CSAA: A Distributed Ant Algorithm Framework for Constraint Satisfaction

Koenraad Mertens and **Tom Holvoet**
KULeuven Department of Computer Science
Celestijnenlaan 200A
B-3001 Leuven, Belgium
koenraad.mertens, tom.holvoet@cs.kuleuven.ac.be

Abstract

In this paper the distributed Constraint Satisfaction Ant Algorithm (CSAA) framework is presented. It uses an ant-based system for the distributed solving of constraint satisfaction problems (CSPs) and partial constraint satisfaction problems (PCSPs). Altough ant algorithms have already proven to be a viable alternative to classical approaches for a variety of optimization and combinatorial problems, current ant systems work in a centralized manner. Problems where the flexibility of ant systems can be useful, usually tend to get large. Therefore a distributed solving approach is needed. We show that when the distribution is done in an appropriate manner, ant algorithms conserve their flexibility. The distribution however is not trivial. A number of difficulties (especially with relation to speed and accuracy) emerge when the centralized framework would just be distributed over multiple hosts. In this paper we address those difficulties and provide solutions. We show that with the right design decisions, a distributed ant algorithm is a viable alternative for classical approaches. When flexibility in the solving method becomes an issue (for example in dynamic problems), ant algorithms, who use an flexible decision mechanism without hard commitments, even have an advantage over traditional algorithms.

Ant Algorithms

An Ant-based System can be thought of as a special kind of multi-agent system, where each agent is modelled after a biological ant (therefore, agents are also called *ants* in ant-based systems). Each ant-based system consists of an *environment* and a number of agents. The environment is the topology of the system, the structure wherein agents are situated.

Agent (ants) can move around in their environment and manipulate the objects that are placed inside it. Moving around (*walking*) allows the agents to find (an approximation for) the solution to the problem the system is used to solve. The agents are not able to communicate directly with each other, but they put objects (named *pheromones*, after the biological chemicals) in the environment, which can be observed by other agents. Pheromones *evaporate*, thereby limiting their influence over time. In ant-based systems, a large number of agents are placed in the environment simultaneously (a *swarm* of agents), each dropping a small amount of pheromones, but enough to influence other agents when a number of similar pheromones are dropped at the same locations. These influences are exploited while the agents walk in the environment. At first, each agent walks around randomly, but the agents who find the best approximations to the solution, drop pheromones at the paths they walked on. In a next step, other agents (slightly) prefer these paths. Stochastic processes and evaporation of pheromones allow for further exploration of the environment, but in a guided manner. This way, the agents continuously try to improve the currently best solution, in a manner similar to local search.

In general, ant-based systems tend to perform well on a number of optimization problems (e.g. the Travelling Salesman Problem (Dorigo, Maniezzo, & Colorni 1991)). In (Mertens & Holvoet 2004), we presented the *Constraint Satisfaction Ant Algorithm* (CSAA) framework: an ant-based system using a graph structure for the environment that is able to solve Constraint Satisfaction Problems (CSPs) and Partial Constraint Satisfaction Problems (PCSPs). The framework incorporates a number of heuristics that are widely used in traditional solving methods. These heuristics increase the efficiency of the ant algorithm and allow to reach an initial solution fast. This paper describes the distribution of the CSAA framework.

Our ultimate goal is to solve dynamic constraint satisfaction problems (DCSP). These are problems where the problem instance itself changes while it is being solved (addition/removal of variables, values and constraints). Therefore, maintaining the flexibility of ant-based systems was a very important aspect for the system. Flexibility is this context means that the system should be able to react to changes quickly. In other words, an initial solution should not take a long time to compute and no hard commitments (like no-goods) should be made: when the problem changes, the relevance of those commitments cannot be predicted, and we want to avoid checking them all as this takes a lot of time. Evaporating pheromones are very efficient to avoid hard commitments: they suggest a certain direction to the agents, but changes in the problem instance are dealt with a transparent manner.

