Available online at www.sciencedirect.com

SCIENCE DIRECT®

Artificial Intelligence 161 (2005) 117–147

Artificial Intelligence

www.elsevier.com/locate/artint

Sensor networks and distributed CSP: communication, computation and complexity ☆

Ramón Béjar [a,*], Carmel Domshlak [b], Cèsar Fernández [a], Carla Gomes [b], Bhaskar Krishnamachari [c], Bart Selman [b], Magda Valls [d]

[a] *Departament d'Informàtica i Enginyeria Industrial, Universitat de Lleida, Jaume II, 69, E-25001 Lleida, Spain*
[b] *Department of Computer Science, Cornell University, Ithaca, NY 14853, USA*
[c] *Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA 90089, USA*
[d] *Departament de Matemàtica, Universitat de Lleida, Jaume II, 69, E-25001 Lleida, Spain*

Received 3 April 2004; accepted 13 September 2004

Abstract

We introduce SensorDCSP, a naturally distributed benchmark based on a real-world application that arises in the context of networked distributed systems. In order to study the performance of Distributed CSP (DisCSP) algorithms in a truly distributed setting, we use a discrete-event network simulator, which allows us to model the impact of different network traffic conditions on the performance of the algorithms. We consider two complete DisCSP algorithms: asynchronous backtracking (ABT) and asynchronous weak commitment search (AWC), and perform performance comparison for these algorithms on both satisfiable and unsatisfiable instances of SensorDCSP. We found that random delays (due to network traffic or in some cases actively introduced by the agents) combined with a dynamic decentralized restart strategy can improve the performance of DisCSP algorithms.

☆ This work was supported in part by AFOSR (F49620-01-1-0076, Intelligent Information Systems Institute and MURI F49620-01-1-0361), CICYT (TIC2001-1577-C03-03 and TIC2003-00950), DARPA (F30602-00-2-0530), an NSF CAREER award (IIS-9734128), and an Alfred P. Sloan Research Fellowship. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the US Government.

* Corresponding author.
E-mail address: ramon@eup.udl.es (R. Béjar).

doi:10.1016/j.artint.2004.09.002

In addition, we introduce GSensorDCSP, a plain-embedded version of SensorDCSP that is closely related to various real-life dynamic tracking systems. We perform both analytical and empirical study of this benchmark domain. In particular, this benchmark allows us to study the attractiveness of solution repairing for solving a sequence of DisCSPs that represent the dynamic tracking of a set of moving objects.

Keywords: Distributed CSP benchmark; Phase transitions; Randomized combinatorial search; Communication network delays; NP-completeness

1. Introduction

In recent years we have seen an increasing interest in Distributed Constraint Satisfaction Problem (DisCSP) formulations to model combinatorial problems arising in distributed, multi-agent environments. In the world of networked systems, there is a rich set of distributed applications for which the DisCSP paradigm is particularly useful. In such distributed applications, constraints among agents, such as communication bandwidth and privacy issues, preclude the adoption of a centralized approach. During the last decade, many interesting results have been presented on algorithmic [3,5,10,24,27,31,32,34–36,38] and applicative [4,11,21–23,25] issues of dealing with DisCSPs (and this list is far from being exhaustive).

Study of alternative algorithms for a certain class of computational problems requires a comprehensive set of benchmark domains. These domains should provide us with problem instances inducing various forms of structure and various levels of complexity. Several success stories in the recent research in AI and other related areas show us that a wide palette of benchmark domains, accomplished with extensive analysis of their structure and complexity, helps to develop new algorithmic techniques. Examples of this can be found in the areas of AI planning [12–14], SAT solvers [30], etc.

To the best of our knowledge, so far DisCSP algorithms have been mostly studied on benchmarks from classical CSP (such as N-Queens, Graph Coloring, etc.), formulated in a distributed fashion. In this paper we introduce and study SensorDCSP, a naturally distributed benchmark inspired by several distributed applications arising in networked systems [2,8,19]. SensorDCSP involves a network of distributed sensors simultaneously tracking multiple mobile objects, and the problem underlying SensorDCSP is NP-complete. We show that the SensorDCSP domain undergoes a phase transition in satisfiability with respect to two control parameters: the level of sensor compatibility and the level of the sensor visibility. Standard DisCSP algorithms on problem instances of SensorDCSP exhibit the easy-hard-easy profile in complexity, peaking at the phase transition, which is similar to the pattern observed in centralized CSP algorithms. More interestingly, the relative strength of standard DisCSP algorithms on SensorDCSP is highly dependent on the satisfiability of the instances. This aspect has been overlooked in the literature on account of the fact that, so far, the performance of DisCSP algorithms has been evaluated primarily on satisfiable instances [37,38]. We study the performance of two well-known DisCSP algorithms—Asynchronous Backtracking (ABT) [36], and Asynchronous Weak-Commitment search

(AWC) [35]—on SensorDCSP. Both ABT and AWC use agent priority ordering during the search process. While these priorities are static in ABT, AWC allows for dynamic changes in the ordering and was originally proposed as an improvement over ABT. One of our findings is that although AWC does indeed perform better than ABT on satisfiable instances, just the opposite is true on unsatisfiable instances.

