrow Retracted \cup \{t\}$
7. **end-loop**
8. $\text{Return}(Retracted)$
9. **end**

Figure 2: Basic MissionSwap Search Procedure

Given these preliminaries, the basic repair search procedure for inserting an unassignable task, referred to as **MissionSwap**, is outlined in Figure 2. It proceeds by computing $ConflictSet_{task}$ (line 2), and then retracting one conflicting task for each $Conflict_{r,int} \in ConflictSet_{task}$ (line 3). This frees up capacity for inserting $task$ (line 5), and once this is done, an attempt is made to feasibly reassign each retracted task (line 6). For those retracted tasks that remain unassignable, **MissionSwap** is recursively applied (lines 7-10). As a given task is inserted by **MissionSwap**, it is marked as protected, which prevents subsequent retraction by any later calls to **MissionSwap**.

In Figure 3, top-level **InsertUnassignableTasks** procedure is shown. Once **MissionSwap** has been applied to all unassignable tasks, one last attempt is made to schedule any remaining tasks. This step attempts to capitalize on any opportunities that have emerged as a side-effect of **MissionSwap**’s schedule re-arrangement.

The driver of this repair process is the retraction heuristic **ChooseTaskToRetract**. We define several possibilities in the next subsection.

InsertUnassignableTasks(*Unassignables*)

```

1. Protected  $\leftarrow \emptyset$ 
2. loop for (task  $\in$  Unassignables) do
3.   SaveScheduleState
4.   Result  $\leftarrow$  MissionSwap(task, Protected)
5.   if Result  $\neq \emptyset$ 
6.     then Protected  $\leftarrow$  Result
7.     else RestoreScheduleState
8.   end-loop
9. loop for (i  $\in$  Unassignables  $\wedge$  statusi = unassigned) do
10.  ScheduleTask(i)
11.end-loop
12.end

```

Figure 3: InsertUnassignableTasks procedure

3.2 Retraction Heuristics

In designing a retraction heuristic, our general goal is to retract the task assignment that possesses the greatest potential for reassignment. One simple estimate of this potential is the scheduling flexibility provided by a task's feasible execution interval. More precisely, let $avail\text{-}dur_i = lft_i - eft_i$ represent the amount of time available for executing task *i*. Then a simple, resource-independent measure of flexibility is

$$\frac{d_i}{avail\text{-}dur_i}$$

However, recall that d_i is not the total amount of time that the supporting resource must be allocated for. d_i only accounts for the time required to execute the task; it does not account for the time to position and deposition the resource for task execution. The total time that a given resource *r* must be allocated for is $alloc\text{-}dur_{r,i} = pos_{r,i} + d_i + depos_{r,i}$. Given that the resources being allocated are generally scheduled near to capacity, a worst-case, resource-dependent measure of flexibility is

$$\frac{alloc\text{-}dur_{r,i}}{avail\text{-}dur_i}$$

Using this notion, we define an overall measure of *i*'s temporal flexibility as

$$Flex_i = \frac{\sum_{r \in R_i} alloc\text{-}dur_{r,i}}{avail\text{-}dur_i \times |R_i|}$$

and the following retraction heuristic:

$$MaxFlex = i \in C : Flex_i \leq Flex_j \forall j \neq i$$

where $C \in ConflictSet_u$ for some unassignable task *u*.

An alternative approach to estimating a task *i*'s potential for reassignment is to measure the level of contention across its required execution interval $ReqInt_{r,i}$. Assume as before that $ConflictSet_i$ is the set of conflicts within $ReqInt_{r,i} \forall r \in R_i$. Then, a very basic measure of contention for a given task *i* is simply the number of conflicts, i.e. $|ConflictSet_i|$. For purposes of choosing which task to retract in the case of a conflict $C \in ConflictSet_u$, this measure leads to a "min-conflicts" like heuristic[Minton *et al.*, 1992]:

$$MinConflicts = i \in C : |ConflictSet_i| \leq |ConflictSet_j|$$

$$\forall j \neq i$$

A potentially more informed measure of contention is one that considers the proportion of a task *i*'s required execution interval on resource *r* that is currently unavailable (i.e., contained in a conflict). For any conflict $C_{r,int} \in ConflictSet_i$, assume that dur_C designates the duration of *int*. Then we define a task *i*'s overall contention level as

$$Cont_i = \frac{\sum_{C \in ConflictSet_i} dur_C}{\sum_{r \in R_i} ReqInt_{r,i}}$$

Using this measure, we can define a third retraction heuristic:

$$MinContention = i \in C : Cont_i \leq Cont_j, \forall j \neq i$$

4 Computational Analysis

The genesis of the max-flexibility heuristic and the mission-swap algorithm took place while experimenting with the canonical data set used to test and demonstrate the AMC Allocator. This data set, which we'll refer to as the Tutorial Data Set, consists of 982 actual missions (3,251 operations) and 12 actual air wings, and represents a two to three week horizon of AMC airlift and air refueling missions.

Using the basic allocation procedure the system is able to feasibly allocate all but two of the 982 missions in the Tutorial Data Set. For demonstration purposes those two missions are usually added to the schedule by over-allocating a given wing.

It turned out that this particular data set was not as resource over-constrained as we had thought. Application of the mission-swap algorithm with even a random retraction heuristic is able to schedule the two unassignable missions in a few seconds.

4.1 Experimental Design

For our experiments we generated data sets using the Tutorial Data Set as a seed: five data sets of twenty problems each were generated, with the wing capacities randomly reduced from 0 to 10%, 0 to 20%, 0 to 30%, 0 to 40%, and 0 to 50%. For each data set, the basic allocation procedure was employed to quickly schedule as many missions as possible in priority order. The number of unassignable missions was recorded, and then **InsertUnassignableTasks** was executed on the set of unassignable missions. Then run-time, nodes searched (number of times **MissionSwap** was called), and final number of unassignable missions were recorded. For each run this process was repeated with min-conflicts, min-contention, max-flexibility and random choice as the retraction heuristic (**ChooseTaskToRetract**, in Figure2).