In the next section we recall the most important design decisions of the CSAA framework. It is followed by the main section of this paper, dedicated to the distribution of

Copyright © 2004, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

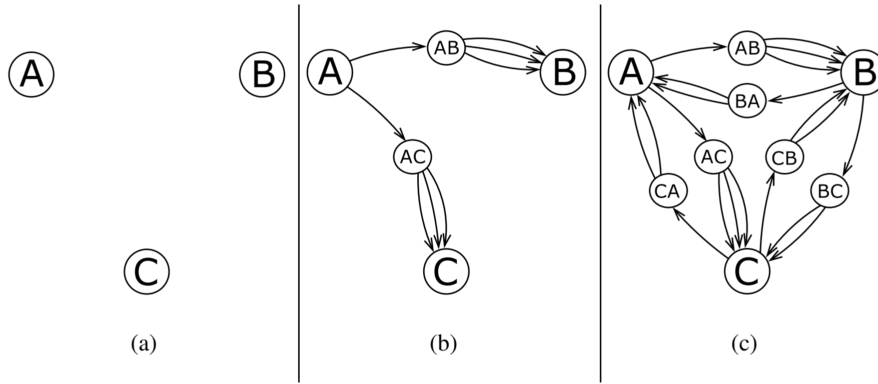

Figure 1: Construction of a graph for a problem with 3 variables: $A \in \{4,5,6\}$; $B \in \{2,3\}$; $C \in \{2,3\}$ and 2 constraints: $A = B + C$; $B > C$ (a) Each main node represents a variable. (b) Going to the next main node is a two-step process: first the selection edge is chosen, then the value for the current variable (A has 3 possible values, thus there are 3 value edges from AB). (c) The complete graph for the problem.

the algorithm. The conclusion and future work are described in the last section.

Principles of the Constraint Satisfaction Ant Algorithm (CSAA)

This section recalls briefly the design decisions of our algorithm. It addresses first the environment of the algorithm, followed by the algorithm itself. Full details about the algorithm can be found in (Mertens & Holvoet 2004).

Environment

Figure 1 shows the construction of a CSAA graph for a problem with three variables: $A \in \{4,5,6\}$, $B \in \{2,3\}$ and $C \in \{2,3\}$. The problem has two constraints: $A = B + C$ and $B > C$. The graph consists of main nodes, selection nodes, selection edges and value edges. Each main node represents a variable. The number of main nodes is the same as the number of variables. This is depicted in Figure 1(a). In Figure 1(b), we see that each main node is connected through a number of selection edges to selection nodes. The selection edges determine the ordering of the variables: from a selection node, only one main node can be reached. In a problem with N variables, every main node has $N - 1$ selection nodes. The selection nodes are connected to the next main node through value edges. Each value edge is associated with one possible value for the variable that is associated with the previous main node. The constraints are stored in the main nodes: a reference to $A = B + C$ is stored in nodes A, B, and C; a reference to $B > C$ is stored in nodes B and C.

Ant Algorithm

The algorithm uses a swarm of agents. The size of the swarm depends on the size of the problem and the computational power of the host. Each agent is placed at a randomly chosen main node of the graph. The agent walks the graph, thereby remembering the path it walked (and thus the partial solution it has created). The agents are also responsible for checking the constraints. Pheromones are only updated after each agent of the swarm has finished. Only the best agents (those that found the best solutions) and agents that failed are allowed to drop pheromones. We first give a detailed overview of the agent behavior in the CSP case. Differences for the PCSP case are described afterwards.

When an agent arrives in a main node (suppose: main node A in Figure 1), it first chooses which selection edge to take next (suppose: the edge that leads to selection node AB). From the selection node, there are a number of value edges, each leading to the same main node (B). The number of edges is equal to the number of values in the domain of the previous main node (A). Each value edge is associated with one of those values (so there are three value edges between AB and B, because the domain of A consists of three values: $A \in \{4,5,6\}$). Thus, choosing a value edge means choosing a value for the variable, associated with the previous main node. Only value edges whose values do not create a conflict with the values that have already been assigned can be allowed. This implies that the agent has to check the appropriate constraints (those that involve the present variable A) at this point. If no value exists that does not violate any constraints, no solution can be found with the currently chosen values: the variable associated with the previous main node *fails*. When all variables can be assigned a value, the agent has constructed a solution by combining all collected values from the value edges it walked upon. No pheromones are dropped in this case: the algorithm has ended.

