

# On the Parallelization of Randomized Propagation-based Constraint Solving

Alejandro Arbelaez<sup>1</sup>, Charlotte Truchet<sup>2</sup>, and Philippe Codognet<sup>3</sup>

<sup>1</sup> INSIGHT Centre for Data Analytics  
University College Cork, Ireland  
alejandros.arbelaez@insight-centre.org

<sup>2</sup> LINA, UMR 6241/ University of Nantes  
charlotte.truchet@univ-nantes.fr

<sup>3</sup> JFLI - CNRS / UPMC / University of Tokyo  
codognet@is.s.u-tokyo.ac.jp

**Abstract.** We propose to apply a general framework for estimating the parallel performance of *Las Vegas* algorithms to randomized propagation-based constraint solvers, *i.e.*, when, in the labeling phase, both the variable to instantiate and its value are chosen randomly. Indeed, by analyzing the runtime of the sequential process (which is variable from one run to another because of the stochastic nature of the algorithm) and approximating this runtime distribution with statistical methods, the runtime behavior of the parallel process can be predicted by a model based on *order statistics*. We apply this approach to study the behavior of a randomized version of the Gecode solver on three classical CSP problems, and compare the predicted performance to the results of an actual experimentation on parallel hardware up to 256 cores.

## 1 Introduction

In the last years, parallel algorithms for solving hard combinatorial problems, such as Constraint Satisfaction Problems (CSP), have been of increasing interest in the scientific community. The combinatorial nature of the problem makes it difficult to parallelize existing solvers, without costly communication schemes. Several parallel schemes have been proposed for incomplete or complete solvers, the most popular being to run several competing instances of the algorithm on different cores with different initial conditions or parameters, and let the fastest process win over others. The resulting parallel algorithm thus terminates with the minimal runtime among the launched processes. The framework of independent multi-walk parallelism, seems to be a promising way to deal with large-scale parallelism. Cooperative algorithms might perform well on shared-memory machines with a few tens of cores, but are difficult to extend efficiently on distributed hardware. This leads to so-called independent multi-walk algorithms in the CSP community [1] and portfolio algorithms in the SAT community (satisfiability of Boolean formula) [2].

However, although it is easy to obtain good speed-up on a small-scale parallel machine (*viz.* with a few tens of cores), it is not easy to know how a parallel

variant of a given algorithm would perform on a massively parallel machine (*viz.* with thousands of cores). Parallel performance models are thus particularly important for parallel constraint solvers, and any indication on how a given algorithm (or, more precisely, a couple formed by the algorithm and the problem instance) would scale on massively parallel hardware is valuable. If it is possible to estimate the maximum number of cores until which parallelization is efficient, then the actual parallel computing power needed to solve a problem can be deduced. This can be a decisive information when one knows supercomputers or systems such as Google Cloud and Amazon EC2 can be rented by core-hour with a limit on the maximum number of cores to be used.

The goal of this paper is to study the parallel performances of randomized constraint solving algorithms under the independent multi-walk scheme, and to model performances of the parallel execution from the runtime distribution of sequential runs of a given algorithm. In particular, we focus our attention in propagation-based backtrack search with random heuristics. More precisely, we first approximate the empirical sequential runtime distribution by a well-known statistical distribution (*e.g.*, exponential or lognormal) and then derive the runtime distribution of the parallel version of the solver. Our model is related to *order statistics*, a rather recent domain of statistics [3], which is the statistics of sorted random draws. Our method encompasses any algorithm whose solving time is random, and makes it possible to formally determine the average parallel runtime of such algorithms for any number of cores.

We will confront the performance predicted by the statistical model with actual speed-ups obtained for parallel implementations of the above-mentioned algorithms and show that the prediction can be quite accurate. More interestingly, we can also model both the initial and the asymptotic behavior of the parallel algorithm.

The paper is structured as follows. Section 2 presents background information on Constraint Solving. Section 3 presents a general framework to analysis and estimate the performance of parallel solvers based on the multi-walk architecture. Section 4 presents an experimental analysis of the performance of the parallel algorithm. A conclusion and perspective on future work end the paper.

