

The DynCOAA Algorithm for Dynamic Constraint Optimization Problems

Koenraad Mertens, Tom Holvoet, Yolande Berbers

AgentWise, Distrinet
Department of Computer Science, K.U.Leuven
200A, Celestijnenlaan
3001 Leuven, Belgium

{Koenraad.Mertens, Tom.Holvoet, Yolande.Berbers}@cs.kuleuven.be

ABSTRACT

Numerous problems in software coordination, operations research, manufacturing control and others can be transformed in constraint optimization problems (COPs). Moreover, most practical problems change constantly, requiring algorithms that can handle dynamic problems.

In this paper we present the dynamic constraint optimization ant algorithm (DynCOAA) that can solve dynamic COPs. DynCOAA is specifically designed for dynamic problems, as it is based upon the ant colony optimization (ACO) meta-heuristic that has already proven its merit in other dynamic optimization problems. DynCOAA is a distributed algorithm that is suited for a one-on-one mapping between variables and hosts, but it can effectively accommodate multiple variables per host. We compared our algorithm to two existing algorithms for dynamic constraint optimization: DynAWC and DynDBA. We find that DynCOAA outperforms DynAWC and DynDBA on some problems, while remaining competitive on others.

Categories and Subject Descriptors

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms, Design, Performance

Keywords

Dynamic, ACO, Distributed Constraint Optimization

1. INTRODUCTION

Numerous problems in software coordination, operations research, manufacturing control and others can be transformed in constraint satisfaction and constraint optimization problems (CSPs and COPs). In many practical applications there is not one static

CSP or COP that has to be solved: practical problems change often. Changes can happen either because additional information is added to the model, or because of a changing nature in the problem that one tries to solve. An example of a changing problem is a GPS navigation system that has to recalculate the route when the car does not follow the proposed route.

When solving a changed version of a CSP or COP, the changed problem is often similar to the old problem: it only differs in a few constraints or variables. While it is possible that a small change in the problem creates a big change in the optimal solution, in most practical cases the optimal solutions of such similar problems do not differ a lot. E.g. in the GPS system when a car goes forward instead of turning left, the initial positions are almost the same and it may suffice to take the next left turn. Changing problems are called dynamic constraint satisfaction/optimization problems (DynCSP/DynCOP).

In this paper, we present a new distributed algorithm for constraint optimization that is specifically designed for dynamic problems. It is based upon the ant colony optimization (ACO) meta-heuristic [1]. In ACO, the environment of the multi-agent system represents the problem that has to be solved and the agents use this environment to give feedback to one another. This makes ACO-based algorithms naturally suited for dynamic problem solving, as changes in the problem are reflected automatically in the environment of the agents. In spite of this inherent suitability for dynamic problem solving, to this date and to the best of our knowledge, the dynamic constraint optimization ant algorithm (DynCOAA) that we present in this paper is the first distributed algorithm for dynamic constraint satisfaction or dynamic constraint optimization that uses the ACO meta-heuristic.

We compare DynCOAA to two other algorithms: the complete algorithm DynAWC [6] and the non-complete algorithm DynDBA [3]. We use two different cases for evaluating the algorithms. When the three algorithms are used to solve artificial graph coloring problems, DynDBA turns out to be the most effective algorithm. But when a real-world problem – scheduling ships that transport liquefied natural gas – is tested, our DynCOAA clearly beats the other two algorithms.

In the following section we present our dynamic constraint optimization ant algorithm (DynCOAA) and the algorithms we use for comparison. Section 3 describes the tests we performed and the results we found. Section 4 concludes this paper with an overview of our findings and some pointers for future work.