When a variable fails, the agent returns to all edges it walked upon in order to drop *pheromones* on them. Pheromones are used to influence choices: positive pheromones that are placed on an edge increase the probabilities of that edge to be chosen by an agent, negative pheromones decrease those probabilities. The use of pheromones, in combination with the structure of the graph, mimics the behavior of aspects of traditional algorithms:

- Arc consistency: because the pheromone values on the edges (and thus the experiences of previous agents) are taken into account when the next node and the value are

chosen, arc consistency is mimicked.

- Min-conflict (Minton *et al.* 1992): the pheromones on value edges indicate whether a value is likely to be included in a solution or not. This way, "bad" values are avoided, minimizing the risk of conflicts with future values.
- First-fail: the pheromones on selection edges indicate if a variable is likely to cause conflicts. This way, "difficult" variables can be addressed quickly.

Pheromones *evaporate*. This limits their influence over time, allowing for a more flexible adaptation to changes, and a weaker commitment to wrong assumptions (e.g. an invalid assumption about the likeliness of a value). Pheromones due to correct assumptions are enhanced by future pheromone droppings, pheromones due to wrong assumptions are not and evaporate. Mathematic formulas of how fast pheromones should evaporate and how much importance an agent should assign to it, can be found in (Rauch, Millonas, & Chialvo 1995; Chialvo & Millonas 1995; Dorigo, Bonabeau, & Theraulaz 2000).

There are some differences for the PCSP case. When solving a PCSP, it is assumed that no solution can be found which satisfies all constraints. Instead of searching for such a solution, variable-assignments that satisfy part of the constraints are allowed. An ordering between these "solutions" has to be made. Our framework is well suited for a special case of such an ordering, one that assigns priorities to constraints. We distinguish between *hard* constraints (with an infinite priority) and *soft* constraints (with a limited priority). While walking the graph, agents must satisfy hard constraints. Violations of soft constraints can be allowed. This means a variable can only *fail* when no value can be found that does not violate any hard constraints. The order of a solution is determined by the sum of the priorities of the violated soft constraints. In the PCSP case, agents that found the best solutions drop pheromones.

When solving a PCSP, an extra heuristic can be incorporated: the hill-climbing heuristic. When using hill-climbing, an agent chooses the value that violates "the least" constraints. In our stochastic ant-based system, we increase the probability that an agent chooses a value that violates few constraints. At the beginning of the algorithm, the pheromone trails do not contain much information, so the impact of the hill-climbing probabilities is large compared to those of the pheromones. The longer the algorithm is working, the larger becomes the impact of the pheromones.

Conclusion

As stated in (Mertens & Holvoet 2004), the CSAA framework reaches its goals: it is able to find a first solution rather quickly and the first solution as well as the final solution violate less constraints than other algorithms that are also designed for flexibility.

A Distributed Framework

Some constraint problems are not suited to be solved by one host, e.g. when a problem is too big or when some constraints can not be generally known for security reasons. In such cases, a distributed solution for solving them has to be used. In this section, we elaborate on the distribution of the CSAA framework. First we discuss the layout of the distributed framework. The second subsection gives details about the pheromone dropping process: how does it work, what are the problems that are associated with it and how can they be solved. The following subsection details about the security concerns. Thereafter comes a subsection with a number of additional optimizations. These include performance issues that are not yet implemented in the framework. We conclude this section with experimental results of the distributed framework.