Experiments were run on a 1.8Ghz Pentium IV PC with 1Gb of RAM, running Windows 2000. The scheduling engine is implemented in Allegro Common Lisp 6.1.

4.2 Results

The results of our experiments are shown in Figures 4, 5, and 6. As expected, as the random degree of over-allocation was increased from 10% to 50%, the number of initial unassignable missions increased, as did the numbers of unassignables after application of **InsertUnassignableTasks**. Over all problem sets the baseline random-choice heuristic

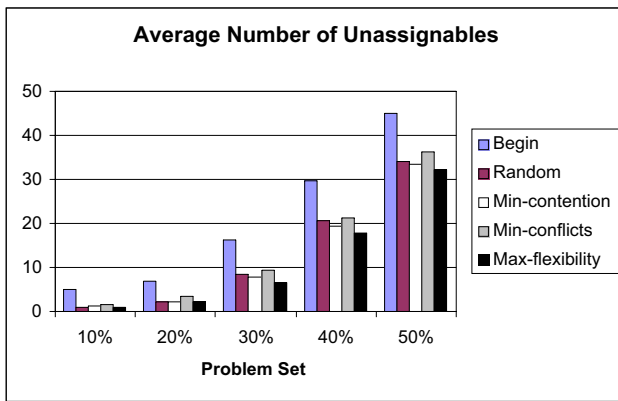


Figure 4: Solution Quality

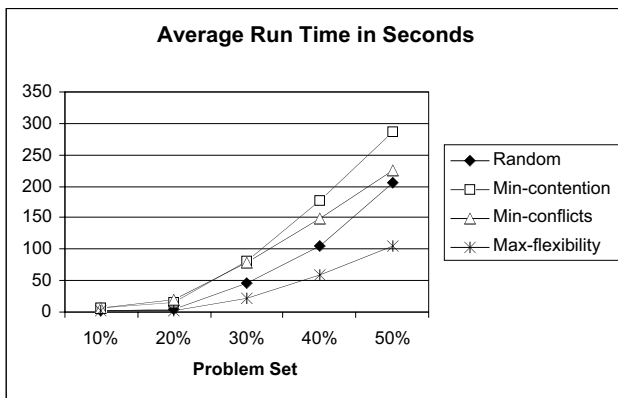


Figure 5: Computational Cost

was able to assign 36% of the unassignable missions. Min-conflicts, min-contention and max-flexibility were able to assign 30%, 38%, and 42%, respectively.

What is most striking, though, are the results shown in figures five and six. While max-flexibility achieved somewhat better results in solution quality, it did so while searching far less and reaching a solution far more quickly than any of the competing heuristics.

4.3 Discussion

Our preliminary results show that in the face of a resource constrained scheduling problem, the decision of what tasks to temporarily retract in order admit more tasks into the schedule may be delegated to an extremely simple and cheap retraction heuristic: max-flexibility. In order for this heuristic to be effective it should retract tasks that have a high likelihood of reassignment elsewhere within their feasible windows.

It is somewhat counter-intuitive, then, that max-flexibility is able to perform so well, based only on task flexibility to re-schedule, irrespective of other competing tasks in the schedule. Min-contention performance approaches that of max-flexibility, but only at significant additional cost – approximately triple the run-time on average in our experiments.

Our conjecture is that as a schedule becomes more and

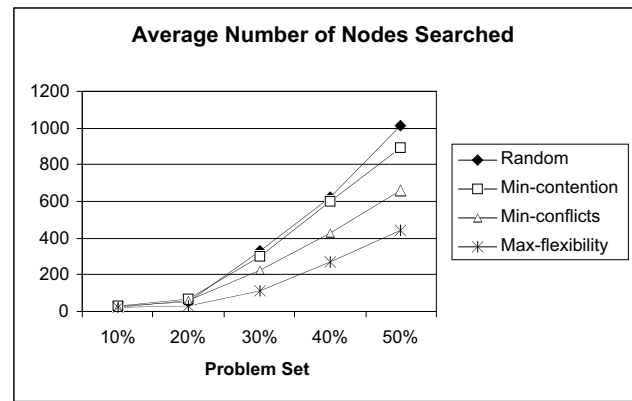


Figure 6: Search Nodes Explored

more highly constrained, contention will increase to the limit of available capacity, and thus its measure will serve as a less informed heuristic. Under these conditions, though, a metric based on a task's "innate" flexibility to schedule along a timeline might be more useful.

5 Related Work

The use of measures of temporal flexibility and resource contention as guidance for variable and value ordering has a long history in the field of constraint-directed scheduling [Sadeh, 1991; Smith and Cheng, 1993; Beck, 1999], although for the most part use has been in constructive search contexts. Our work, alternatively, seeks to exploit these measures in an iterative repair search context, to determine which tasks to retract and reassign.

Work in manufacturing scheduling domains has addressed a broad range of priority based scheduling problems [Morton and Pentico, 1993], and some of this work (e.g., [Smith, 1994]) discusses heuristic techniques for schedule repair. However, a broad assumption that underlies most of this work is that due dates are relaxable and the objective is to minimize tardiness. As such, these are not over-subscribed problems in the same sense as the type of problem considered here.

Research in the domain of space mission planning and scheduling, alternatively, has focused on the solution of over-subscribed problems (e.g., [Minton *et al.*, 1992; Johnston and Miller, 1994; Rabideau *et al.*, 1999]). Generally this research is aimed at solving the single mission, single resource problem (e.g., the observing schedule for a space telescope). [Zweben *et al.*, 1994]'s shuttle ground processing domain is one exception, wherein both multiple resources and multi-capacity resources are considered. The approach employed here is repair-based, and simple heuristics are employed to mitigate resource constraint violations. It is not clear how useful this technique would be, however, in addressing problems where maintaining task priority is crucial.