2. ALGORITHMS

In this section we present the dynamic constraint optimization ant algorithm (DynCOAA). The construction of the graph struc-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.
Copyright 2006 ACM 1-59593-303-4/06/0005 ...\$5.00.

ture and the behavior of the agents for a static problem are described in [4, 5]. When adding or removing variables or values in a dynamic problem, the corresponding nodes and edges have to be added or removed on the appropriate hosts. When adding or removing constraints, references to those constraints must be registered or deregistered with the appropriate nodes. While it is possible to leave all existing pheromone trails unchanged and only to initialize new edges with the default pheromone quantity, better results can be achieved with dynamic ACO algorithms when all pheromone quantities are equalized to a certain degree. We use the Restart-Strategy that they describe in their paper, using a λ_R value of 0.5, which gave the best results in their experiments. This means that all pheromone quantities are set to the average of the previous pheromone quantity and the default pheromone quantity.

The first algorithm we compare DynCOAA to is DynBDA. This is a non-complete algorithm, based on the Distributed Breakout Algorithm (DBA) and first introduced by [3]. DynBDA is said to be a good algorithm for fast-changing dynamic problems.

The second algorithm we use is DynAWC. This is a complete algorithm that uses nogood recording [6]. The processing of nogoods allows it to reuse information that was found in a previous search after a modification to the problem was made.

3. EXPERIMENTAL RESULTS

In this section we discuss the results of the tests we have done to compare DynCOAA with DynAWC and DynBDA. The three algorithms were implemented in C++, using the same constraint processing engine. We compare the algorithms for two different categories of problems. The first category of problems, discussed in Sect. 3.1, is artificial graph coloring problems. In Sect. 3.2 we discuss the results of a real-world problem, in which a schedule for a number of ships that transport liquified gas must be assembled. Section 3.3 gives an interpretation of the obtained results.

3.1 Graph Coloring Tests

For our first series of tests, we use the same problems as can be found in [3], i.e. graph coloring problems with 30 variables, which require a coloring with 3 colors. Graph densities varied over $\{2.0; 2.3; 2.5\}$. These densities represent the three different phases of the phase transition for random 3-coloring graphs (respectively mostly satisfiable; within the transition; and mostly unsatisfiable). The number of changes between successive problems, indicated by δ , varied over $\{2; 4; 6; 8; 12\}$, where 6 changes means that 3 constraints were removed from the problem and 3 constraints added to the problem, so that the overall density remained unchanged. For each combination of graph density and change rate δ , 20 problems were constructed, each consisting of an initial COP and 100 successive COPs.

As proposed in the future work section of [3], we measured the real-time performance of the algorithms: for each successive COP, the algorithms could calculate during 2 seconds. The best solution at each moment in time was stored. This form of measurement provides a more realistic indication of the performance than can be obtained through cycle-based tests.

Table 1 gives an overview of the results of the three algorithms. The table includes the average time (in seconds) that passed before all 20 problems found the first solution, the average cost of that first solution and the average cost of the best solution that was found after 2 seconds.

When looking at the table, an interesting observation can be made. DynCOAA needs about 5 times longer to calculate its first solution than is necessary for DynBDA and DynAWC. This is due to the fact that DynCOAA uses a swarm of 5 agents, who all need

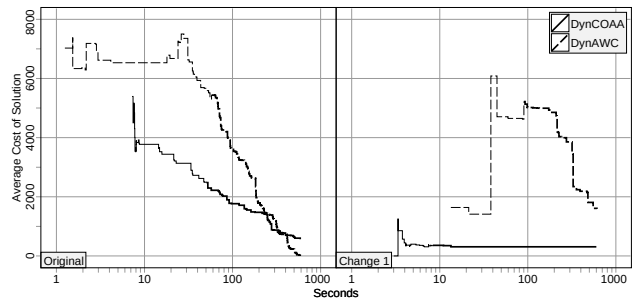


Figure 1: Anytime curves for DynCOAA (solid line) and DynAWC (dashed line) when assembling a schedule for ships. DynBDA was unable to find any solution.

about the same amount of time. But when we look at the final solution of DynCOAA and DynAWC during changes, DynCOAA clearly beats DynAWC, even though it takes about 5 times longer to find its first solution. The result of DynBDA remains better than that of DynCOAA.