Our SensorDCSP benchmark also allows us to study other interesting properties that are specific to DisCSPs and dependent on the physical characteristics of the distributed environment. For example, while the underlying infrastructure or hardware is not critical in studying CSPs, we argue that this is not the case for DisCSPs in communication networks. This is because the traffic patterns and packet-level behavior of networks affect the order in which messages from different agents are delivered to one another, and thus can significantly impact the distributed search process. To investigate these kinds of effects, we implemented our DisCSP algorithms using a fully distributed discrete-event network simulation environment with a complete set of communication oriented classes. The network simulator allows us to realistically model the message delivery mechanisms of varied distributed communication environments ranging from wide-area computer networks to wireless sensor networks.

We study the impact of communication delays on the performance of DisCSP algorithms. We consider different link-delay distributions. Our results show that the presence of a random element due to the delays can improve the performance of AWC. Moreover, though link delay causes the performance of the standard ABT algorithm to deteriorate, a decentralized restart strategy that we have developed for ABT improves its solution time dramatically while also increasing the robustness of solutions with respect to the variance of the network link-delay distribution. These results are consistent with results on successful randomization techniques that were developed for the purpose of improving the performance of CSP algorithms [9]. Another novel aspect of our work is the introduction of a mechanism for *actively* delaying messages. The active delay of messages decreases the communication load of the system and, somewhat counter-intuitively, can also decrease the overall solution time.

While SensorDCSP provides a general abstraction for many real-life resource allocation problems, in tracking systems (that SensorDCSP was inspired by) the problems typically induce some clear spatial structure, leading to a relatively high decomposability of the problem. Addressing these systems, we introduce GSensorDCSP, a variant of SensorDCSP in which constrainedness of compatibility and visibility is conditioned by the locations of the sensors and objects on the plane. For this benchmark we perform both analytical and empirical complexity analysis. We show that, inspite of its inherently decomposable nature, GSensorDCSP is NP-complete, except for some special tractable cases. Identification of these tractable cases allows us to study performance of the DisCSP algorithms on a provably polynomial distributed problems. On the other hand, we show that DisCSP algorithms scale nicely on a wide subclass of GSensorDCSP, and this scalability makes using DisCSP algorithms feasible in many real-life applications. In particular, we analyze the AWC algorithm on a sequence of GSensorDCSP problems that represent a system of sensors tracking a set of moving objects. We discuss some properties of such a dynamic GSensorDCSP, and show how these properties could be exploited in the dynamic tracking systems.

The remainder of the paper is organized as follows: In Section 2 we describe SensorDCSP and model it as a DisCSP. In Section 3 we describe two standard DisCSP algorithms and the modifications we have incorporated into the algorithms. In Section 4 we present an empirical complexity analysis for SensorDCSP, and study active introduction of randomization by the agents. In Section 5 we present results on delays caused by different traffic conditions in the communication network. In Section 6 we introduce GSensorDCSP, and describe its modeling as a DisCSP. In Section 7 we present formal complexity results for GSensorDCSP, evaluate the performance of the DisCSP algorithms on various subclasses of this problem, and discuss solution repairing as a technique for dynamic CSPs. Finally, we present our conclusions in Section 8.

2. SensorDCSP—a benchmark for DisCSP algorithms

In a distributed CSP, variables and constraints are distributed among the different autonomous agents that have to solve the problem. A DisCSP is defined as follows: (1) A finite set $\{A_1, A_2, \ldots, A_n\}$ of agents; (2) A set $\{P_1, P_2, \ldots, P_n\}$ of local (private) CSPs, where CSP P_i pertains to agent A_i (and A_i is the only agent that can modify the values assigned to the variables of P_i); (3) A global CSP, each of whose variables is also a variable of one of the local CSPs.

In analysis of DisCSP algorithms, each agent is traditionally assumed to control only one problem variable. However, in our DisCSP modeling of SensorDCSP every agent needs to control not one, but three local variables. We extend the single-variable approach by modeling each agent as a set of multiple virtual agents, one for each agent's local variable. In order to distinguish between communication and computation costs, in our discrete-event simulator we use different delay distributions to distinguish between messages exchanged between virtual agents of a single real agent (intra-agent messages) and those between virtual agents of different real agents (inter-agent messages). This modeling technique seems to be useful in general, since in many realistic problems an agent might control more than one variable. In that case, the time spent by an agent trying to find an assignment for its own local variables (consistent with its intra-agent and inter-agent constraints) would be only affected by the computation cost of the agent hardware.

The availability of a realistic benchmark of satisfiable and unsatisfiable instances, with tunable complexity, is critical for the study and development of new search algorithms. Unfortunately, in the DisCSP literature one cannot find such a benchmark. SensorDCSP, the sensor-mobile problem, is inspired by a real distributed resource allocation problem [1] and offers such desirable characteristics.

In SensorDCSP we have multiple sensors $S = \{s_1, \ldots, s_m\}$ and multiple mobiles $T = \{t_1, \ldots, t_n\}$ which are to be tracked by the sensors. The goal is to allocate three sensors to track each mobile node, such that all these triplets of sensors are pair-wise disjoint and consistent with two sets of constraints: visibility constraints and compatibility constraints. Fig. 1 shows an example with six sensors and two mobiles. Each mobile has a set of sensors that can possibly detect it, as depicted by the bipartite visibility graph in Fig. 1(a). In addition, it is required that each mobile be assigned three sensors that satisfy a compatibility relation with each other; this compatibility relation is depicted by the graph in Fig. 1(b).