## 2 Preliminaries

A Constraint Satisfaction Problem (CSP) is a triple  $\langle X, D, C \rangle$  where,

- $X = \{X_1, X_2, \dots, X_n\}$  represents a set of  $n$  variables;
- $D = \{D_1, D_2, \dots, D_n\}$  represents the set of associated domains, *i.e.*, possible values for the variables;
- $C = \{C_1, C_2, \dots, C_m\}$  represents a finite set of constraints.

Each constraint  $C_i$  is associated to a set of variables, and is used to restrict the combinations of values between these variables. Solving a CSP involves finding a solution, *i.e.*, an assignment of values to variables such that all the constraints are satisfied. If a solution exists, the problem is stated as satisfiable, and unsatisfiable

otherwise. Typically, a backtracking algorithm is used to tackle CSPs. At each step of the search a value is assigned to a given variable. Each assignment is combined with a look-ahead process called constraint propagation which can prune the domains of the remaining variables.

The backtracking algorithm is usually equipped with a restart strategy which helps to reduce the effects of early mistakes in the search. Currently, there are several methodologies, among which the most important are: static restarts [4], that implements a static restart policy by means of restarting every  $c$  backtracks, geometric restarts [5] that systematically increase the cutoff of each restart by multiplying it by a constant factor (e.g.,  $\times 1.2$ ), Luby [6] defines the cutoff of each restart based on the sequence  $Luby = \{1, 1, 2, 1, 1, 2, 4, 1, 1, \dots\}$ , the cutoff of the  $i^{th}$  restart is  $p \times Luby(i)$ .

In this paper we focus our attention on randomized propagation-based constraint solving, where variables and values are selected uniformly at random. In addition, we analyse the degree of parallelisation of this approach through the Multi-walk framework, that is, of executing multiple copies of the algorithms with different starting seeds.

Early experiments on the parallelization of propagation-based complete solvers date back to the beginning of Constraint Logic Programming, cf. [7], and used the search parallelism of the host logic language [8]. Most of the proposed implementations have been based on some kind of OR-parallelism, splitting the search space between different cores and relying on the shared-memory multicore architecture as the different cores work on shared data-structures representing a global environment in which the subcomputations take place. However only very few implementations of efficient constraint solvers on such machines have ever been reported, for instance [9] or [10] for a shared-memory architecture with 8 CPU cores. The Comet system [11] has been parallelized for small clusters of PCs, both for its Local Search solver [12] and its propagation-based constraint solver [13]. More recent experiments have been done up to 12 cores [14].

Search-space splitting techniques such as domain decomposition have also been implemented in the domain of Constraint Programming, but initial experiments [15] show that the speed-up tends to flatten after a few tens of cores, thus far away from linear speed-up. A recent approach based on a smaller granularity domain decomposition [16] shows better performance. The results for all-solution search on classical CSPLib benchmarks are quite encouraging and show an average speed-up of 14 to 20 up with 40 cores w.r.t. the base sequential solver. Another promising work is described in [17] where the authors describe parallel discrepancy-based search (PDS), this work aims at providing an efficient balance of the workload assigned to the processors and minimising the communication between them at the same time. Experiments on industrial instances report a 75% speedup factor with 4096 cores.

In the domain of combinatorial optimization, the most popular complete method that has been parallelized at a large scale is the classical branch and bound method [18], because it does not require much information to be communicated between parallel processes: basically only the current upper (or lower)

bound of the solution. It has thus been a method of choice for experimenting the solving of optimization problems using grid computing, see for instance [19] and also [20], which use several hundreds of nodes of the Grid'5000 platform. Good speed-ups are achieved up to a few hundreds of cores but interestingly, their conclusion is that runtimes tend to stabilize afterward. A simple solving method for project scheduling problems has also been implemented on an IBM Bluegene/P supercomputer [21] up to 1,024 cores, but with mixed results since they reach linear speed-ups until 512 cores only, and then no improvements beyond this limit.

### 3 Speed-Up Analysis Through Runtime Distributions

The notion of *runtime distributions* has been introduced by [22] to characterise the cumulative distribution function of the sequential execution time of stochastic algorithms. It has been used mainly to analyze the behavior of local search algorithms in the context of SAT [23], network design [24], and TSP [25].

