Which Dynamic Constraint Problems Can Be Solved By Ants?

Koenraad Mertens* and **Tom Holvoet** and **Yolande Berbers**
AgentWise, Distrinet
Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A
B-3001 Leuven, Belgium
{Koenraad.Mertens, Tom.Holvoet, Yolande.Berbers}@cs.kuleuven.be

Abstract

There exist a number of algorithms that can solve dynamic constraint satisfaction/optimization problems (DynCSPs or DynCOPs). Because of the large variety in the characteristics of DynCSPs and DynCOPs, not all algorithms perform equally well on all problems.

In this paper, we present the Dynamic Constraint Optimization Ant Algorithm (DynCOAA). It is based upon the ant colony optimization (ACO) meta-heuristic that has already proven its merit in other dynamic optimization problems. We perform a large number of experiments to identify the dynamic constraint problems which our algorithm is most suited for. It turns out that this is a large class of problems, namely heterogeneous problems that change often. We find this to be common characteristics in real-world applications. For these problems, DynCOAA outperforms both the complete and non-complete traditional algorithms that were used for comparison.

Introduction

Numerous real-world problems 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 but a constantly changing problem that has to be solved. 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 (DynCSPs/DynCOPs). They were first introduced by Dechter and Dechter in (Dechter & Dechter 1988).

A number of algorithms have been proposed for solving dynamic constraint problems. They can be divided into traditional methods and methods based on meta-heuristics. Traditional solving methods include both complete and non-complete algorithms. They are mostly adaptations of existing algorithms for static problems. While the complete methods perform better on harder static problems, the more reactive nature of the incomplete methods makes them more suited for easier static problems and for tracking changes in dynamic problems (Davenport 1995; Mailler 2005).

Meta-heuristic methods include evolutionary algorithms and ant colony optimization (ACO) algorithms. In (Craenen, Eiben, & van Hemert 2003), it is shown that evolutionary algorithms cannot compete with complete algorithms on static problems, while (van Hemert & Solnon 2004) and (Mertens & Holvoet 2004) show that ACO algorithms are able to compete with complete methods on static problems.

In this paper, we present the Dynamic Constraint Optimization Ant Algorithm (DynCOAA), which is specifically designed to solve DynCOPs. We perform a large number of experiments on a broad range of DynCOPs, in order to identify those problems where DynCOAA achieves the best results. We compare those results to the performance of two traditional algorithms: the complete algorithm Dynamic Nogood Recording (DynNR) (Schiex & Verfaillie 1993) and the non-complete Dynamic Breakout Algorithm (DynBA) (Morris 1993; Mailler 2005). It turns out that there is a huge difference in the performance of DynCOAA with respect to the type of problem that has to be solved. For homogeneous problems (those where no distinction can be made between variables), DynCOAA is not the most efficient algorithm, while for heterogeneous problems (those that do exhibit such a distinction) that change rapidly, DynCOAA is the best choice.

In the following section we present our Dynamic Constraint Optimization Ant Algorithm (DynCOAA). This is followed by the tests we performed and the results we found. We conclude this paper with an overview of the relevance of our findings in real-world problems.

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

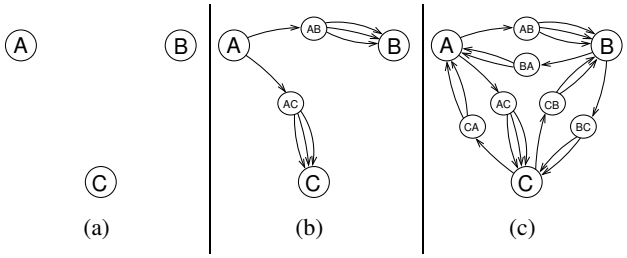


Figure 1: Construction of a graph for a problem with 3 variables: $A \in \{4, 5, 6\}$; $B \in \{2, 3\}$; $C \in \{2, 3\}$ and 2 constraints: $A = B+C$; $B > C$. (a) Each main node represents a variable. (b) Going to the next main node is a two-step process: first the next variable is chosen, then the value for the current variable (A has 3 possible values: there are 3 value edges from AB to B). (c) The complete graph for the problem. For reasons of efficiency, a reference to $A = B + C$ is stored in A, B and C, and a reference to $B > C$ in B and C, although one reference to each constraint suffices when privacy concerns come into play in a distributed setting.