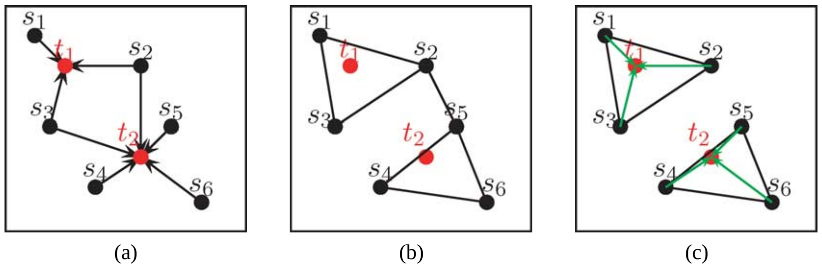


Fig. 1. A SensorDCSP problem instance: (a) Visibility graph; (b) Compatibility graph; (c) Feasible sensors/mobiles assignment.

Finally, it is required that each sensor only track at most one mobile. A possible solution is depicted in Fig. 1(c), where the set of three sensors assigned to each mobile is indicated by the lighter edges.

SensorDCSP is NP-complete, since the problem of partitioning a graph into cliques of size three can be reduced to it [2,17]. This is not true, however, of the limiting case in which every pair of sensors is compatible. That case is polynomially solvable, because as each such problem can be reduced to a feasible flow problem in a bipartite graph [18].

For our experiments, we define a random distribution of instances of SensorDCSP. An instance of the problem is generated from two different random graphs, the visibility graph and the compatibility graph. Apart from the number of mobiles and number of sensors, we also specify parameters controlling edge density of the visibility graph (P_v) and edge density of compatibility graph (P_c). Each of these parameters specifies the independent probability of including a particular edge in the corresponding graph. As these two graphs model the resources available to solve the problem, P_v and P_c control the number of constraints in the generated instances.

We have developed an instance generator for these random distributions that generates DisCSP-encoded instances of SensorDCSP. We believe that SensorDCSP is a good benchmark problem because it abstracts many real-life resource allocation problems, and because, as we shall show, one can easily generate easy/hard, unsatisfiable/satisfiable instances by tuning the parameters P_v and P_c appropriately. Our DisCSP encoding of SensorDCSP is as follows: Each mobile is associated with a different agent. There are three different variables per agent, one for each sensor that we need to allocate to the corresponding mobile. The value domain of each variable is the set of sensors that can detect the corresponding mobile. The intra-agent constraints between the variables of one agent are that the three sensors assigned to the mobile must be distinct and pair-wise compatible. The inter-agent constraints between the variables of different agents are that a given sensor can be selected by at most one agent. In our implementation of the DisCSP algorithms, this encoding is translated to an equivalent formulation where we have three virtual agents for every real agent, each virtual agent handling a single variable.

We should also address the question of where the agents actually reside. The original problem [1] that inspired SensorDCSP assumes passive/non-collaborative mobile nodes, in which case it must be assumed that there exists a mechanism whereby one of the sensor nodes within range of each mobile node contains the agent corresponding to that mobile (the description of such a mechanism is beyond the scope of this study, but it could be implemented using distributed leader election algorithms, for example). Alternatively, in other tracking scenarios the mobile nodes may be collaborative and have the computational ability to execute their own agents.

3. DisCSP algorithms

In this work we consider two specific DisCSP algorithms, Asynchronous Backtracking Algorithm (ABT), and Asynchronous Weak-Commitment Search Algorithm (AWC). We provide a brief overview of these algorithms but refer the reader to [38] for a more comprehensive description. We also describe the modifications that we introduced into these algorithms. As mentioned earlier, we assume that each agent can only handle one variable. In what follows, the *neighbors* of a given agent are the agents with whom it shares constraints.

The *Asynchronous Backtracking Algorithm* (*ABT*) is a distributed asynchronous version of a classical backtracking algorithm. This algorithm needs a static agent ordering that determines an ordering of the variables of the problem. Agents use two kinds of messages for solving the problem—*ok* messages and *nogood* messages. Agents initiate the search by assigning an initial value to their variables. An agent changes its value when it detects that it is not consistent with the assignments of higher priority neighbors, and so it maintains an agent view, which consists of the variable assignments of its higher priority neighbors.

Each time an agent assigns a value to its variable, it issues the *ok* message to inform its lower-priority neighbors of this new assignment. If an agent is unable to find an assignment that is consistent with the assignments of all of its higher-priority neighbors, it sends a *nogood* message, which consists of a subset of that agent's view that makes it impossible for the agent to find a consistent assignment for itself; the *nogood* message is sent to the lowest-priority agent among all the (higher-priority) agents in that particular subset of that agent's view. Receipt of a *nogood* message causes the receiver agent to record the content of that message as a new constraint and then try to find an assignment that is consistent with its higher-priority neighbors and with all of its recorded constraints. If the top-priority agent is forced to backtrack (which implies that its assignment is inconsistent with at least one of its recorded constraints, since there is no higher-priority neighbor with which its assignment could possibly clash), this means that the problem has no solution. If, on the other hand, the system reaches a state where all agents are happy with their current assignments (no *nogood* messages are generated), this means that the agents have found a solution.