A general statistical model for studying the performance of parallel multi-walk algorithms has been recently proposed in [26]. We will now present a brief summary of this model, which will be used in the rest of the paper to study the behavior of randomized propagation-based constraint solving.

#### 3.1 How to Estimate Parallel Speed-Up ?

Let  $Y$  be the runtime of a Local Search algorithm on a given problem instance. It can be considered as a random variable with values in  $\mathbb{N}$  (*number of iterations*), or in  $\mathbb{R}$  (*cpu-time*). In general, it is more convenient to consider distributions with values in  $\mathbb{R}$  because calculations are easier.  $Y$  can be studied through its cumulative distribution, which is by definition, the function  $\mathcal{F}_Y$  s.t.  $\mathcal{F}_Y(x) = \mathbb{P}[Y \leq x]$ . By definition, the distribution of  $Y$  is the derivative of  $\mathcal{F}_Y$ :  $f_Y = \mathcal{F}'_Y$ . The expectation of  $Y$  is defined as

$$\mathbb{E}[Y] = \int_0^\infty t f_Y(t) dt$$

Assume that  $n$  copies of the base algorithm are running in parallel on  $n$  cores. The first process finding a solution terminates all others, and the overall parallel algorithm terminates. The  $i$ -th process corresponds to a draw of a random variable  $X_i$ , following the distribution  $f_Y$ . The variables  $X_i$  are thus independent and identically distributed (i.i.d.). The computation time of the whole parallel process is also a random variable, let  $Z^{(n)}$ , with a distribution  $f_{Z^{(n)}}$  that depends both on  $n$  and on  $f_Y$ . Since all the  $X_i$  are i.i.d., the cumulative distribution  $\mathcal{F}_{Z^{(n)}}$  and the distribution  $f_{Z^{(n)}}$  can be computed as follows:

$$\begin{aligned} \mathcal{F}_{Z^{(n)}} &= \mathbb{P}[Z^{(n)} \leq x] &= \mathbb{P}[\exists i \in \{1 \dots n\}, X_i \leq x] \\ &= 1 - \mathbb{P}[\forall i \in \{1 \dots n\}, X_i > x] &= 1 - (1 - \mathcal{F}_Y(x))^n \\ f_{Z^{(n)}} &= (1 - (1 - \mathcal{F}_Y)^n)' = n f_Y (1 - \mathcal{F}_Y)^{n-1} \end{aligned}$$

Thus, knowing the distribution for the base algorithm  $Y$ , one can calculate the distribution for  $Z^{(n)}$ . The formula shows that the parallel algorithm favors short runs, by terminating slower processes. Thus, compared to the distribution of  $Y$ , the distribution of  $Z^{(n)}$  moves toward the origin and is more peaked.

We can also compute the expectation  $\mathbb{E}[Z^{(n)}]$  for the parallel process, from which we derive the expected speed-up  $\mathcal{G}_n$  of the parallel algorithm versus the sequential one:

$$\mathbb{E}[Z^{(n)}] = n \int_0^\infty t f_Y(t) (1 - F_Y(t))^{n-1} dt$$

$$\mathcal{G}_n = \mathbb{E}[Y] / \mathbb{E}[Z^{(n)}]$$

Again, no explicit general formula can be computed and the expression of the speed-up will depend on the distribution of  $Y$ . We will thus study in the following different specific distributions. This computation of the speed-up is actually related to a field of statistics called order statistics, see [3] for a detailed presentation. Order statistics are the statistics of sorted random draws. For instance, the first order statistics of a distribution is its minimal value. For predicting the speedup, we are indeed interested in computing the expectation of the distribution of the minimum draw. As the above formula suggests, this may lead to costly calculations, but recent studies such as [27] give explicit formulas defining this quantity for several classical probability distributions.

### 3.2 Shifted and non-Shifted Exponential Distribution

On each problem, the sequential benchmark gives observations of the distribution of the algorithm runtime  $f_Y$ . Yet, the exact distribution is still unknown. It can be any real distribution, not even a classical one. In the following, we will rely on the assumption that  $Y$  is distributed with a known parametric distribution. We perform a statistical test, called Kolmogorov-Smirnov test, on the hypothesis  $\mathcal{H}_0$  that the collected observations correspond to a theoretical distribution. Assuming  $\mathcal{H}_0$ , the test first computes the probability that the distance between the collected data and the theoretical distribution does not significantly differ from its theoretical value. This probability is called the  $p$ -value.