Recent work in the area of scheduling observations on multiple earth satellites comes closest to tackling the same issues which we address: multiple resources, multi-capacity resources, fixed time windows, and mission priority. [Pemberton, 2000] proposes a solution to preserving priority in the

face of over-constrained resources by segmenting the problem into priority classes, solving them individually, and recombining them. [Frank *et al.*, 2001] propose a contention-based heuristic for use in solving a very similar problem within an iterative sampling framework. However, no experimental analysis is given.

Finally, work in constraint satisfaction problem solving (CSP) has explored ideas similar to those introduced in this paper. [Verfaillie and Schiex, 1994] describe a repair procedure similar to **MissionSwap** for non-disruptively resolving conflicts in dynamic CSPs (actually motivated by previous work on a continuous, single-resource scheduling problem). [Prestwich, 2001] utilizes a “largest domain first” retraction heuristic to drive an incomplete backtracking procedure.

6 Conclusions

We have presented a novel retraction heuristic, max-flexibility, and shown its applicability to a multi-mission airlift scheduling problem. This heuristic is much faster and less memory intensive than min-conflicts and min-contention, and provides better solution quality. We suggest that max-flexibility as used in a generic task swapping algorithm like mission-swap may generally be applicable to resource-constrained scheduling problems with fixed time windows. This method is particularly suitable to dynamic scheduling domains, where an existing schedule must be preserved as tasks change and new ones are added.

It is possible that with further study and experimentation a more informed retraction heuristic than max-flexibility might be found. It is likely, though, that such a heuristic would be much more expensive, and that a heuristic that is fairly smart, but very cheap, may be the best choice of all.

Acknowledgements

The work reported in this paper was sponsored in part by the Department of Defense Advanced Research Projects Agency (DARPA) and the US Air Force Research Laboratory under contracts F30602-00-2-0503 and F30602-02-2-0149, by the USAF Air Mobility Command under subcontract 10382000 to Northrop-Grumman Corporation, and by the CMU Robotics Institute.

References

- [Beck, 1999] J.C. Beck. *Texture measurements as a Basis for Heuristic Commitment Techniques in Constraint-Directed Scheduling*. PhD thesis, Dept. of Computer Science, University of Toronto, 1999.
- [Becker and Smith, 2000] M.A. Becker and S.F. Smith. Mixed-initiative resource management: The amc barrel allocator. In *Proc. 5th Int. Conf. on AI Planning and Scheduling*, pages 32–41, Breckenridge CO, April 2000.
- [Bresina, 1996] J. Bresina. Heuristic-baised stochastic sampling. In *Proceedings 13th national Conference on AI*, pages 271–278, Portland OR, 1996. AAAI Press.
- [Cicirello and Smith, 2002] V. Cicirello and S.F. Smith. Amplification of search performance through randomization of heuristics. In *Proc. 8th Int. Conf. on Principles and Practice of Constraint Programming*, Ithaca NY, Sept 2002. Springer-Verlag.
- [Frank *et al.*, 2001] J. Frank, A. Jónsson, R. Morris, and Smith D.E. Planning and scheduling for fleets of earth observing satellites. In *Proc. 6th Int. Symposium on AI, Robotics and Automation for Space*, 2001.
- [Johnston and Miller, 1994] M.D. Johnston and G. Miller. Spike: Intelligent scheduling of hubble space telescope observations. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann Publishers, 1994.
- [Joslin and Clements, 1998] D.E. Joslin and D.P. Clements. Squeakywheel optimization. In *Proc. 15th National Conference on AI*, Madison WI, July 1998. AAAI Press.
- [Kramer and Smith, 2002] L. Kramer and S.F. Smith. Optimizing for change: Mixed-initiative resource management with the amc barrel allocator. In *Proc. 3rd Int. Workshop on Planning and Scheduling for Space*, Houston, Oct 2002.
- [Minton *et al.*, 1992] S. Minton, M.D. Johnson, A.B. Philips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1):161–205, 1992.
- [Morton and Pentico, 1993] T.E. Morton and D.W. Pentico. *Heuristic scheduling Systems*. John Wiley and Sons, 1993.
- [Pemberton, 2000] J.C. Pemberton. Toward scheduling over-constrained remote-sensing satellites. In *Proceedings 2nd Int. NASA Workshop on Planning and Scheduling for Space*, San Francisco, CA, March 2000.
- [Prestwich, 2001] S. Prestwich. Local search and backtracking vs non-systematic backtracking. In *Proc. AAAI Fall Symposium on Using Uncertainty within Computation*, North Falmouth, MA, Nov 2001.
- [Rabideau *et al.*, 1999] G. Rabideau, R. Knight, S. Chien, A. Fukunaga, and A. Govindjee. Iterative planning for spacecraft operations using the aspen system. In *Proc. 5th Int. Sym. on AI, Robotics and Automation for Space*, 1999.
- [Sadeh, 1991] N. Sadeh. *Look-Ahead techniques for Micro-Opportunistic Job Shop Scheduling*. PhD thesis, Dept. of Computer Science, Carnegie Mellon University, 1991.
- [Smith and Cheng, 1993] S.F. Smith and C. Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Proc. 11th National Conference on Artificial Intelligence*, pages 139–144, Wash DC, July 1993. AAAI Press.
- [Smith, 1994] S.F. Smith. Opis: A methodology and architecture for reactive scheduling. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.
- [Verfaillie and Schiex, 1994] G. Verfaillie and T. Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proc. 12th National Conf. on AI*, Seattle WA, Aug 1994.
- [Zweben *et al.*, 1994] M. Zweben, B. Daun, E. Davis, and M. Deale. Scheduling and rescheduling with iterative repair. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann Publishers, 1994.