The *Asynchronous Weak-Commitment Search Algorithm* (*AWC*) can be seen as a modification of the ABT algorithm. The primary differences are as follows: A priority value is determined for each variable, and the priority value is communicated using the *ok* message. If an agent's current assignment is inconsistent with that agent's view, the agent selects a new consistent assignment that minimizes the number of constraint violations with lower-priority neighbors. When an agent cannot find a consistent value and generates a new *nogood*, it sends the *nogood* message to all its neighbors and raises its priority by one unit above the maximal priority of its neighbors. Then it finds an assignment that is consistent with the assignments of its higher-priority neighbors and informs its neighbors by sending them *ok* messages. If no new *nogood* can be generated, the agent waits for the next message.

Considering both ABT and AWC, in recent years it has been recognized that randomization is a useful technique for enhancing the performance of complete backtracking-based CSP solvers [9]. We therefore wish to explore randomization strategies in our context. The most obvious way of introducing randomization in DisCSP algorithms is by randomizing the value selection strategy used by the agents. In the ABT algorithm this is done by performing a uniform random value selection, among the set of values consistent with the agent view and the *nogood* list, every time the agent is forced to select a new value. In the AWC algorithm, we randomize the selection of the value among the values that are not only consistent with the agent view and the nogood list but also minimize the number of violated constraints. This form of randomization is analogous to the randomization techniques used in backtrack search algorithms.

A novel way of randomizing the search in the context of DisCSP algorithms is to introduce forced delays in the delivery of messages. Delays introduce randomization in that the order in which messages from different agents reach their destination agents determines the order in which the search space is traversed. More concretely, every time an agent has to send a message, it follows the following procedure:

```
1. with probability $p$:
       $d := D \cdot (1 + r)$;
   else (with probability $(1 - p)$)
       $d := D$;
2. deliver the message with delay $d$.
```

Transmitting message m with delay d means that the agent requires its communication interface to add d seconds to the delivery time currently scheduled for m and all the successors of m in the message queue. The latter preserves the order of transmission and reception for the messages sent from one agent to another agent. The parameter r is the fraction of the communication delay (D) added by the agent. Section 4.2 details more accurately this active introduction of delays. In our implementation of the algorithms, this strategy is performed by using the services of the discrete event simulator that allow specific delays to be applied selectively in the delivery message queue of each agent.

We have also developed the following decentralized restarting strategy suitable for the ABT algorithm: the highest-priority agent uses a timeout mechanism to decide when a restart should be performed. It performs the restart by changing its value at random from the set of values consistent with the *nogoods* learned so far. Then, it sends *ok* messages to its neighbors, thus producing a restart of the search process, but without forgetting the *nogoods* learned. This restart strategy is different from the restart strategy used in centralized procedures, such as Satz-rand [9]. Here, the search is not restarted from scratch, but rather benefits from prior mistakes since all agents retain the *nogoods*.

4. Complexity profiles of DisCSP algorithms on SensorDCSP

As mentioned earlier, when studying distributed algorithms several factors may determine their performance. Some of those factors are inherent to the search procedure, such as agent ordering. In this work, we always assume an arbitrary lexicographic ordering of the agents, focusing our attention on other factors, specifically these related to the physical characteristics of the distributed environment. For example, the traffic patterns and packet-level behavior of networks can affect the order in which messages from different agents are delivered to each other, significantly impacting the distributed search process. To investigate these kinds of effects, we have developed an implementation of the algorithms ABT and AWC using the Communication Networks Class Library (CNCL) [16]. This library provides a discrete-event network simulation environment with a complete set of communication-oriented classes. The network simulator allows us to realistically model the message-delivery mechanisms of various distributed communication environments ranging from wide-area computer networks to wireless sensor networks. Finally, in our implementations of ABT and AWC we have not limited the number of nogoods learned by the agents. Although in the worst case this can require exponential space, in our experiments we have not noticed any exponential blow-up in the number of nogoods learned by any agent. Two reasons for this could be as follows. First, for the typical instances of the random distribution, the particular characteristics of the SensorDCSP constraints could be bounding the number of variables that a given variable can have constraints with. Second, it is possible that the size of the instances tested so far is insufficient to discover such an exponential blow-up in space complexity.

The results shown in this section have been obtained according to the following scenario: The communication links used for communication between virtual agents of different real agents (inter-agent communication) are modeled as random-delay links, with a negative-exponential distribution and a mean delay of 1 time unit. The communication links used by the virtual agents of the same real agent (intra-agent communication) are modeled as fixed delay links, with a delay of 10^{-3} time units. Here we use fixed-delay links because we assume that a set of virtual agents work inside a private computation node and this allows virtual agents to communicate with each other using dedicated communication links. This scenario could correspond to a heavy-loaded network situation where inter-agent delay fluctuations obey to the queuing-time process on intermediate systems. The difference between the two delays by a factor of 1000 reflects that intra-agent computation is usually less expensive that inter-agent communication. In Section 5 we will see how different delay-distribution models over the inter-agent communication links can impact the performance of the algorithms.

In our experiments with SensorDCSP we considered different sets of instances with 3 mobiles and 15 sensors. Every set contained 19 instances and was generated with a different pair of values for the parameters P_c and P_v (ranging from 0.1 to 0.9), providing us with 81 data points. Each instance has been executed 9 times, each time with a different random seed. The results reported in this section were obtained using a sequential value selection function for the different algorithms. By sequential we mean that values are chosen according to a lexicographic order.