Then, the  $p$ -value is compared to a fixed threshold (usually 0.05). If it is smaller, one rejects  $\mathcal{H}_0$ . For us, it means that the observations do not correspond to the theoretical distribution. If the  $p$ -value is high, we will consider that the distribution of  $Y$  is approximated by the theoretical one. Note that the Kolmogorov-Smirnov test is a statistical test, which in no way proves that  $Y$  follows the distribution. However, it measures how well the observations fit a theoretical curve and, as it will be seen in the following, it is accurate enough for our purpose.

Assume that a given algorithm  $Y$  has a shifted exponential distribution, as would be the case for an ideal randomized algorithm. The minimum distribution can be computed by a direct integration:

$$\begin{aligned}
f_Y(t) &= \begin{cases} 0 & \text{if } t < x_0 \\ \lambda e^{-\lambda(t-x_0)} & \text{if } t > x_0 \end{cases} \\
\mathcal{F}_Y(t) &= \begin{cases} 0 & \text{if } t < x_0 \\ 1 - e^{-\lambda(t-x_0)} & \text{if } t > x_0 \end{cases} \\
\mathbb{E}[Y] &= x_0 + 1/\lambda \\
f_{Z^{(n)}}(t) &= \begin{cases} 0 & \text{if } t < x_0 \\ n\lambda e^{-n\lambda(t-x_0)} & \text{if } t > x_0 \end{cases} \\
\mathcal{F}_{Z^{(n)}}(t) &= \begin{cases} 0 & \text{if } t < x_0 \\ 1 - e^{-n\lambda(t-x_0)} & \text{if } t > x_0 \end{cases}
\end{aligned}$$

In case of a non-shifted exponential distribution,  $x_0 = 0$  and the speed-up is thus equal to the number of cores  $n$ , up to infinity. This case has already been studied by [1]. However for  $x_0 > 0$ , the speed-up admits a finite limit, even when  $n$  tends to infinity, which is  $\frac{x_0 + \frac{1}{\lambda}}{x_0} = 1 + \frac{1}{x_0\lambda}$ . The closer to zero  $x_0$  is, the larger the limit.

Other distributions can be considered, depending on the behavior of the base algorithm. Indeed, in [26, 28] we have observed that the lognormal distribution can be used to characterize the sequential runtime distribution of a large variety of local search solvers to tackle SAT and CSPs instances.

## 4 Experimental Protocol and Benchmarks

We detail here the performance and speed-ups obtained with both sequential and parallel multi-walk randomized propagation-based Constraint Solving. We have chosen to test this method on a hard combinatorial problem abstracted from radar and sonar applications (COSTAS ARRAY) and two problems from the CSPLib benchmark library<sup>4</sup>: ALL-INTERVAL Series (prob007 in CSPLib), and the MAGIC-SQUARE problem (prob019 in CSPLib).

A Costas array is an  $N \times N$  grid containing  $N$  marks such that there is exactly one mark per row and per column and the  $N(N-1)/2$  vectors joining the marks are all different. It is convenient to see the COSTAS ARRAY Problem (CAP) as a permutation problem by considering an array of  $N$  variables  $(V_1, \dots, V_N)$  which forms a permutation of  $\{1, 2, \dots, N\}$ . We refer the reader to [29] for a complete survey on the COSTAS ARRAY Problem and to [30] for the modeling and solving by Local Search.

In this paper we used the model of ALL-INTERVAL and MAGIC-SQUARE provided in the Gecode distribution (version 4.2.1), and the gecode version of COSTAS ARRAY is available at [http://www.hakank.org/gecode/costas\\_array.cpp](http://www.hakank.org/gecode/costas_array.cpp). It is worth noticing that the ALL-INTERVAL model for Gecode allows the trivial solution  $(0, n-1, n-2, n-3, \dots)$ , and such a solution is obtained without backtracking. Therefore, we removed the trivial solution in the model by adding

---

<sup>4</sup> <http://www.csplib.org>

the constraint  $X_0 > 0$ , i.e., the first variable in the problem definition must be set greater than 0.