Algorithms

In this section we describe the algorithms we use in our experiments. We first detail the mechanism behind our Dynamic Constraint Optimization Ant Algorithm (DynCOAA). Next we state the main characteristics of Dynamic Nogood Recording (DynNR) and the Dynamic Breakout Algorithm (DynBA).

Dynamic Constraint Optimization Ant Algorithm

The basic idea of the ACO meta-heuristic is that agents give positive feedback to other agents when they have found a solution. The feedback is given indirectly by placing artificial *pheromones* in the environment of the agents. The amplitude of the feedback depends on the quality of the solution.

In order to use the ACO meta-heuristic, the problem that has to be solved must be translated into an environmental structure. In the case of solving a DynCOP with the DynCOAA algorithm, this structure is a graph, where the nodes represent variables and the edges represent values for those variables. Each node stores references to the constraints its associated variable is involved in. Additional nodes and edges are inserted to allow a more differentiated feedback, such as an approximation of the fail-first heuristic. The construction of such a graph for a simple problem with three variables is shown in Fig. 1.

In its simplest form, an agent that tries to solve a COP "walks around" in the graph structure. At each node, an agent has to decide which edge to follow next. This decision is based upon the amount of pheromones that are present on each edge (feedback from previous agents). Moreover, when the cost function is composed of the sum of weights of violated constraints, each agent can check the constraints before choosing a value and thus avoids bad solutions at an early stage (as is done in a tree search). The relative importance of the feedback on the one hand and the cost function on the other hand can be tuned for each class of problems. It is also possible to adapt the relative importances of the cost function and the feedback while solving: as more agents provide feedback, the usefull information that is provided by that feedback increases. In any case, decisions are made stochastically to prevent that after some time all agents compose the same variable-value assignment.

While going from node to node, an agent remembers the path it has followed (and thus the values it has assigned to variables). When an agent has visited all nodes, it has constructed a complete variable assignment and the complete cost function can be evaluated. According to this cost, the agent puts more or less pheromones on the path it has walked.

Future agents that walk the graph are influenced by pheromones that are present in the graph, but because pheromones evaporate, their influence is limited in time. At the start of the algorithm, all edges are initialized with a default pheromone quantity. As long as no agent puts pheromones on a specific edge – because no agent finds a good solution using the value associated with that edge – the pheromone quantity of that edge gradually decreases. This means that the probability that the edge gets chosen by an agent also decreases.

For reasons of efficiency, not a single agent, but a swarm of agents walk the graph simultaneously. Only the agent that has found the best solution is allowed to put pheromones on the graph. The agent that has found the best overall solution is also remembered. The pheromones on the path of this best agent are increased together with that of the best agent of each swarm. Additional details on the construction of the graph and the behavior of the ants can be found in (Mertens & Holvoet 2004).

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, Guntsch et. al. show in (Guntsch, Middendorf, & Schmeck 2001) that 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.

In (van Hemert & Solnon 2004), it is shown that ACO algorithms can compete with complete algorithms when solving static optimization problems. When we look at the general properties of ACO algorithms (Dorigo, Maniezzo, & Colorni 1991; Guntsch, Middendorf, & Schmeck 2001), they should be even better suited for dynamic optimization problems.

Algorithms for Comparison

We use two existing algorithms for comparing the efficiency of the DynCOAA algorithm. Dynamic Nogood Recording

(DynNR) (Schiex & Verfaillie 1993) is a complete algorithm, based on nogood recording, which was introduced by (Dechter 1990). The nogoods that are recorded during a search, are filtered and those nogoods that are still valid can be reused in the next search. Nogoods remain valid as long as the constraints that they are deduced from do not weaken. We included DynNR in our comparison because (Schiex & Verfaillie 1993) states that this algorithm is the best complete method for dynamic constraint solving.