Distributed Patient Scheduling in Hospitals

T. O. Paulussen*

Inf. Sys. Dept.
Univ. of Mannheim
Mannheim, Germany
paulussen@uni-mannheim.de

N. R. Jennings

Elect. & Comp. Sci. Dept.
Univ. of Southampton
Southampton, UK
nrj@ecs.soton.ac.uk

K. S. Decker

Comp. & Inf. Sci. Dept.
Univ. of Delaware
Newark, Delaware, USA
decker@cis.udel.edu

A. Heinzl*

Inf. Sys. Dept.
University of Mannheim
Mannheim, Germany
heinzl@uni-mannheim.de

Abstract

Patient scheduling in hospitals is a highly complex task. Hospitals have a distributed organisational structure; being divided into several autonomous wards and ancillary units. Moreover, the treatment process is dynamic (information about the patients' diseases often varies during treatments, causing changes in the treatment process). Current approaches are insufficient because they either focus only on the single ancillary units, and therefore do not consider the entire treatment process of the patients, or they do not account for the distribution and dynamics of the patient scheduling problem. Therefore, we propose an agent based approach in which the patients and hospital resources are modelled as autonomous agents with their own goals, reflecting the decentralised structures in hospitals. In this multi-agent system, the patient agents compete over the scarce hospital resources. Moreover to improve the overall solution, the agents then negotiate with one another. To this end, a market mechanism is described, in which each self interested agent tries to improve its own situation. In particular we focus on how the agents can calculate demand and supply prices based upon their current schedule. Further, an evaluation of first results of the proposed method is given.

1 Introduction

Patient scheduling in hospitals is concerned with the optimal assignment of patients to hospital resources. Hospitals are divided into several autonomous wards and ancillary units, which are visited by the patients for treatments and examinations during hospitalisation in accordance with their illness [Schlüchtermann, 1990]. However, the pathways (the needed treatments and examinations) and the medical priorities (the health condition of the patients) are likely to change due to new findings about the diseases of the patients during examination. Further, complications and arrivals of emergency patients, which are in urgent need for treatment, result in sched-

ule disturbances¹ [Schlüchtermann, 1990]. Due to this, patient scheduling in hospitals requires an approach which is distributed, in order to leave the authority at the responsible hospital units, and flexible, to be able to react to plan changes in an efficient manner.

We have chosen to adopt a multi-agent based approach to this problem, because such systems allow the representation of every single coordination object, i.e. the responsible entities, as single autonomous agents with their own goals [Weinhardt and Gember, 1996]. This, in turn, reflects the existing decentralised structures in hospitals [Decker and Li, 2000]. Through social abilities, the agents can interact with each other to reach their own goals. Moreover they can react with the needed flexibility to changes (as new information about the health status of a patient becomes available) and disturbances (e.g. emergencies and complications) through proactiveness and responsiveness [Jennings, 2001].

This paper advances the state of the art in two main ways. First is the design of a novel procedure, i.e. a patient centred multi-agent based coordination mechanism, where the patients are represented as autonomous agents, trying to improve their current scheduling state by negotiation with other agents. Second is the derivation of health state dependent opportunity cost functions, based upon which the agents evaluate their current schedule and compute the gains and losses through plan modifications.

The remainder of the paper is structured as follows. Section 2 describes the domain of patient-scheduling in hospitals. Section 3 details the conceptual framework of our multi-agent system, the relevant coordination objects and the coordination mechanism. The results of the proposed mechanism are also evaluated. The paper ends with conclusions and an outlook to further work in section 4.

2 The Patient Scheduling Problem

In addition to the complexity arising from the distributed structure of hospitals, patient scheduling has to be performed in the face of a high degree of uncertainty about the treatment pathways of patients within the hospital. Thus patients arrive

*We would like to thank the DFG, Deutsche Forschungsgemeinschaft for funding this work as part of SPP 1083.

¹This problem can be ameliorated by the provision of additional (exclusive) emergency resources (e.g. separate x-ray facilities). However, for economic reasons, hospitals try to minimise these extra resources.

continuously at the hospital and the necessary medical treatments are often not able to be completely determined at the beginning of the treatment process. Moreover the results of a diagnostic examination might change the (medical) priority of the patients, invoke additional activities and/or make other medical actions obsolete.

Due to this complexity, the application of traditional (operations research) methods from industrial scheduling to the patient scheduling problem is problematic [Schlüchtermann, 1990]. To be able to handle the process dynamics in a distributed environment, hospitals commonly use a very flexible approach for patient scheduling. Typically, the wards send treatment and examination requests to the ancillary units. Based upon these requests, the ancillary units order the patients from the wards. This allows the units to react very flexibly to changes with very low communication needs. If, for example, an emergency patient needs to be inserted, the next patient will simply be called from the ward later, leaving this patient available to other ancillary units.

However, because there is no inter-unit coordination, this procedure cannot resolve resource conflicts, which occur if the same patient is requested by more than one ancillary unit at the same time [Decker and Li, 2000]. Because the ancillary units only have a local view, i.e. they do not – and cannot – take the whole pathway of the treated patients into their scheduling consideration, no inter-unit process optimisation can be undertaken (i.e. the medical tasks for the patients cannot be scheduled and coordinated in an efficient manner). This causes undesired idle times as well as overtime hours for the hospital resources and extended patient stay times.

3 Agent-Based Patient Scheduling

In this section the conceptual framework of our multi-agent system is described. Then, the relevant coordination objects, i.e. the patients and hospital resources, are modelled as agents. After this, a coordination mechanism is introduced. Finally, examples are given and the first results of the implementation are presented.