As a conclusion, we can say that for these dynamic graph coloring problems, DynBDA is the best algorithm, closely followed by DynCOAA.

3.2 Real-World Ship Scheduling Tests

For our second series of tests, we decided to use a real-world problem instead of an artificial problem. This real-world problem is to assemble a schedule for transport ships. The ships transport liquified natural gas from different ports around the world to one destination port. Using data provided by Tractebel Engineering [2], the algorithms had to calculate a schedule for 7 ships, for a period of three months.

After a decent schedule had been found, we simulated a storm: one of the ships got delayed for three days. The algorithms had to recalculate the schedule, incorporating the delay. The goal was twofold: to obtain a good schedule, but also to leave the individual schedules for the 6 ships that did not get stuck in the storm as unchanged as possible.

This problem is quite complicated: the original scheduling problem is made up of a total of 69 variables and 101 constraints. The variable domains contain up to 200 possible values. 76 constraints are hard constraints (those have to be obeyed in each valid solution: it are constraints with a weight of infinity) and 25 are soft constraints, used for optimizing the cost function. 57 of the hard constraints are binary constraints. All other hard constraints involve at least 19 variables. The soft constraints even involve all variables. A solution with a cost of 1000 means that the local transport rate in the destination port has to deviate from the optimal rate during a total of one day (in smaller intervals distributed over the three months). At a cost of 2000, it has to deviate during a total of 2 days, etc.

Each of the three algorithms made 20 schedules. For the original schedule as well as for the storm-adaptation, 10 minutes of calculation were allowed. The parameters that were used for the DynCOAA are the same as described in Sect. 3.1, except for the swarm size, which was increased to 30. The anytime curves for this problem are shown in Fig. 1.

The first thing to notice is that DynBDA is missing from Fig. 1. During the 10 minutes of calculation time, none of the 20 runs were able to find any valid schedule (one that satisfies all hard constraints). The most probable reason for this is that a number of the constraints are global constraints, making it very hard for a lo-

Density	δ	DynCOAA			DynAWC			DynDBA		
		First Solution		Best Solution	First Solution		Best Solution	First Solution		Best Solution
		Time	Cost	Cost	Time	Cost	Cost	Time	Cost	Cost
2.0	2	0.13	3.99	2.42	0.053	19.24	8.48	0.034	1.90	1.40
	4	0.13	4.85	3.00	0.044	19.86	9.29	0.034	2.65	1.87
	6	0.14	5.47	3.37	0.044	20.70	9.84	0.032	3.35	2.23
	8	0.13	6.27	4.02	0.039	21.05	9.71	0.034	3.69	2.25
	12	0.13	7.35	4.79	0.054	22.08	10.61	0.034	4.51	2.48
2.3	2	0.13	5.43	3.67	0.055	22.65	12.23	0.025	3.07	2.60
	4	0.14	6.21	4.25	0.056	23.36	12.20	0.028	3.61	2.84
	6	0.14	6.73	5.49	0.060	24.09	12.89	0.026	4.33	3.23
	8	0.13	7.51	5.05	0.049	24.62	13.29	0.025	4.77	3.37
	12	0.13	8.90	6.18	0.047	25.42	13.64	0.027	5.60	3.59
2.5	2	0.13	6.42	4.53	0.057	25.76	14.70	0.018	4.07	3.59
	4	0.15	7.43	5.26	0.058	26.13	15.06	0.028	4.88	4.09
	6	0.17	8.14	5.87	0.057	26.85	15.63	0.027	5.45	4.33
	8	0.14	8.71	6.21	0.043	27.50	15.76	0.036	5.87	4.46
	12	0.14	10.01	7.25	0.057	28.79	16.59	0.028	6.70	4.70

Table 1: Results of testing DynCOAA, DynAWC and DynDBA on 30-node, dynamic 3-coloring problems. Averages over 20 problem instances per Density- δ combination. Time is expressed in seconds.

cal search algorithm to find a solution. Because no initial schedule was found, DynDBA could not even begin to search for a solution after the change.