Layout of the Distributed Environment

The easiest way to distribute the centralized version of the CSAA, is to place each node of the graph on a different host (this can also be a virtual host, with one physical host containing multiple virtual hosts). For problems with many variables, this would require a lot of (virtual) hosts. In addition, communication between different physical hosts is more expensive than internal communication in a host and should be avoided as much as possible. Therefore, we developed a distributed version of the CSAA framework, optimized for flexibility and bandwidth.

On each host, a centralized version of the CSAA is running, with only those variables that were assigned to the host. When a constraint involves variables that are on different hosts, the constraint is kept on all those hosts. Figure 2 shows a layout with nine variables and three hosts: each host is responsible for three variables. A swarm of agents is started on each host. The agents in the swarm search for a solution for all variables on their host (e.g. variables A, B and C on $Host\ I$). All agents that find a solution for those variables, are transferred to the next host (and placed in a queue on that host). Agents that can not find a solution (because all possible values of a variable have hard conflicts with values of other variables) do not get transferred to the next host, but drop pheromones on the edges they walked upon after which they are terminated. Below, we explain how the next host should be chosen. After transferring the agents, a new swarm is started. Therefore, the agents that are present in the queue of the host are used. If the number of agents in the queue is not enough for composing the swarm, a number of newly created agents is added. An agent that has already found a solution for variables on a previous host, takes the values it found for those variables into account when searching for a solution for the variables on its present host.

There are two ways of choosing the next host. The first is to simply number all hosts and to visit them in order. This is depicted in Figure 2. The second way is to implement some sort of "inter-host" pheromones and to let each agent decide for itself (based upon those pheromones) which host to visit next. Using pheromones has the advantage of a more accurate first-fail imitation. A disadvantage is that because of this first-fail principle, all pheromone trails tend to go the the host with the most constrained variables, causing a bottleneck at that host.