The Dynamic Breakout Algorithm (DynBA) is a non-complete, local search algorithm, that is able to escape from local minima. The (static) Breakout Algorithm was first introduced in (Morris 1993). According to (Mailler 2005), where the dynamic version was first proposed, DynBA is a good choice for rapidly changing, hard problems: those that DynCOAA is designed for.

Experimental Results

In this section we discuss the results of the tests we have done to compare DynCOAA with DynNR and DynBA. The three algorithms were implemented in C++, using the same constraint processing engine. All tests were performed on a Intel Pentium 4 CPU 3.0 GHz.

The problems that we tested were graph coloring problems where the number of constraint violations had to be minimized. In the first subsection, we describe the behavior of the three algorithms on homogeneous graph coloring problems. In the second subsection heterogeneous graph coloring problem are solved. The difference between homogeneous and heterogeneous problems is that in homogeneous problems, the constraints are chosen randomly, while in heterogeneous problems, the random constraint generator is guided, so that the variables become clustered. There exist more constraints between variables of the same cluster than there are constraints between variables of different clusters. As we will show, ant algorithms perform better on these types of problems. We end this section with the results of a real-world application.

Homogeneous Graph Coloring Problems

For our first series of tests, we used homogeneous graph coloring problems. In homogeneous problems, all variables are treated equally by the random constraint generator (the common practice for constructing graph coloring problems). For each problem, the algorithms could compute for x seconds. After those x seconds, three constraints were replaced by three different constraints and again the algorithms could compute for x seconds. This was repeated 100 times. For each combination of parameters, 20 different dynamic problems (with 1 initial and 100 changes) had to be solved. The initial problem as well as the changes were remembered, so that each algorithm had to solve exactly the same set of problems. For DynCOAA, a swarm of 5 ants was used.

There are a number of parameters in homogeneous graph coloring problems that can be varied: the number of variables, the number of values and the number of constraints. For static problems, these parameters determine where the problem is located in relation to the phase transition. For easy problems (those that are clearly under- or over-constrained), incomplete algorithms like BA or COAA can find a good solution fast, by following heuristics. Complete algorithms also find a good solution, but it can take longer. For hard problems (those within the phase transition between under- and over-constrained problems) the complete algorithms like NR have the advantage of only considering each assignment once. If only one or a few assignments are allowed by the constraints, a complete algorithm like NR will probably find it faster than an incomplete algorithm like BA or COAA.

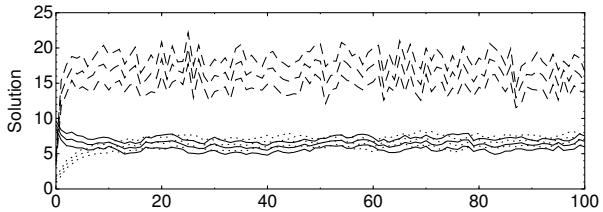
For dynamic problems, we notice a different behavior: whether or not the parameters place a problem in the phase transition zone is far less important when that problem changes constantly. For a changing problem, the rate of change becomes the dominant factor. This is consistent with the findings in (Mailler 2005). For slowly changing problems the DynNR algorithm displayed the best performance. This can be easily explained by intuition: a complete algorithm will always find the optimal solution when given enough time. When a problem changes slowly, the DynNR algorithms has enough time between changes to find (a good approximation to) the optimal solution. For rapidly changing problems the more reative DynCOAA and DynBA algorithms outperform DynNR.