In order to perform the following experimentation we equipped Gecode<sup>5</sup> with the multi-walk framework. We also used a randomized version of the solver: at each node, the variable to instantiate is selected randomly with a uniform probability among the free variables, and the value for the chosen variable is also selected uniformly at random. It is highly recognized that the runtime distribution for tree-based search methods for a large number of problems is heavy-tailed [31] and adding restarts (*i.e.*, stopping the search after a given number of backtracks, and re-initializing it) help to alleviate the heavy-tailed phenomenon. In this paper, we use a fixed restart strategy where the restart cutoff is 250 backtracks.

This set of experiments were performed on the Fujitsu FX10 supercomputer of the University of Tokyo with a total number of 76,800 cores. This machine is composed of 4,800 nodes, each of which is composed of one SPARC64 IXfx processor (16 cores, 1.848GHz, 768KB of L2-cache/core) with 32GB of memory. We performed 300 runs of the algorithm for the experiments in sequential settings, and 50 runs of the multi-walk parallel algorithm.

#### 4.1 Analysis of the Parallel Speed-ups

Table 1 shows the runtime of the sequential version. We present the minimum, maximum, mean, and standard deviation of the sequential runs for MAGIC-SQUARE 7, ALL-INTERVAL 15, and COSTAS ARRAY 17. The Kolmogorov-Smirnov test suggests that sequential execution for the three problems can be characterized using a shifted exponential distribution with a p-value of: 0.90 (MS 7), 0.86 (AI 15) and 0.90 (CAP 17). We use the formulas described in the previous section with the estimated parameter for the theoretical distribution, *i.e.*,  $\lambda=1/\text{mean}$  and  $x_0$  is set to the minimal value of the empirical data.

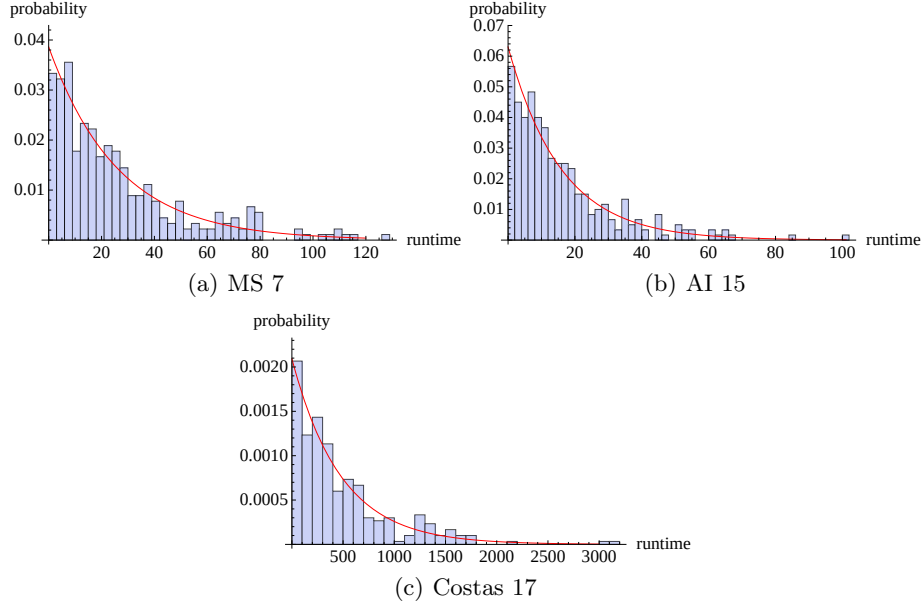
Taking this into account, Figure 1 depicts the normalized histogram of the sequential runtime for each problem instance and the approximated shifted exponential distribution.

|         | MS 7   | AI 15   | Costas 17 |
|---------|--------|---------|-----------|
| Min     | 0.35   | 0.008   | 0.04      |
| Max     | 127.69 | 101.128 | 3134.75   |
| Mean    | 26.24  | 15.869  | 477.77    |
| Std Dev | 25.37  | 479.63  | 15.47     |

**Table 1.** Sequential executions in seconds for complete search

Tables 2 and 3 report the estimated runtime and speed-up of the multi-walk version of the propagation-based constraint solving algorithm. Interestingly, this

<sup>5</sup> [www.gecode.org](http://www.gecode.org)



**Fig. 1.** Histogram for the observed runtimes for complete search

simple parallel scheme exhibits near optimal empirical speed-ups up to 256 cores for COSTAS ARRAY (factor 232), while the speed-up drops for MAGIC-SQUARE and ALL-INTERVAL when using 128 cores to a factor 105 and 115 respectively.