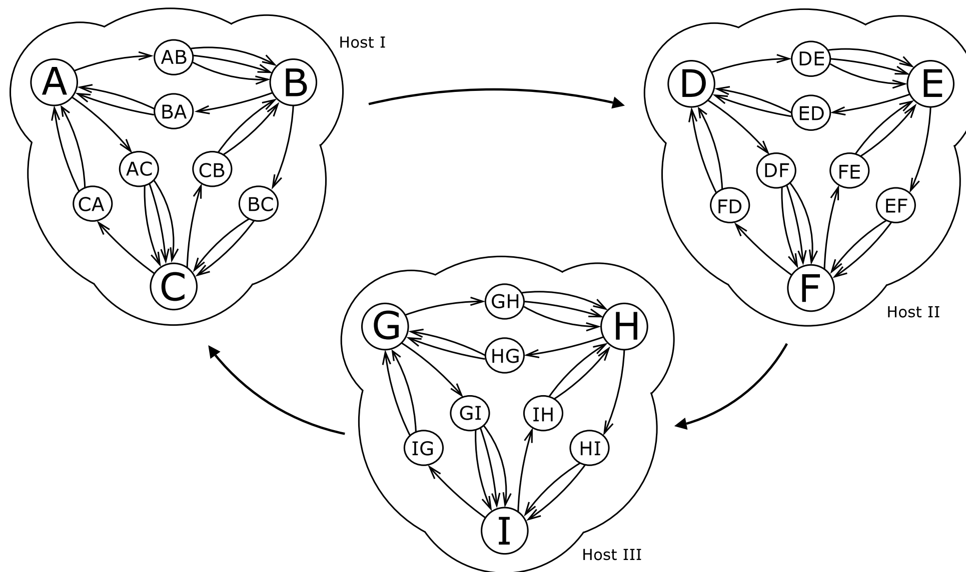

Figure 2: Distributed graph for the CSAA framework

Feedback with Pheromones

In the centralized version of the CSAA framework, an agent drops pheromones on the edges it walked when it fails (because of hard constraints) or, when solving a PCSP, after finding a solution that still violates a number of (soft) constraints. Doing the same thing in the distributed version of the framework would introduce a serious bandwidth issue: all agents that had visited more than one host would have to be retransferred to their previous hosts, causing the used bandwidth to be doubled (even worse, because the agents have acquired more information than when they were first transferred). Using a swarm of agents is somewhat better, because only the best agents drop pheromones and have to be transferred. Nevertheless, it is something we would like to avoid if there is an alternative. Only dropping pheromones on the last host the agent was on would mean a performance penalty: less feedback is provided for the future agents, causing the algorithm to slow down or, even worse, causing it to find an inferior solution. Therefore, we developed an intermediate solution. The best agents of the swarm drop pheromones before being transferred. This way, they do not have to return after they fail or have found a solution. The feedback that the pheromones provide for future agents is based on partial information and reduces the accuracy of the heuristics.

Dropping pheromones before being transferred to the next host introduces a new difficulty. Because some agents fail to find a partial solution on one host (due to hard constraint violations), they are not transferred to the next host. The incomplete swarm from the queue on that next host is complemented with new agents, because of efficiency reasons. This way, not all agents that have completed their work on a host, have visited the same number of hosts. This makes it hard to decide which agents have the best results so far. An agent that has only visited one host is likely to have found a solution that violates less constraints that an agent that has already visited several hosts. If we would only consider the partial solution on the present host, we would not make use of all available information, making the feedback information less useful. More important, an agent that has visited several hosts would be more restricted in choosing its values on the present host. As a solution several groups of agents within the swarm are identified. Each group consists of agents that have visited the same hosts. The best agents of each group are allowed to drop pheromones. The more hosts an agent has visited (and thus the more information is contained in the pheromones), the more pheromones it drops.

Security

Next to very big problem instances, constraint satisfaction problems that require some degree of security are an important reason for distributed solving. Keeping the exact nature of a constraint private to one host is one of the main objectives. This type of security concern can be addressed by our framework. In the distributed version of the CSAA framework, each variable is assigned to a host. For security sensitive problems, the nature of the problem determines which host this is. In contrast with non-security sensitive problems, where a constraint between two variables that are located on a different host is present on each of those hosts, constraints that should remain private are only present on one host (suppose $Host\ I$). Such a constraint is checked when the second variable must be given a value. If the constraint is not present on the host the second variable is at (suppose $Host\ II$), it can not be checked at that moment. It may be even so that the second host ($Host\ II$) is not even aware there is a constraint at all. Consequently, an agent that has found a solution, must revisit all hosts to check if there are any constraints that have not been checked yet. If other hosts are allowed to know there is a constraint, but not allowed to

know the constraint itself, an agent can remember if there are any unchecked constraints and avoid unnecessary bandwidth consumption.

Having to revisit (some or all) hosts also has consequences on the pheromone dropping. If the possibility that an agent will revisit a node is large (this is the case when there are only a few hard constraints and thus the possibility of failure of a variable is small), the best agents could drop pheromones only when they revisit the host. This would delay the pheromone dropping, but it would incorporate more information in the pheromone trail (because all instead of part of the constraint checks are taken into account). An intermediate solution would be to drop pheromones two times: once during the first visit and a second time during the revisit. In that case, a suited ratio between the first and second dropping is very important.

In order to prevent an agent to remember a constraint and to make sure it can not violate the privacy, agents should no longer be responsible for checking the constraints. Instead, constraints should be checked by the nodes, only providing the agents with permitted values and, if applicable, the accumulated priorities of the violated soft constraints. Agents can then choose between those values.