The performance of DynBA and DynCOAA is almost the same on homogeneous dynamic graph coloring problems. Figure 2 shows the average best solution right before each change for a number of typical problems (thus each of the 101 data points is the average of 20 solution values). The three lines for each algorithm indicate the average and 95% confidence interval. Whether the algorithms are allowed to compute for 0.25 seconds or for 2 seconds on dynamic graph coloring problems that are within the phase transition, DynCOAA and DynBA reach almost the same solutions. Evidently, this solution is better when the algorithms are allowed to compute longer (figure 2(a) versus figure 2(b)). Whether the algorithms have to solve problems that are within the phase transition or problems that are over-constrained has no influence on the relative performance of both algorithms (figure 2(a) versus figure 2(c)).

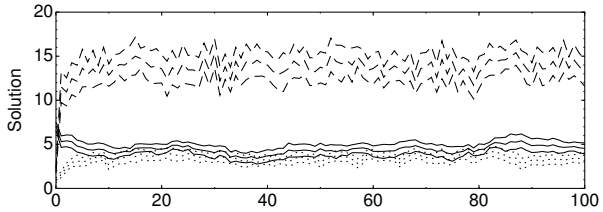
Heterogeneous Graph Coloring Problems

Heterogeneous graph coloring problems are graph coloring problems where not all variables are treated equally. An example of a heterogeneous optimization problem is the clustered traveling salesman problem (van Hemert & Urquhart 2004). In these problems, cities are clustered in certain regions of space. Optimal solutions are likely to visit all cities within one cluster, before visiting other clusters.

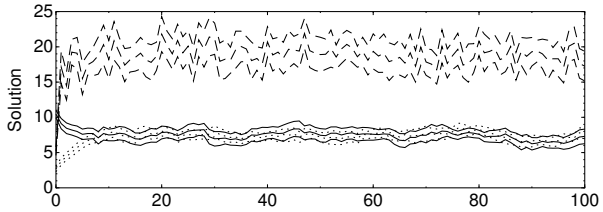
Two examples of heterogeneous graph coloring problems are shown in figure 3. Figure 3(a) shows a cluster structured graph coloring problem. Variables within a cluster are connected by more edges than variables in different clusters. Figure 3(b) shows a star structured graph coloring problem. In these problems, all inter-cluster edges connect a variable from one central cluster to a variable of another cluster. For our tests, we used clusters of 30 variables (resulting in total problems of 30, 60, 90,... variables), with inside each



(a) Degree 2.3, 0.25 seconds



(b) Degree 2.3, 2 seconds

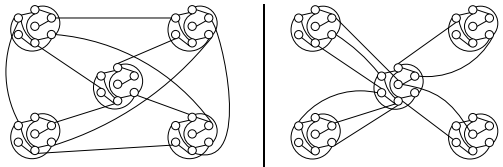


(c) Degree 2.5, 0.25 seconds

Figure 2: Average best solution right before each change for DynCOAA (solid line), DynBA (dotted line) and DynNR (dashed line) on homogeneous dynamic graph coloring problems with 30 variables, 3 colors, a degree of 2.3 (for (a) and (b)) or 2.5 (for (c)) and a computing time of 0.25 seconds (for (a) and (c)) or 2 seconds (for (b)).

cluster an edge degree of 2.3. On average, each variable was involved in 2 inter-cluster edges. As with the homogeneous problems, for each combination of parameters, 20 different problems, consisting of 1 initial problem and 100 successive problems were generated.

When we look at the results for relatively small (90 variables, thus 3 clusters) problems (figure 4), we notice that both DynCOAA and DynBA perform better that DynNR. This is due to the relatively short computing time of 2 seconds for each change. If we would increase the computation time, the difference would become smaller or even be inverted. For problems with a cluster structure (figure 4(a)), the performance of DynBA is slightly better that that of DynCOAA. At least, this holds after about 20 changes. During the first 20 changes however, the performance of DynBA is clearly better than that of DynCOAA. This is due to the pheromone trails in DynCOAA. When agents give feedback, they do so by laying pheromone trails in their environment. Because of the special construction of the environment in DynCOAA, the pheromone trails can approximate the fail-first heuristic, which is very helpfull in structured problems. But it takes time for the ants to gather these pheromone trails. During this time, the information that is present in the