Fig. 2 shows the percentage of satisfiable instances as a function of P_c and P_v. When both probabilities are low, most of the generated instances are unsatisfiable. For high probabilities, however, most of the instances are satisfiable. The transition between the satisfiable and unsatisfiable regions occurs within a relatively narrow range of these control parameters, analogous to the phase transition in CSP problems, e.g., in SAT [26]. Also consistent

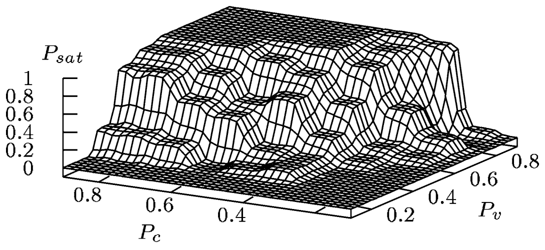


Fig. 2. Percentage of satisfiable instances depending on the density parameter for the visibility graph (P_v) and the density parameter for the compatibility graph (P_c).

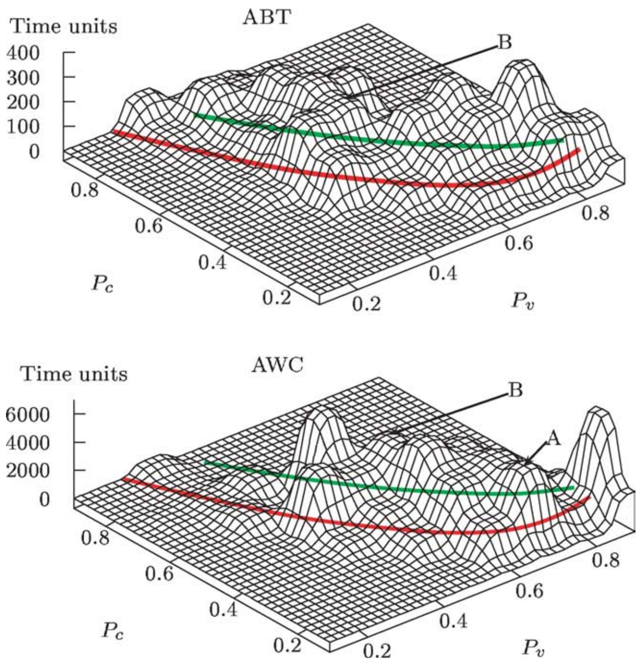


Fig. 3. Mean solution time with respect to P_v and P_c for ABT and AWC algorithms. Points A and B show the locations of the two hard instances analyzed in Section 4.2.

with other CSP problems is our observation that the hardest instances for these backtracking algorithms generally occur in the region where the phase transition occurs. Fig. 3 shows the mean solution time with respect to the parameters P_c and P_v. As can be seen there, the hardest instances lie on the diagonal that defines the phase-transition zone, with a peak for instances with a low P_c value. The dark and light solid lines overlaid on the mesh depict the location of the iso-lines for $P_{sat} = 0.2$ and $P_{sat} = 0.8$, respectively, as per the phase-transition surface of Fig. 2. As mentioned earlier, the SensorDCSP problem is NP-complete only when not all sensors are pairwise compatible (i.e., when $P_c < 1$) [18]. Therefore, the parameter P_c could separate regions of different mean computational complexity, as in other mixed P/NP-complete problems like 2 + p-SAT [26] and 2 + p-COL [33]. This is particularly noticeable in the mean-time distribution for AWC shown in Fig. 3.

We observe that the mean times to solve an instance with AWC appear to exceed those with ABT by an order of magnitude. At first glance, this is a surprising result, considering that the AWC algorithm is a refinement of ABT and that results reported for satisfiable instances in the literature [37,38] point to better performance for AWC. One plausible explanation for the discrepancy is the fact that our results deal with both satisfiable and unsatisfiable instances. On further investigation, we found that while AWC does indeed outperform ABT on satisfiable instances, it is much slower on unsatisfiable instances. This result seems consistent with the fact that the agent hierarchy on ABT is static, while for AWC the hierarchy changes during problem solving; consequently, AWC might be expected to take more time to inspect all the search space when unsatisfiable instances are considered.

4.1. Randomization and restart strategies

In this subsection we present the benefits of using randomized value selection and restart strategies for distributed CSP algorithms. The introduction of a randomized value selection function was directly assumed in [37]. In extensive experiments we performed with our test instances, we found that the randomized selection function is indeed better than a sequential value selection. Randomization can result in greater variability in performance, however, so ABT should be equipped with a restart strategy. We have not defined a restart strategy for AWC, because, as will be seen in Section 5, the dynamic priority strategy of AWC can be viewed as a kind of built-in partial restart strategy. In the results reported in the rest of the paper, both ABT and AWC use randomized value selection functions.