Additional Optimizations

Initially, the variables (of a non-security related problem) are distributed randomly over the available hosts. This is not the optimal distribution: a better distribution can be achieved by using heuristics or by dynamically rearranging the variables. The number of constraints between a set of variables can be a heuristic for grouping them on the same host: a strong connection between variables on the same host tends to improve the behavior of the arc consistency and min-conflict heuristics. But sometimes, the number or importance of constraints is not a very good measure for such a strong connection. Furthermore, searching the most constrained groups would take a long time. Therefore, we chose to let the CSAA framework decide at runtime which variables should be grouped on the same host. This way, the startup-time can remain very short. The decision of which variable has to be transferred is made by using pheromones. Each main node keeps one pheromone trace for each remote host. When an agent has finished a run and drops pheromones on the selection and value edges, it can also drop pheromones on the main nodes. Pheromones are dropped when the variable that is associated with the main node has violated some constraints that involve variables on remote hosts. An amount of pheromones proportional to the priority of the violated constraints is dropped on the trace of that remote host. When one of the amounts of pheromone exceeds a predefined value, the variable is transferred to that host. Because pheromones evaporate, variables that only violate low-priority constraints with remote variables will never be transferred.

The distribution of the variables has one last drawback. Because not all variables are equally constrained, not all constraints are equally computationally intensive, not all hosts have an equal amount of variables because of the dynamic transfer of variables, and not all hosts have the same processing power, a swarm on one host can finish faster than a swarm on another host. When there are two hosts, this leads to very long queues on the slower host, because during the time one swarm is processed, the agents of more than one swarm, coming from the faster host arrive. Consequently, the slower host never needs to add any newly constructed agents. This also means that no agents are transferred from the slower host to the faster host, causing the queue of the faster host to be always empty and the queue of the slower host to keep growing. Eventually, this leads to memory overflows. In distributions with more than two hosts, similar queue problems arise.

There are two possible solutions for this problem. The first is to dynamically transfer variables and the associated constraints from the slower host to the faster host. However, this could interfere with the dynamic transfer of variables for efficiency reasons. The second solution is to restrict the number of agents in a swarm that are transferred. Agents that are not transferred, get terminated. Transferring only those agents that have the best solution so far, would prevent any local exploration of the graph (the violation of one extra constraint on the first host, that allows for a number of violations not to occur on a second host would never get transferred). Therefore, we chose to add a stochastic function to the transfer process. The better the partial solution of an agent, the bigger its chances of being transferred. The number of agents that can be transferred depends on the homogeneity of the hosts (measured in the number of agents that can be processed per unit of time). Because of possible transfer of variables, this number varies in time.

Experimental Results

We tested the distributed version of the CSAA framework on graph coloring problems: 10 different problems were generated, each with 30 variables. Each problem has a number of hard constraints (in the first problem 3% of all constraints were hard, in the second 6%,. . . , in the last one 30%), randomly chosen but ensuring a coloring with 20 colors is possible. Soft constraints with a random importance between 1 and 50 were added between all variable pairs that did not have a constraint yet. We used the distributed version of the CSAA framework on two hosts to color these problems using maximal 20 colors, trying each problem 10 times.

We compared the performance of the distributed CSAA framework with the Iterative Distributed Breakout (IDB) algorithm (Hirayama & Yokoo 1997). The IDB algorithm was chosen because it leads quickly to a (near-optimal) solution. As stated at the beginning of this paper, this is an important characteristic when dynamic problems have to be solved and the CSAA framework is also designed to exhibit this behavior. The IDB algorithm was distributed the same way as the CSAA framework was: the variables were divided randomly into two groups, each of which was placed on a different host. The CSAA framework was executed with the additional optimizations from the previous subsection (dynamic transfer of variables and stochastic choosing which agents are transferred) implemented. The best agents dropped pheromones each time before they left a host and did not return to previous hosts after a complete solution