(a) Cluster structure (b) Star structure

Figure 3: Examples of a cluster structured and a star structured graph coloring problem. Each cluster contains 30 variables in the problems that were tested. Internally, the clusters have an edge density of 2.3 (this is within the phase transition for 30 variables). For cluster structured problems (a), there are extra edges that connect variables from different clusters. For star structured problems (b), each extra edge connects a variable from the central cluster to a variable from another cluster. On average, each variable is involved in two inter-cluster edges.

trails continuously increases. The solution that is found also gradually enhances. After about 20 changes, the pheromone trails reach the maximum amount of information that is possible at the given change rate and the solution stagnates. For problems with a star structure (figure 4(b)), the same conclusions can be drawn. But because a star structured graph has an even more explicit structure, the information that can be captured in the pheromone trails is even more meaningful.

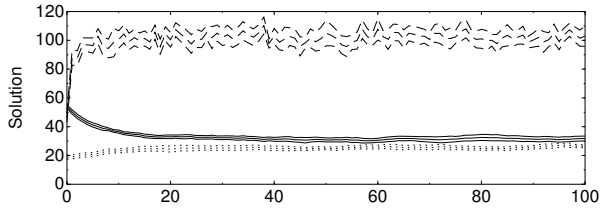
It is also clear from figure 4(b) that the nogood recording strategy of DynNR has an important drawback when rapidly changing problems have to be solved. During the first changes, the solution that is found by DynNR is better than that of DynCOAA. But while DynCOAA increases its amount of information, without the need of filtering it after each change, DynNR does need to filter the nogood it has collected. This takes time, and limits the amount of time that remain available for searching a solution. After about 10 changes, an equilibrium is found between filtering the nogoods and creating new ones.

These conclusions can, to a lesser extend, also been applied to the homogeneous problems the we discussed earlier.

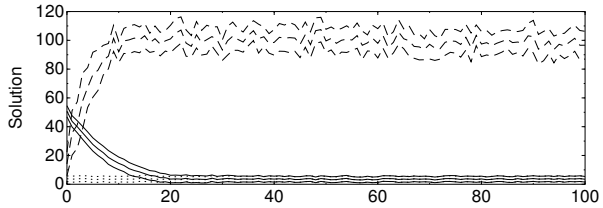
When we look at bigger problems (figure 5: 150 variables, 5 clusters and figure 6: 210 variables, 7 clusters), we notice the same global picture. But the bigger the problem (and thus the bigger the graph structure that is used by the ants in DynCOAA), the more information that can be stored by DynCOAA. For problems of 210 variables, extra information can be stored in the pheromone trails even after 80 changes. This results in a performance for DynCOAA that is clearly better than that of DynBA.

Real-World Ship Scheduling Test

While it is interesting to know that DynCOAA is suited for heterogeneous dynamic constraint problems, the question remains if this class of problems appears in real-world applications. Therefore, we decided to also use a real-world problem instead of an artificial problem to validate our results. The problem we try to solve is to assemble a schedule for transport ships. The ships transport liquified natural gas

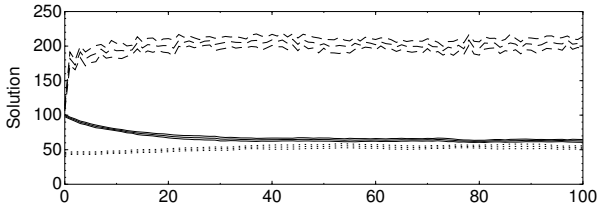


(a) Cluster structured, 90 variables

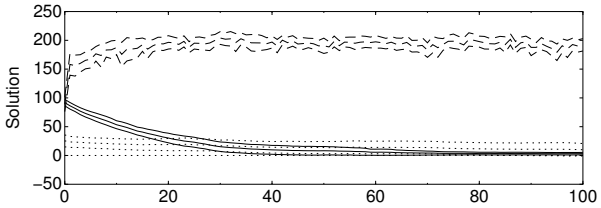


(b) Star structured, 90 variables

Figure 4: Average best solution right before each change for DynCOAA (solid line), DynBA (dotted line) and DynNR (dashed line) on heterogeneous dynamic graph coloring problems with 90 variables (3 clusters) in a cluster structure (a) or a star structure (b). The computation time per problem was 2 seconds.