To study the benefits of the proposed restart strategy for ABT, we have used restarts in solving hard satisfiable instances with ABT. Fig. 4 shows the mean time needed to solve a hard satisfiable instance, together with the corresponding 95% confidence intervals, for a number of cutoff times. We observe that there is clearly an optimal restart cutoff time that

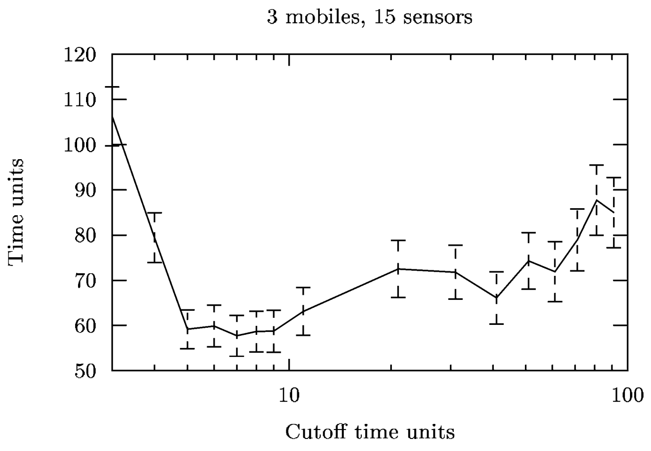


Fig. 4. Mean time to solve a hard satisfiable instance by ABT using restarts, plotted with different cutoff times.

gives the best performance. As will be discussed in Section 5, use of restart strategies is essential when dealing with the delays that occur in real communication networks, given the high variance in the solution time due to randomness of link delays in the communication network.

4.2. Active delaying of messages

One rather novel way of randomizing a DisCSP algorithm is to introduce delays in the delivery of the agents' outgoing messages, as we described in Section 3. In this subsection we present the results of our experiments with the AWC and ABT algorithms. The amount of delay added by the agents is a fraction r (from 0 to 1) of the delay in the inter-agent communication links. Here, we consider the case where all the inter-agent communication links have fixed delays of 1 time unit, because we want to isolate the effect of the delay added by the agents. This is in contrast to the experiments described elsewhere in this section, where we report the effects of allowing variable inter-agent delays.

Fig. 5 shows the results of using AWC to solve a hard satisfiable instance from our SensorDCSP domain (namely, the one that corresponds to point A in Fig. 3). The solution time and the number of messages are plotted for various values of p, the probability of adding a delay, and r, the fraction of delay added with respect to the delay of the link. The horizontal plane cutting the surface shows the median time needed by the algorithm when we consider no added random delays ($p = 0$, $r = 0$). We see that agents can indeed improve the performance of AWC by actively introducing additional, random delays when exchanging messages. The need to send messages during the search process is almost always reduced when agents add random delays; in the best case the number of messages delivered can be as much as a factor of 3 smaller than in the worst case. Perhaps more surprisingly, the solution time can also improve if the increase in delay (r) is not too high.

Fig. 6 shows the results with AWC (left) and ABT (right) for a hard satisfiable instance (namely, the one that corresponds to point B in Fig. 3). We observe that the performance of AWC is improved in a greater number of cases than that of ABT. Moreover, in the best case the solution time is smaller than that in the worst case by a factor of 2.25 for AWC and 1.63 for ABT. It appears that AWC benefits to a greater extent overall than ABT when it comes to the incorporation of delays added by agents. The reason for this could be the ability of AWC to exploit randomization via its inherently restarting search strategy.

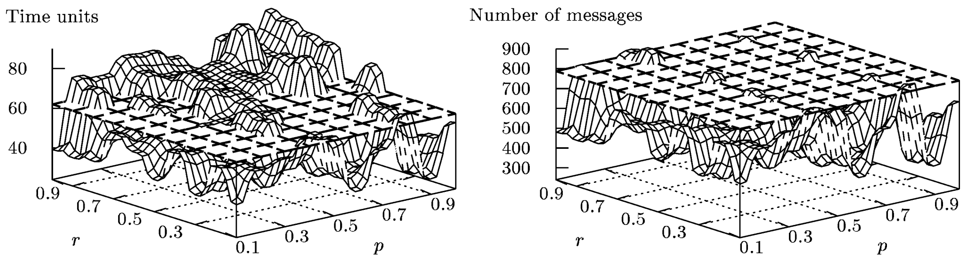


Fig. 5. Median time and number of messages needed to solve a hard satisfiable instance (point A in Fig. 3) with AWC when agents add random delays in outgoing messages. The horizontal plane represents the median time (or the median number of messages) for the case where no delay is added ($p = 0$).

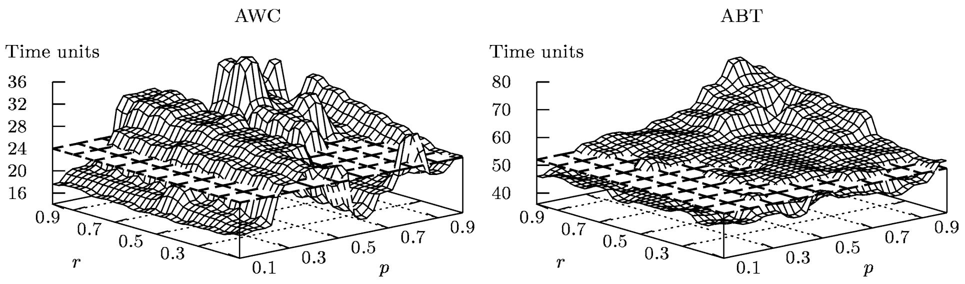


Fig. 6. Median time for AWC and ABT to solve a hard satisfiable (point B in Fig. 3) instance when agents add random delays in outgoing messages. The horizontal plane represents the median time for the case where no delay is added ($p = 0$).