3.1 Conceptual Framework

Using a multi-agent system, [Decker and Li, 2000] addressed the problem of resource conflicts in the patient scheduling domain. Specifically, by adding a coordination mechanism to handle mutually exclusive resources (i.e. the patients) to the Generalized Partial Global Planning (GPGP) approach², they achieved significant scheduling improvements compared to current patient scheduling without coordination. In their work, they took a function-oriented approach, where only the units are represented as autonomous agents. The treatment pathway of the patients is captured by the nursing units. However, the dynamics of the patient scheduling problem and the medical priorities of the patients were not considered.

For this reason, we use a patient-centred approach, in which the patients are modelled as agents, too. This allows the representation – and therefore consideration – of the hospital processes as a whole [Adam and Gorschlüter, 1999]. In

²GPGP is a domain independent, task environment centred, scheduling approach to coordination [Decker and Li, 2000].

contrast to the resource agents, who only see the patients as entities to be treated, the patient agents only see the medical actions as tasks that need to be performed. Due to these opposing forces, the patient agents ensure that the resource agents also consider the treatments of the patients outside their unit (without any explicit knowledge of them) and vice versa.

To this end, the relevant coordination objects in the hospital (i.e. the entities to be coordinated), which will be represented as agents, can be identified as patients and resources (i.e. rooms, machines and personnel). To reduce complexity, all resources needed for a specific medical action are represented by the hospital unit responsible for this action, which will be implemented as the resource agents.

If more than one patient agent now wants to use the identical resource at the same time interval, a resource conflict occurs. To solve these conflicts, a coordination mechanism is needed. For this inter-agent coordination, we decided to use a market mechanism. The rationale for this is that a market mechanism is a distributed coordination mechanism which facilitates efficient solutions with low communication needs. In particular, only prices for specific goods are communicated, keeping all other information private to the market participants [Wellman *et al.*, 2001]. Additionally, a market facilitates a dynamic environment, where the market participants take actions according to their current (dynamically changing) situation based upon private information and preferences. In this market, the coordination objects are modelled as autonomous, self-interested market participants, trying to improve their local schedule. In markets, the price mechanism leads to an efficient resource allocation because the resources are assigned to the agents that are willing to pay the highest price (assuming that the agents bid rationally, these are the agents who gain the highest utility from this resource). A pareto optimal solution will be achieved, because no agent will accept a deal, which worsens its current state. Therefore, the agents will trade resources until no agent can improve its schedule without harming another agent [Weigelt, 1994].

3.2 Coordination Objects

To implement a coordination mechanism, the above mentioned coordination objects have to be modelled. In this work the main goal for the patients is to minimise their stay time and for the resources to minimise idle times. This kind of scheduling problems represents a worth-oriented environment [Rosenschein and Zlotkin, 1994], where the degree of goal achievement can be evaluated by a utility function (rather than a state or task oriented environment, where a single goal cannot be achieved partially). The usage of continuous worth functions instead of single worth values assigned to specific goals enables the agents to relax their goals, i.e. to compromise in order to achieve a better solution [Rosenschein and Zlotkin, 1994]. For example, if two agents want the same time slots at the same resource, the agents can agree upon a solution which does not fully satisfy their own goals, but reaches a better overall solution. This situation is illustrated in the figures 2(a) and 2(b).

Because we adopt a patient-centred approach, the main focus will be on the patient agents, which will be modelled in

the first subsection. Within this part, we introduce the health state dependent cost-functions, which are needed for price articulation in the market mechanism. In the second subsection, the resource agents are described.

Patient Agents

Each patient is represented by an agent. The patient agents can only see their own schedule, containing their pathway through the hospital. The pathway of a patient agent comprises the needed treatments and examinations as well as the order constraints between those tasks. The patient agents are also equipped with a private worth function to be able to negotiate in a goal driven manner. This worth function is realised as a cost function, which they try to minimise. These cost functions increase over time, setting incentives for the patient agents to schedule their treatments and examinations as early as possible in order to reduce costs.

In contrast to commercial domains, the utility or cost functions cannot be based upon monetary values in hospitals. While in e-commerce scenarios the human principals reveal their preferences through their willingness to pay (e.g. by specifying maximum buy and minimum sell prices), patients do not – and should not – reveal their preferences through their willingness to pay for a specific treatment time slot. The preferences rather have to be based upon medical priorities, i.e. their health state. This absence of normalised monetary units additionally causes the problem of inter-agent utility comparison. To solve this problem, we operationalised the progress of the patients' health state. In current hospital practice, numerous health states or patient priority measures and indices are in use (e.g. APACHE II (Acute Physiology And Chronic Health Evaluation) [Knaus *et al.*, 1985] in intensive care units). The choice of the measurement to use is up to the hospital. However, we propose, that it should support a cardinal measurement of the health state progress over time.

The *cardinal measurement* is necessary for inter-agent utility comparison and transfer (as we have to calculate with these units). Because these health state (utility) units cannot be transferred between the actual patients in the hospital, the multi-agent system has to be separated from the actual patient world, where only the agents transfer utility units in order to reach a better overall solution (minimisation of health state adjusted patient stay time).

The *health state progress* is important, because the priority of the patients should not be based upon their current state, but on their health state development. For example, a patient with a currently reasonable but rapidly deteriorating health condition should have a higher priority than a patient with a (slightly) lower current, but continuously constant, health state.

For the necessary cardinal measurement of health, we rely on the concept of *years of well being* [Torrance, 1987] because it handles the health state progress over time (a good overview can be found in [Pedroni and Zweifel, 1990]). In this method, the question is what time period xT of total health (1) equals one specific time period $1T$ of a certain health state H , i.e.

$$1T * H = xT * 1 \Leftrightarrow H = x.$$

Through this, the health state of a patient can be described in time units.