(a) Cluster structured, 150 variables



(b) Star structured, 150 variables

Figure 5: Average best solution right before each change for DynCOAA (solid line), DynBA (dotted line) and DynNR (dashed line) on heterogeneous dynamic graph coloring problems with 150 variables (5 clusters) in a cluster structure (a) or a star structure (b). The computation time per problem was 2 seconds.

from different ports around the world to one destination port. Using data provided by Tractebel Engineering (Engineering), 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, with domains up to 200 possible values, and 101 constraints. The constraints are a mixture of hard constraints (those have to be obeyed in each valid solution: it are constraints with a weight of infinity) and soft constraints, binary constraints and n-ary constraints. 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. An important aspect of this problem is the physical meaning of the variables: there are variables that represent a port, others represent a time or a cargo. Not all these variables have a similar meaning, resulting in a heterogeneous problem.

Instead of the three algorithms that were described in the previous sections, we used the distributed versions of those algorithms (DynDBA and DynAWC) for this problem. Because the principles behind the algorithms remain the same, this should not influence the results, only the speed of calculation. The swarm size for DynCOAA was increased from 5 to 30. Each of the three algorithms made 20 schedules. The anytime curves for this problem are shown in figure 7.

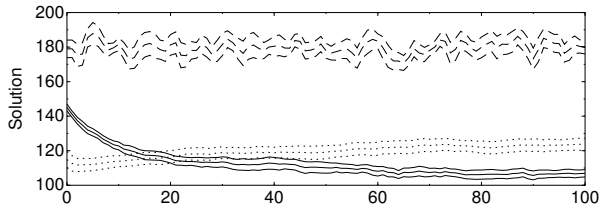
The first thing to notice is that DynDBA is missing from figure 7. During the 10 minutes of calculation time, none of the 20 runs were able to find any valid schedule (one that satisfies all hard constraints). Because no initial schedule was found, DynDBA could not even begin to search for a solution 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. While the initial solution of DynCOAA was better, the solution that DynAWC found after 10 minutes was almost perfect. At that time, DynCOAA still had a cost of about 550.

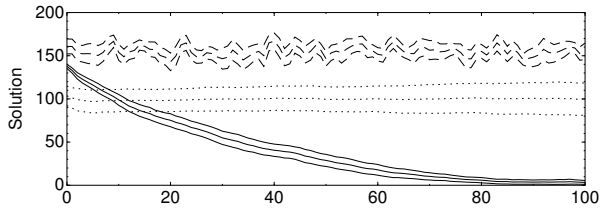
As for the schedule after the storm, it took DynCOAA less than 10 seconds to find a valid schedule in all 20 runs. DynCOAA could not improve this first schedule significantly, but still ended with an average cost of about 300 after 10 minutes of calculation. DynAWC on the other hand took almost 100 seconds 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.

Conclusions

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 to two traditional algorithms: the complete algorithm DynNR and the incomplete algorithm DynBA.



(a) Cluster structured, 210 variables



(b) Star structured, 210 variables

Figure 6: Average best solution right before each change for DynCOAA (solid line), DynBA (dotted line) and DynNR (dashed line) on heterogeneous dynamic graph coloring problems with 210 variables (7 clusters) in a cluster structure (a) or a star structure (b). The computation time per problem was 2 seconds.

From our tests, we can conclude that DynCOAA is suited for heterogeneous, rapidly changing dynamic constraint problems. When other types of problems have to be solved, other algorithms are more suited. As can be seen from the real-world test we did, a heterogeneous problem is not a rare phenomenon. Instead we believe it is present in a lot of real-world applications.