5. The effect of the communication network data load

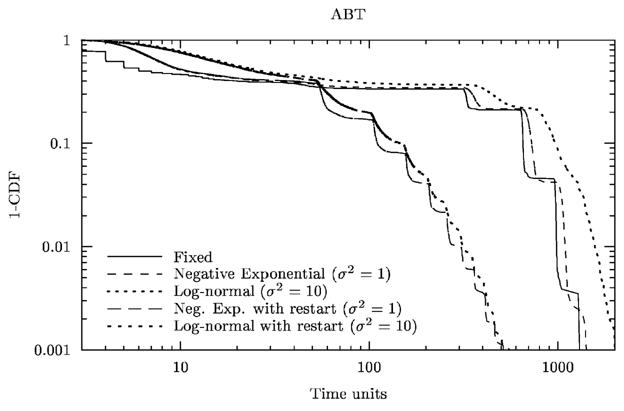
As described in the previous section, when working on a communication network with fixed delays, the performance of AWC can be improved, depending on the amount of random delay addition that the agents introduce into the message delivery system. In real networks, however, the conditions of data load present in the communication links used by the agents cannot always be modeled with fixed-delay links. It would thus seem worthwhile to determine how differences in communication network environments can affect the performance of the algorithms. In Section 4.2 we considered inter-agent communication links with random, exponentially distributed delays. In this section we study the effect produced in the performance of DisCSP algorithms by considering delay distributions corresponding to different traffic conditions.

We examine various link-delay distributions that can be used to model communication network traffic. Because of their attractive theoretical properties, negative-exponential distributions of arrival times have traditionally been used to model data traffic. In the past decade, however, it has been shown that although these models are able to capture single-user-session properties, they are not suitable for modeling aggregate data links in local- or wide-area network scenarios [7,20,28]. In view of this, we have simulated network delays according to three different models for the inter-arrival time distribution: the aforementioned negative-exponential distribution, the log-normal distribution, and the Fractional Gaussian Noise (FGN) [29] distribution.

The log-normal distribution can be used to obtain distributions with any desired variance, whereas FGN processes are able to capture crucial characteristics of the Internet traffic, such as long-range dependence and self-similarity that do not lend themselves to other models. We synthesize FGN from α-stable distributions with typical parameter values of $H = 0.75$ and $d = 0.4$. Fig. 7 shows the cumulative density functions (CDF) of the time required for three algorithms (AWC, ABT, and ABT with restarts) to solve hard instances when all the inter-agent communication links have delays modeled as fixed, negative exponential, and log-normal. The means were nearly identical, but the variances were quite different. Table 1 presents the estimated mean and variance of the number of messages exchanged when using each of the three aforementioned algorithms, together with several different inter-agent link-delay distributions, to solve the same hard instance. The

Table 1
Estimated mean and variance, from the empirical distributions, of the number of messages for different algorithms and different inter-agent link delay models when solving a hard satisfiable instance

Delay distribution	Mean			Variance		
	ABT	ABT-rst	AWC	ABT	ABT-rst	AWC
Fixed	1.8×10^5	1.2×10^5	8.2×10^2	3.6×10^{10}	1.3×10^{10}	3×10^5
Negative expon. ($\sigma^2 = 1$)	1.7×10^5	1.5×10^5	3.5×10^2	2.8×10^{10}	0.9×10^{10}	4.5×10^5
Log-normal ($\sigma^2 = 5$)	2.2×10^5	1.3×10^5	3.5×10^2	5.0×10^{10}	1.7×10^{10}	4.8×10^5
Log-normal ($\sigma^2 = 10$)	2.6×10^5	1.6×10^5	3.5×10^2	7.1×10^{10}	2.4×10^{10}	4.9×10^5



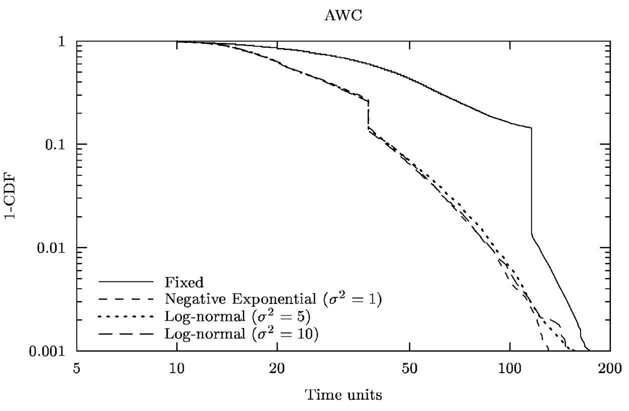


Fig. 7. Cumulative density functions (CDF) of the time needed to solve hard instances for their respective algorithms, AWC, ABT and ABT with restarts under different link delay models.

estimated mean and variance of the solution time for the same scenarios show an analogous behavior to the one observed with the number of messages. The results in Fig. 7 and Table 1 show that the delay distributions have an algorithm-specific impact on the performance of both AWC and basic ABT.

For the basic ABT, the solution time on hard instances becomes worse when channel delays are modeled by random distributions as opposed to the fixed delay case. The greater the variance of the link delay, the worse ABT performs. However, introducing the restart strategy has the desirable effect of improving the performance of ABT. Furthermore, ABT with restarts is fairly robust and insensitive to the variance in the link delays. AWC behaves differently from the basic ABT. On hard instances, having randomization in the link delays improves the solution time compared to the fixed delay channel. Likewise, the mean solution time for AWC is extremely robust to the variance in communication link delays, although the variance of solution time is slightly affected by this. Note that our experiments with FGN delay models show no significant differences in performance for the three algorithms in relation to other traffic models with the same variance.