## 4.2 Analysis and Discussion

We recall that the goal of this paper is not to design a novel parallel algorithm. Instead, we intend to show the application of the proposed framework to a large variety of randomized techniques to solve combinatorial problems. Interestingly, even though the search strategy used in these experiments largely differs from the experiments presented in [26, 28], we were still able to characterize the empirical data into a theoretical distribution, *i.e.*, shifted exponential distribution, and thus estimate the runtime of the parallel algorithm.

The runtime prediction is very close to the empirical data up to 64 cores for MS with a gap of 39% to the actual value, 128 cores for AI with a gap of 7%, and 256 cores for Costas with a gap of 6%. A similar picture is observed in the predictions of the speed-up (see Table 3). On the other hand, the gap between the predicted and the experimental values increases as the number of cores increases, we attribute this to the variability of the *propagation time*, that is, the time spent performing propagation. Indeed the time spent for variable domain reduction might change drastically at different stages of the search tree. For instance, a given propagator<sup>6</sup> might prune more values at the root than at

<sup>6</sup> A propagator can be understood as an implementation of a given constraint.

a leaf node. Finally, we would like to point out that this methodology requires independently and identically distributed executions of the sequential algorithm, so that, parallel algorithms based on search-splitting techniques [15, 10] and co-operative techniques [32] do not fit within this framework.

| Problem   | 1 core |           | time on $k$ cores |       |      |      |      |  |
|-----------|--------|-----------|-------------------|-------|------|------|------|--|
|           |        |           | 16                | 32    | 64   | 128  | 256  |  |
| MS 7      | 26.24  | Actual    | 1.47              | 0.68  | 0.46 | 0.24 | 0.13 |  |
|           |        | Predicted | 1.99              | 1.17  | 0.76 | 0.55 | 0.45 |  |
| AI 15     | 15.86  | Actual    | 0.92              | 0.37  | 0.26 | 0.14 | 0.13 |  |
|           |        | Predicted | 1.00              | 0.50  | 0.25 | 0.13 | 0.07 |  |
| Costas 17 | 477.77 | Actual    | 27.53             | 12.90 | 7.39 | 4.12 | 2.05 |  |
|           |        | Predicted | 29.90             | 14.97 | 7.51 | 3.77 | 1.91 |  |

**Table 2.** Parallel runtimes in seconds

| Problem   |           | speed-up on $k$ cores |       |       |        |        |  |
|-----------|-----------|-----------------------|-------|-------|--------|--------|--|
|           |           | 16                    | 32    | 64    | 128    | 256    |  |
| MS 7      | Actual    | 17.84                 | 38.49 | 56.69 | 105.26 | 199.22 |  |
|           | Predicted | 13.18                 | 22.41 | 34.50 | 47.23  | 57.92  |  |
| AI 15     | Actual    | 17.09                 | 42.73 | 60.83 | 109.29 | 116.37 |  |
|           | Predicted | 15.86                 | 31.44 | 61.83 | 119.62 | 224.54 |  |
| Costas 17 | Actual    | 17.35                 | 37.03 | 64.58 | 115.75 | 232.52 |  |
|           | Predicted | 15.97                 | 31.89 | 63.59 | 126.40 | 249.68 |  |

**Table 3.** Speed-Up for complete search

## 5 Conclusion and Future Work

This paper we have studied the behavior of a simple (independent) parallel version of propagation-based constraint solving with randomization. Interestingly, we have observed that, for the different instances considered in this study, the runtime distributions can be characterized using the (shifted) exponential distribution. In previous studies on different local search algorithms for SAT and CSP problems, some instances exhibit a (shifted) lognormal distribution.