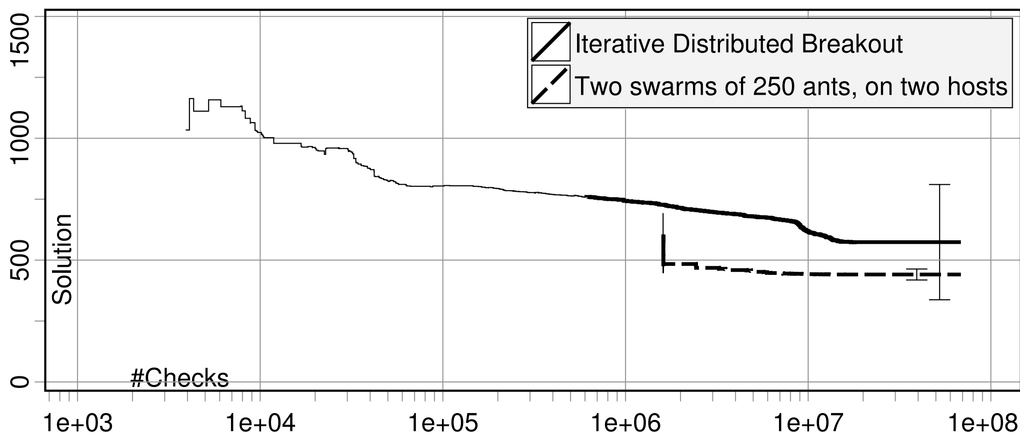

Figure 3: Average any-time curves for PCSP problems with 30 variables and 10 values, using Iterated Distributed Breakout (full line) and using the distributed version of the CSAA on two hosts with a swarm of 250 ants with pheromone trails on each host (dashed line). The thin lines indicate some runs found a solution at that number of checks, other did not yet. Error bars indicate 95% confidence intervals around the mean. The number of checks on the X-axis is the total number of checks on both hosts.

was found.

The results can be found in Figure 3. The thin lines indicate that some runs found a solution at that number of checks, other did not yet. Thick lines indicate all run found a solution. Assignments that violated one of the hard constraints were not counted as solutions. As for the mean performances, the CSAA framework is better than the IDB algorithm. The variance of the IDB algorithm is a lot larger than the one of the CSAA framework (see 95% confidence intervals on Figure 3). Statistically, the performance of the CSAA framework is not significantly better than the performance of the IDB algorithm, but there is only a 17% chance that IDB reaches better results. The Iterative Distributed Breakout algorithm finds a first solution faster than the CSAA framework. Problems the IDB algorithm is well suited for have a first solution 350 times faster with the IDB algorithm than with the CSAA framework (beginning of the thin lines). Problems that are hard for the IDB algorithm still receive a first solution three times faster than with CSAA (beginning of thick lines). This can be explained by the use of swarms: before the CSAA can decide on a first solution, two swarms of 250 agents have to complete the graph (or have to be terminated because of failure of a variable).

Conclusion and Future Work

In this paper, we discussed the distributed Constraint Satisfaction Ant Algorithm (CSAA) framework. It uses a very flexible ant-based system to solve Constraint Satisfaction Problems (CSP) and Partial Constraint Satisfaction Problems (PCSP). *Pheromone* trails are used both to mimic traditional search heuristics and to provide a flexible way of searching. The distribution of the framework is no trivial task, but requires a number of well thought-out design decisions we described in detail. The overall performace of the CSAA framework is good, compared to other algorithms that were designed for flexibility, in the centralized as well as in the distributed version.

Our ultimate goal is to extend the framework for handling Dynamic (Partial) Constraint Satisfaction Problems (DPCSP). That are PCSPs where the problem changes constantly. Given the flexibility and robustness of ant-based systems, the prospects of solving DPCSPs with the CSAA framework are looking very promising.

References

Chialvo, D. R., and Millonas, M. M. 1995. How Swarms Build Cognitive Maps. In Steel, L., ed., *The Biology and Technology of Intelligent Autonomous Agents*, volume 144. Nato ASI Series. 439–450.

Dorigo, M.; Bonabeau, E.; and Theraulaz, G. 2000. Ant Algorithms and Stigmergy. *Future Generation Computer Systems* (16):851–871.

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.

Hirayama, K., and Yokoo, M. 1997. Distributed Partial Constraint Satisfaction Problem. In *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP'97)*, 222–236.

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)*.

Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1992. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence* 58(1-3):161–205.

Rauch, E.; Millonas, M. M.; and Chialvo, D. R. 1995. Pattern Formation and Functionality in Swarm Models. *Phys. Lett. A* 207:185–193.