The primary goal of patients in hospitals is to increase their current health state through treatment, where a disease could be viewed as disutility (decrease in quality of life) for the patient. This loss of utility adds up as long as this disease is not cured. Based on this assumption, the (opportunity) cost C^{opp} for not curing the patient right away equals the difference between the achievable health state (through treatment) z and the patient's health state over the time $H(t)$. Formally, this can be expressed by

$$C^{opp}(t) = \int_0^t z - H(t) dt.$$

In addition, it has to be considered that the health state and the achievable health state can change over time. Therefore, the patient's opportunity costs are influenced by his current health state a , the development of his health state over time $H(t)$, and the maximal reachable health state through treatment z . If the health state does not change over time, i.e. $H(t) = a$, the opportunity costs are

$$C^{opp}(t) = \mu t; \mu = z - a.$$

If the health state of a patient worsens over time, assumptions about the course of the health state have to be made by a physician. If we assume – for clarity – a linear reduction by b of the health state, i.e. $H(t) = a - bt$, we get

$$C^{opp}(t) = zt - at + \frac{b}{2}t^2 = \mu t + \frac{b}{2}t^2.$$

However, this approach works with any health decrease rate, as the health state of a patient normally does not decrease linearly. Nevertheless, a linear approximation of the decrease in health can be justified by practical reasons. Instead of trying to estimate the exact shape of the curve, a physician could rather specify two or more specific points in time and in between a linear reduction could be estimated. Figure 1 shows an exemplary course of an illness with linear reduction of the health state, resulting in a quadratic opportunity cost curve.

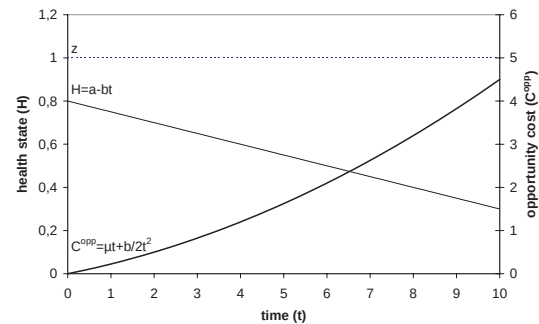


Figure 1: Linear reduction of health state.

Finally, if the maximal reachable health state decreases, the patient must be treated immediately to prevent lifelong damage to his health.

With these cost functions in place, the agents can now evaluate their current state, i.e. the degree of goal achievement. Further, based on these functions, the agents are able to compute supply and demand prices for time slots which correspond to the losses or benefits caused by plan changes, which is essential for a market mechanism. How these prices are computed will be explained later in the context of the description of the coordination mechanism (section 3.3).

Resource Agents

In contrast to the patients, the hospital resources are directly comparable with the resources in industrial scheduling domains. Their main goal can be described as maximising their utilisation or, equivalently, minimising idle time.

To reach this goal, cost-functions can also be articulated for the resource agents. These cost functions represent the reserve prices (i.e. the price that will be charged for an empty slot) for the possible appointments [Wellman *et al.*, 2001]. Through this, priorities between different resources can be established that allow penalisation of undesired appointment times (e.g. evening shift or overtime hours). Here, the basis for the resource agents' cost functions comes from cost accounting.

However, for inter-agent utility comparison (i.e. between patient and resource agents) these measurements have to be equalised, which can be a very difficult task, because the health state of a patient has to be compared with monetary values from cost accounting. A good way out of this problem are *trade-off* considerations, e.g. what amount of idle time of a specific resource equals one hour waiting time for a patient with a specific health state.

3.3 The Coordination Mechanism

The main goal of the coordination mechanism is to minimise the health state adjusted stay time of the patients, which is equivalent to an overall minimisation of suffering for the patients. The basic idea of our coordination mechanism is that the patient agents try to buy into resource time slots for the needed treatments and examinations. However, the usage of (central) auction mechanisms is obstructed by the dynamics of the patient scheduling problem described in section 2.

To ensure feasible (i.e. conflict free) initial task appointments for the patients, all new treatments and examinations are scheduled on a *first-come first-served* (fcfs) basis. To do this, the patient agents who want to add a task to their pathway contact the responsible unit for the execution of this task in order to obtain the earliest time slot which is available at the unit as well as in their own schedule. To illustrate this, figure 2(a) shows a possible initial resource allocation. These initial appointments determine the budget (or better the initial opportunity costs) of the agents. This is important because a hypothetical price system is used (as per section 3.2).

Based upon this initial schedule, the agents try to improve their schedule in order to reduce their opportunity costs. The price p they are willing to pay for a specific time slot (expected gain) or they charge for a time slot (expected loss) is the difference between the cost-value of the current allocation t^{old} and the cost-value for the wanted appointment t^{new} , according to their individual cost function described in the

previous section, i.e.

$$p(t^{old}, t^{new}) = C^{opp}(t^{old}) - C^{opp}(t^{new}).$$

Because in this approach the (opportunity) costs of an appointment increase over time for the patient agents, they must try to schedule their treatments and examinations as early as possible. If a demanded time slot is already occupied by another patient agent, the initial demander must try to buy the time slot from the current owner. With respect to the properties of a market mechanism, the agents act in a rational, self-interested manner. Therefore, the owners of the time slots will only release them, if the price offered equals the losses invoked through rescheduling. Since they only charge for the costs invoked through rescheduling they can be viewed as acting in a partially cooperative manner.