For a medium-size number of cores (up to 64), the model is quite accurate and predicts performance close to the empirical data, but for a larger number of cores (128 and 256) accuracy is only verified for one problem, while there is some divergence between the actual and the predicted performance of the two other benchmark problems. This is less accurate than results obtained for local search algorithms.

Another limitation of our approach is that, in practice, we need to be able to approximate the sequential distribution. In addition, this distribution must be one of the distributions for which the first order statistics is known, symbolically (as the exponential) or numerically (as the lognormal). Nevertheless, recent results in the field of order statistics give explicit formulas for a number of useful distributions: Gaussian, lognormal, gamma, beta. This provides a wide range of tools to analyze different behaviors.

Another interesting extension of this work would be to devise a method for predicting the speed-up from scratch, that is, without any knowledge on the algorithm distribution. Our observations suggest that the sequential runtime of randomized propagation-based constraint solving on the experimented instances is well approximated by the shifted exponential distribution. This could be intensively tested on a wider range of problems. In particular, it is important to know whether the sequential distribution of different instances of a given problem belong to the same family. Then we can devise a method for estimating the sequential distribution based on a limited number of observations, possibly on small instances, and then estimate the parallel speed-up for larger instances. This would allow us to predict if a simple multi-walk parallelization scheme, which does not imply to modify the algorithms, is likely to be efficient, or not.

We are also interested in analysing randomised tree-based algorithm following more sophisticated variable selection heuristics such as: impacts [33] and dom-wdeg [34], as well as additional restart strategies, e.g., Luby [6], and geometric restarts [5]. Finally, we plan to estimate how many cores these randomised portfolio strategies need to outperform dedicated heuristics for a given problem, e.g., protein structured prediction [35], and the container stowage problem [36].

## Acknowledgments

The first author is currently supported by DISCUS, and FP7 Grant Agreement 318137

## References

1. Verhoeven, M., Aarts, E.: Parallel local search. *Journal of Heuristics* **1**(1) (1995) 43–65
2. Gomes, C.P., Selman, B.: Algorithm portfolios. *Artificial Intelligence* **126**(1-2) (2001) 43–62
3. David, H., Nagaraja, H.: Order Statistics. Wiley series in probability and mathematical statistics. Probability and mathematical statistics. John Wiley (2003)
4. Gomes, C., Selman, B., Kautz, H.: Boosting combinatorial search through randomization. In: *AAAI/IAAI*. (1998) 431–437
5. Walsh, T.: Search in a small world. In: *IJCAI*. (1999) 1172–1177
6. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of las vegas algorithms. *Information Processing Letters* **47** (1993) 173–180
7. Van Hentenryck, P.: Parallel constraint satisfaction in logic programming: Preliminary results of CHIP within PEPSys. In: *ICLP’89, International Conference on Logic Programming*, MIT Press (1989) 165–180