The two algorithms that did find a valid solution did so in all 20 runs. But the behaviors of DynCOAA and DynAWC were completely different for the original and the changed problem. For the original schedule, it took both algorithms about 1 minute to find a first solution for all 20 runs. The average solution of DynCOAA was twice as good as the average solution of DynAWC at that time. The average solution of DynAWC quickly became better and after 10 minutes DynAWC had found an almost perfect solution in all 20 runs, while DynCOAA still had a cost of about 550.

As for the schedule after the storm, it took DynCOAA less than 10 seconds to find a valid schedule in all 20 runs. Even more, the first average solution was almost as good or even better than the last average solution before the storm. DynCOAA could further improve this first schedule to end at an average cost of about 300 after 10 minutes of calculation. DynAWC on the other hand took almost 100 seconds to find a first solution in all 20 runs. The first average solution had a cost of about 5000. In the remaining time, the solution could be improved to less than 2000, but this still was a lot higher than the average solution of DynCOAA.

One of the reasons of the slower response rate of DynAWC is the preprocessing of the nogoods. After 10 minutes of calculation for the initial schedule, more than 10.000 nogoods had been accumulated. It took DynAWC about 10 seconds to filter out the nogoods that had become invalid after the change. By that time, DynCOAA had already found a new solution.

3.3 Evaluation

The results of DynDBA are the most dissimilar when we look at the behavior over the two different categories of tests. For the artificial graph coloring problems, it was the clear winner of the three algorithms. For the real-world ship scheduling problems, it could not find any solution.

DynCOAA and DynAWC on the other hand behave rather similar for the two different categories of problems. For the first, static problem, DynAWC is a better choice. But when the static problem becomes a dynamic problem by changing some constraints, DynCOAA reacts faster to those changes and finds a better solution.

4. CONCLUSIONS AND FUTURE WORK

In this paper we presented DynCOAA. It is based on the ACO meta-heuristic and designed from the start for solving dynamic constraint optimization problems (DynCOPs). We compared this algorithm against two other algorithms in two different categories of problems: artificial graph coloring problems and real-world ship scheduling problems.

There are two important conclusions that can be drawn from this comparison: (1) DynCOAA is suited for solving DynCOPs, as it beats DynAWC in all two and DynDBA in one of the two categories of problems, and (2) Choosing the right algorithm for a specific problem is very important, because the performance of algorithms can depend greatly on the type of problem.

This work has been funded by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen)

5. REFERENCES

- [1] M. Dorigo, V. Maniezzo, and A. Colomi. Positive Feedback as a Search Strategy, Technical Report 91016, Dipartimento di Elettronica e Informatica, Politecnico di Milano, Italy, 1991.
- [2] T. Engineering. <http://www.engineering.tractebel.com/>.
- [3] R. Mailler. Comparing two approaches to dynamic, distributed constraint satisfaction. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '05)*, pages 1049–1056, Utrecht, The Netherlands, July 25–29 2005.
- [4] K. Mertens and T. Holvoet. CSAA; a Constraint Satisfaction Ant Algorithm Framework. In *Proceedings of the Sixth International Conference on Adaptive Computing in Design and Manufacture (ACDM'04)*, pages 285–294. Springer-Verlag, 2004.
- [5] K. Mertens and T. Holvoet. CSAA; a Distributed Ant Algorithm Framework for Constraint Satisfaction. In *Proceedings of the 17th International FLAIRS Conference*, pages 764–769. AAAI Press, 2004.
- [6] T. Schiex and G. Verfaillie. Nogood Recording for Static and Dynamic Constraint Satisfaction Problem. *International Journal of Artificial Intelligence Tools*, 3(2):187–207, 1994.