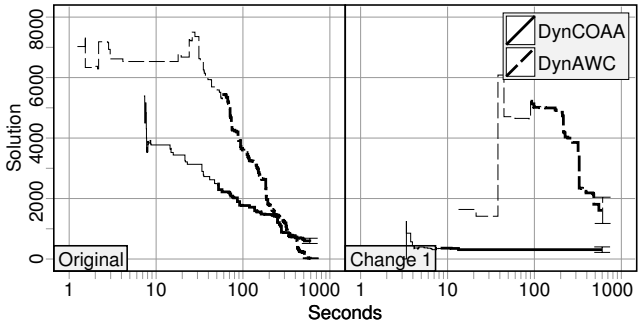


Figure 7: Anytime curves for DynCOAA (solid line) and DynAWC (dashed line) when assembling a schedule for ships. DynDBA was unable to find any solution. The error bars indicate the 95% confidence interval of the final average solution.

References

Craenen, B. G. W.; Eiben, A. E.; and van Hemert, J. I. 2003. Comparing evolutionary algorithms on binary constraint satisfaction problems. *IEEE Trans. Evolutionary Computation* 7(5):424–444.

Davenport, A. 1995. A comparison of complete and incomplete algorithms in the easy and hard regions. In *Proceedings of the Workshop on Studying and Solving Really Hard Problems, CP-95*, 43–51.

Dechter, R., and Dechter, A. 1988. Belief maintenance in dynamic constraint networks. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI '88)*, 37–42.

Dechter, R. 1990. Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition. *Artif. Intell.* 41(3):273–312.

Dorigo, M.; Maniezzo, V.; and Colorni, A. 1991. Positive Feedback as a Search Strategy, Technical Report 91016, Dipartimento di Elettronica e Informatica, Politecnico di Milano, Italy.

Engineering, T. http://www.engineering.tractebel.com/.

Guntsch, M.; Middendorf, M.; and Schmeck, H. 2001. An ant colony optimization approach to dynamic TSP. In Spector, L.; Goodman, E. D.; Wu, A.; Langdon, W. B.; Voigt, H.-M.; Gen, M.; Sen, S.; Dorigo, M.; Pezeshk, S.; Garzon, M. H.; and Burke, E., eds., *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 860–867. San Francisco, California, USA: Morgan Kaufmann.

Mailler, R. 2005. Comparing two approaches to dynamic, distributed constraint satisfaction. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '05)*, 1049–1056.

Mertens, K., and Holvoet, T. 2004. CSAA; a Constraint Satisfaction Ant Algorithm Framework. In *Proceedings of the Sixth International Conference on Adaptive Computing in Design and Manufacture (ACDM'04)*, 285–294. Springer-Verlag.

Morris, P. 1993. The breakout method for escaping from local minima. In *AAAI*, 40–45.

Schiex, T., and Verfaillie, G. 1993. Two approaches to the solution maintenance problem in dynamic constraint satisfaction problems. In *Proceedings of the IJCAI-93/SIGMAN Workshop on Knowledge-based Production Planning, Scheduling and Control, Chambery, France*.

van Hemert, J., and Solnon, C. 2004. A study into ant colony optimization, evolutionary computation and constraint programming on binary constraint satisfaction problems. In *Applications of evolutionary computing (EvoCOP 2004)*, Lecture Notes in Computer Science. Springer Verlag.

van Hemert, J. I., and Urquhart, N. 2004. Phase transition properties of clustered travelling salesman problem instances generated with evolutionary computation. In Yao, X.; Burke, E. K.; Lozano, J. A.; Smith, J.; Guervós, J. J. M.; Bullinaria, J. A.; Rowe, J. E.; Tiño, P.; Kabán, A.; and Schwefel, H.-P., eds., *Parallel Problem Solving from Nature - PPSN VIII, 8th International Conference, Birmingham, UK, September 18-22, 2004, Proceedings*, volume 3242 of *Lecture Notes in Computer Science*, 151–160. Springer.