8. de Kergommeaux, J.C., Codognet, P.: Parallel logic programming systems. *ACM Computing Surveys* **26**(3) (1994) 295–336
9. Perron, L.: Search procedures and parallelism in constraint programming. In: CP'99, 5th Int. Conf. on Principles and Practice of Constraint Programming. *Lecture Notes in Computer Science*, Springer Verlag (1999) 346–360
10. Chu, G., Schulte, C., Stuckey, P.J.: Confidence-based work stealing in parallel constraint programming. In Gent, I.P., ed.: CP 2009, 15th International Conference on Principles and Practice of Constraint Programming. Volume 5732 of *Lecture Notes in Computer Science.*, Springer (2009) 226–241
11. Van Hentenryck, P., Michel, L.: *Constraint-Based Local Search*. The MIT Press (2005)
12. Michel, L., See, A., Van Hentenryck, P.: Distributed constraint-based local search. In Benhamou, F., ed.: CP'06, 12th Int. Conf. on Principles and Practice of Constraint Programming. *Lecture Notes in Computer Science*, Springer Verlag (2006) 344–358
13. Michel, L., See, A., Van Hentenryck, P.: Parallelizing constraint programs transparently. In Bessiere, C., ed.: CP'07, 13th Int. Conf. on Principles and Practice of Constraint Programming. *Lecture Notes in Computer Science*, Springer Verlag (2007) 514–528
14. Michel, L., See, A., Van Hentenryck, P.: Parallel and distributed local search in comet. *Computers and Operations Research* **36** (2009) 2357–2375
15. Bordeaux, L., Hamadi, Y., Samulowitz, H.: Experiments with massively parallel constraint solving. In Boutilier, C., ed.: *Proceedings of IJCAI 2009, 21st International Joint Conference on Artificial Intelligence*. (2009) 443–448
16. Régim, J.C., Rezgui, M., Malapert, A.: Embarrassingly parallel search. In Schulte, C., ed.: *Proceedings of CP'2013, 19th International Conference on Principles and Practice of Constraint Programming*. Volume 8124 of *Lecture Notes in Computer Science.*, Springer Verlag (2013) 596–610
17. Moisan, T., Gaudreault, J., Quimper, C.G.: Parallel discrepancy-based search. In: CP. (2013) 30–46
18. Gendron, B., Crainic, T.: Parallel branch-and-bound algorithms: Survey and synthesis. *Operations Research* **42**(6) (1994) 1042–1066
19. Aida, K., Osumi, T.: A case study in running a parallel branch and bound application on the grid. In: SAINT '05: *Proceedings of the The 2005 Symposium on Applications and the Internet*, Washington, DC, USA, IEEE Computer Society (2005) 164–173
20. Caromel, D., di Costanzo, A., Baduel, L., Matsuoka, S.: Grid'BnB: a parallel branch and bound framework for grids. In: *proceedings of HiPC'07, 14th international conference on High performance computing*, Springer Verlag (2007) 566–579
21. Xie, F., Davenport, A.J.: Massively parallel constraint programming for supercomputers: Challenges and initial results. In: CPAIOR'10, 7th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Volume 6140 of *Lecture Notes in Computer Science.*, Springer (2010) 334–338
22. Hoos, H., Stütze, T.: *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann (2005)
23. Hoos, H.H., Stütze, T.: Towards a Characterisation of the Behaviour of Stochastic Local Search Algorithms for SAT. *Artif. Intell.* **112**(1-2) (1999) 213–232
24. Ribeiro, C.C., Rosseti, I., Campos, R.V.: Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. *J. Global Optimization* **54**(2) (2012) 405–429

25. Stutzle, T., Hoos, H.H.: Analyzing the run-time behaviour of iterated local search for the tsp. In: MIC'99. (1999)
26. Truchet, C., Richoux, F., Codognet, P.: Prediction of Parallel Speed-ups for Las Vegas Algorithms. In Dongarra, J., Robert, Y., eds.: Proceedings of ICPP-2013, 42nd International Conference on Parallel Processing, IEEE Press (October 2013)
27. Nadarajah, S.: Explicit expressions for moments of order statistics. *Statistics & Probability Letters* **78**(2) (February 2008) 196–205
28. Arbelaez, A., Truchet, C., Codognet, P.: Using sequential runtime distributions for the parallel speedup prediction of sat local search. *TPLP* **13**(4-5) (2013) 625–639
29. Drakakis, K.: A review of costas arrays. *Journal of Applied Mathematics* **2006** (2006) 1–32
30. Diaz, D., Richoux, F., Caniou, Y., Codognet, P., Abreu, S.: Parallel local search for the costas array problem. In: IEEE Workshop on new trends in Parallel Computing and Optimization (PC012), in conjunction with IPDPS 2012, Shanghai, China, IEEE Press (May 2012)
31. Gomes, C.P., Selman, B., Crato, N., Kautz, H.A.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Autom. Reasoning* **24**(1/2) (2000) 67–100
32. Arbelaez, A., Codognet, P.: Massively Parallel Local Search for SAT. In: ICTAI'12, Athens, Greece, IEEE Computer Society (November 2012) 57–64
33. Refalo, P.: Impact-based search strategies for constraint programming. In: CP. (2004) 557–571
34. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: ECAI. (2004) 146–150
35. Delgado, A., Jensen, R.M., Schulte, C.: Generating optimal stowage plans for container vessel bays. In: CP. (2009) 6–20
36. Palù, A.D., Dovier, A., Pontelli, E.: A constraint solver for discrete lattices, its parallelization, and application to protein structure prediction. *Softw., Pract. Exper.* **37**(13) (2007) 1405–1449