The detailed negotiation process goes as follows:

1. A patient agent initiates a negotiation for rescheduling, if the pathway (additional or obsolete medical actions) or the health state of its patient has changed.
2. The initiating agent selects the task with the highest possible improvement (difference between the costs of the current owned and the best reachable time slot) and contacts the resource agent that is responsible for the execution of this task.
3. The resource agent reserves that time slot, and contacts all affected patient agents, i.e. the agents that currently own this interval, and informs them about the proposal of the initiator.
4. The affected patient agents (sellers) try to reschedule to the first nonreserved time slots (see step 3) and notify the initiator about their costs due to rescheduling. To prevent cycles, reserved intervals cannot be demanded by other agents.
5. If the alternative time slots for the sellers are already occupied, they again become demanders for those time slots and accumulate the invoked costs. Here, order constraints can invoke additional rescheduling in other resources.
6. After all prices are computed and submitted to the initiator, the initiator compares its expected gains from rescheduling to the total price asked for this interval. If the gains exceed the costs it accepts or rejects otherwise, and the negotiation for this time slot terminates.

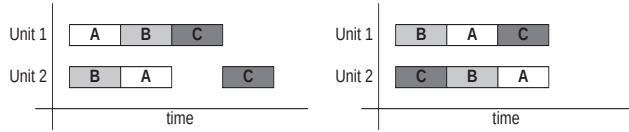
The former initiator continues its rescheduling activities by opening new negotiations for the next task with the (now) highest possible improvement until it cannot improve any task any further. Previously rejected time slots will not be considered unless these time slots are released by their owners. For concurrency issues only one (randomly chosen) agent can initiate a negotiation at a time.

In this coordination mechanism only the patient agents are active components. However, as described earlier, cost functions can be implemented into the resource agents, allowing prioritisation between the resources. The resource price for a time slot is charged when a patient agent buys a time slot, and reimbursed when the patient agent releases this time slot.

As described earlier, the pathway of the patients through the hospital is likely to change during the treatment process, that is, additional treatments and/or examinations may become necessary while other tasks may become obsolete. In our coordination mechanism, additional tasks (as well as additional patients) can be added at any time using the fcfs rule explained above. If a treatment becomes obsolete, the responsible patient agent notifies the affected resource agent that, again, informs the other patient agents in its schedule.

3.4 Example Scenarios

Figure 2(a) and 2(b) illustrate the functioning of our coordination mechanism. Figure 2(a) shows an initial schedule with three patients (A,B,C). For illustration purposes, we start with identical cost-functions with an assumed initial health state a of 0.7, an achievable health state z of 1.0, a decrease rate b of 0.001 and equal possible starting times for all patients. The used cost function resolves as $0.3t + 0.0005t^2$. Further, in this example, all task durations are set to 10 t . We will use the \$-sign to indicate utility units.



(a) Schedule before negotiation. (b) Schedule after negotiation.

Figure 2: Example Schedule before and after negotiation.

Based upon this initial schedule, agent C negotiates with agent B in unit 2 in order to improve its current situation. Agent C's current opportunity costs in unit 2 are \$12.8 ($0.3 \times 40 + 0.0005 \times 40^2$). Its opportunity costs for the first time slot would be \$3.05 (i.e. a difference of \$9.75). For the calculation of the offer price, agent B has to determine its additional rescheduling costs. Therefore agent B has to negotiate with agent A in unit 2 for the next best time slot and in unit 1, because the patients are exclusive resources which can only perform one task at a time. Agent A loses \$3.25 in unit 2 and another \$3.15 in unit 1. Agent B loses \$3.15 in unit 2 but gains \$3.15 in unit 1. Agent B is charged with the losses of Agent A (\$6.40) and adds this to its supply price for agent C (here $\$3.15 - \$3.15 = \$0.00$). Because the total supply price (\$6.40) is less than agent C's gains (\$9.75) agent C accepts the deal (because it can compensate the losses of the other agents) and the plan is changed. No more deals are possible and figure 2(b) shows the result of this negotiation.

In this example, patient agent A accepted a schedule of lower quality in favour of patient agent C. However, only patient agent A – not the real patient – has received a compensation from patient agent C. This illustrates the necessary disjunction between the real patients and the multi-agent system in the hospital domain as described earlier (section 3.2).

In the next step, we relax the assumption, that all agents have the same cost functions. If agent C would have had a lower health state (e.g. $\mu = 0.35$) it could have even improved its appointment in unit 1 because it would have gained \$3.75 from moving to the second slot (agent A in figure 2(b)) while

agent A would only ask \$3.25 for its slot (agent A moves to slot 1 and gains \$3.15, and agent B moves to slot 3 and loses \$6.40).

We have applied the described coordination mechanism to the first Taillard 5×5 open shop problem [Taillard, 1993] to analyse the behaviour for tasks with different durations. Figure 3 shows the resulting Gantt chart with a solution of 349. The optimal solution is 300. To analyse the convergence of the mechanism, we logged the changes of the latest task for each agent. Figure 4 shows the corresponding convergence behaviour, which indicate a fast convergence in the first third of the graph.

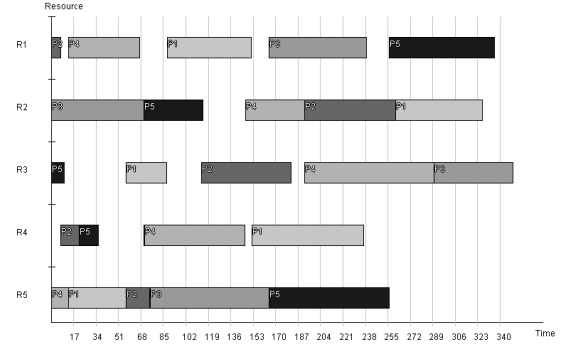


Figure 3: Gantt of the results of the Taillard 5x5 open shop problem.

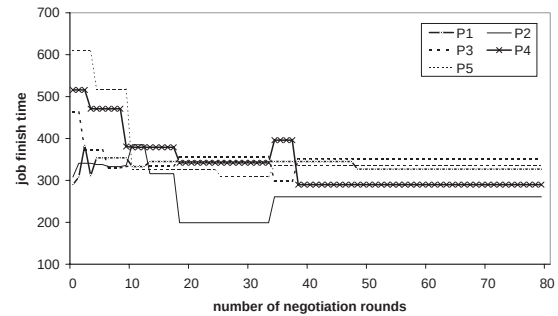


Figure 4: Converge of the Taillard 5x5 open shop problem.

The *number of negotiation rounds* counts each improvement attempt (successful or not) of the agents. The same procedure was executed for the $n \times n$ open shop problems with *equal task durations*. For these problems the optimal solution will be always achieved. The resulting Gantt chart is shown in figure 5. Figure 6 shows the corresponding convergence, illustrating the simplicity of this problem type in contrast to the used Taillard benchmark.

However, a better agent-based solution to this problem can be achieved through the usage of (central) appointment auctions. Implementing an auction based approach using the same cost functions resulted in a solution at 308. However, these auctions cannot handle the dynamics of the patient scheduling problem, because the auction process would have

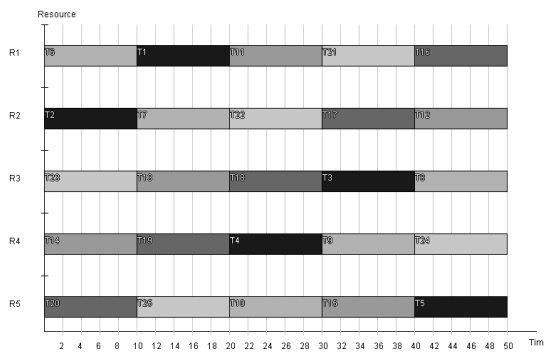


Figure 5: Gantt of the results of the easy 5x5 open shop problem.

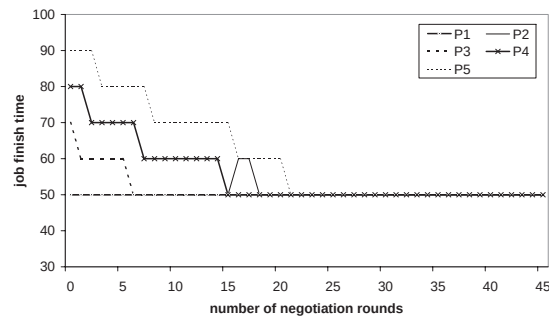


Figure 6: Converge of the easy 5x5 open shop problem.

to restart again after severe changes in the pathway occurred.

4 Conclusions and Future Work

In this paper, we presented a patient-centred coordination mechanism for inter-unit patient scheduling in hospitals. Therefore, health state dependent cost functions were developed. We described the implemented coordination mechanism and evaluated first results, showing the different convergence for problems with equal and different task durations.

In the next step, we will run our approach on real hospital data, already retrieved out of a field study in five hospitals. Based upon these results, our coordination mechanism will be enhanced in future work, where the main focus is on the development of a flexible auction mechanism in order to handle the dynamics of the patient scheduling problem. In this context, we will analyse how the usage of texture measurements (here: criticality and goodness measures) [Sycara *et al.*, 1990] can facilitate our coordination mechanism in order to reduce backtracking. Additionally, our agent based approach will be benchmarked with a genetic and evolutionary algorithm, which is under development in the context of this work. Further, a decision theoretic approach facilitating stochastic task duration will be presented in the near future.

References

[Adam and Gorschlüter, 1999] D. Adam and P. Gorschlüter. Qualitätsmanagement im Krankenhaus. *Zeitschrift für Be-*

triebswirtschaftslehre (ZfB), Ergänzungsheft 5: Krankenhausmanagement, 1999.

[Decker and Li, 2000] K. Decker and J. Li. Coordinating mutually exclusive resources using GPGP. *Autonomous Agents and Multi-Agent Systems*, 3(2):133–157, 2000.

[Jennings, 2001] N. R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, April 2001.

[Knaus *et al.*, 1985] W. A. Knaus, E. A. Draper, D. P. Wagner, and J. E. Zimmermann. Apache ii: a severity of disease classification system. *Critical Care Medicine*, 13(10):818–829, 1985.

[Pedroni and Zweifel, 1990] G. Pedroni and P. Zweifel. *Wie misst man Gesundheit?* Pharma-Information, Basel, 1990.

[Rosenschein and Zlotkin, 1994] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter*. MIT Press, Cambridge, MA, 1994.

[Schlüchtermann, 1990] J. Schlüchtermann. *Patientensteuerung*. Verlag Josef Eul, Bergisch Gladbach, Germany, 1990.

[Sycara *et al.*, 1990] K. Sycara, S. Roth, N. Sadeh, and M. S. Fox. An investigation into distributed constraint-directed factory scheduling. In *Proceedings of the Sixth IEEE Conference on AI Applications (CAIA-90)*, Santa Barbara, CA, 1990.

[Taillard, 1993] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operations Research*, 64:278–285, 1993.

[Torrance, 1987] G. W. Torrance. Utility approach to measuring health-related quality of life. *Journal of Chronic Diseases*, 1987.

[Weigelt, 1994] M. Weigelt. *Dezentrale Produktionssteuerung mit Agenten-Systemen: Entwicklung neuer Verfahren und Vergleich mit zentraler Lenkung*. Dt. Univ.-Verl., Wiesbaden, 1994.

[Weinhardt and Gomber, 1996] Ch. Weinhardt and P. Gomber. Domänenunabhängige Koordinationsmechanismen für die dezentrale betriebliche Planung. *Information Management*, pages 6–16, 1996.

[Wellman *et al.*, 2001] M. Wellman, W. Walsh, P. Wurman, and J. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35:271–303, 2001.