In general, we found that on satisfiable instances, AWC always performs significantly better than both basic and restarts-enhanced ABTs. Therefore, AWC appears to be a better candidate in situations where most instances are likely to be satisfiable, and where we cannot avoid random delays in the links.

6. Grid-based SensorDCSP

The above analysis of SensorDCSP problems provides us with the first results on behavior of distributed CSP algorithms in close-to-real-world distributed applications. Observe that the very concrete specification of the SensorDCSP problem helps us both to analyze its computational complexity, and to establish coherent experiments for empirical analysis. However, getting closer to the real-world tracking systems, one may have to further specify the properties of the domain. The main information that we believe should be captured in analysis of various tracking systems is the *spatial* properties of both communication between the sensors and visibility of the mobiles. Two reasons make capturing this information essential:

(1) Given spatial limitations for both communication between the sensors and visibility of the mobiles, the complexity analysis for general SensorDCSP provides only upper bounds on the complexity of any spatially-limited SensorDCSP. In addition, deriving conclusions on various sub-classes of spatially-limited SensorDCSP from the empirical results on general SensorDCSP is not straightforward whatsoever. In particular, this makes hard to analyze *scalability* of the DisCSP algorithms with respect to real-life tracking systems.
(2) The overall goal of any tracking system is to track a set of *moving* objects, and this set is not necessarily constant over time (e.g., some tracked objects run out of the region covered by the sensors, while some new objects are getting into this region). Performance analysis of such a dynamic system is impossible without some realistic assumptions about the *dynamics* of the moving objects, which in turn can be specified only with respect to some concrete spatial model of SensorDCSP.

In addition, spatial nature of the problem instances is likely to lead to inherently decomposable problems, making adopting the DisCSP approach even more attractive. Influenced by the above motivation and the properties of a recently studied challenge problem for distributed tracking systems [1,15], we introduce a *grid-based* SensorDCSP benchmark, and perform both analytical analysis of this problem and empirical study of DisCSP algorithms on both static and dynamic settings of this problem.

The *Grid-based SensorDCSP* (or *GSensorDCSP*, for short) is a specific variant of the general SensorDCSP: as before, we have multiple sensors $S = \{s_1, \dots, s_m\}$, multiple objects $T = \{t_1, \dots, t_n\}$ which are to be tracked by the sensors subject to visibility and compatibility constraints, and the goal is to allocate three sensors to track each object, while keeping these triplets of sensors pair-wise disjoint. However, in GSensorDCSP the sensors are located on the nodes of a uniform grid of m nodes, and the mobile objects are located within the surface enclosed by the grid (i.e., the grid specifies the generally trackable region).[1] Furthermore, the visibility and compatibility constraints in GSensorDCSP relate to the physical limitations of the sensors and the properties of the terrain on which the sensors are located. In this section we provide a formal classification of the GSensorDCSP problem instances, together with an abstract model for specifying such instances with requested properties.

6.1. Locality of communication and visibility

The physical limitations of the sensors are modeled by the notions of *k-compatibility* and *k-visibility*. The k-compatibility window for sensor s_i, denoted as $\mathbb{C}^k(s_i)$, corresponds to the set of all sensors that are at most k general (rectilinear and/or diagonal) hops from s_i. For example, the black sensors in Fig. 8(a) correspond to 1-compatibility windows for the gray sensor. Similarly, the k-visibility window for a mobile t_j, denoted as $\mathbb{V}^k(t_j)$, corresponds to the set of all sensors that are at most k general hops around t_j. For example, the black sensors in Fig. 8(b) correspond to 2-visibility windows for the rectangular mobile. Note that, we have $|\mathbb{C}^k(s_i)| \leqslant (2k+1)^2 - 1$ (where the strict equality holds for all sensors located at least k hops from the boundaries of the grid), and $|\mathbb{V}^k(t_j)| \leqslant 4k^2$.

Given a GSensorDCSP problem instance Π, denote by $\mathcal{C}(s_i)$ the set of sensors that the sensor s_i can communicate with, and by $\mathcal{V}(t_j)$ the set of sensors that can track the mobile t_j. The compatibility graph of Π is called *k-restricted* if and only if, for every sensor s_i we have $\mathcal{C}(s_i) \subseteq \mathbb{C}^k(s_i)$, and there exists a sensor $s_{i'}$ such that $\mathcal{C}(s_{i'}) \not\subseteq \mathbb{C}^{k-1}(s_{i'})$. In turn, the compatibility graph of Π is called *k-enhanced* if and only if, for every sensor s_i we have $\mathbb{C}^k(s_i) \subseteq \mathcal{C}(s_i)$, and there exists a sensor $s_{i'}$ such that $\mathbb{C}^{k+1}(s_{i'}) \not\subseteq \mathcal{C}(s_{i'})$. The corresponding notions of k-restrictness and k-enhanceness for the visibility graph of Π are defined similarly. It is easy to see that higher values of k for both compatibility and visibility correspond to more powerful sensors. For example, thinking of the gray sensor in Fig. 8(c) as of the only sensor, the compatibility graph corresponding to Fig. 8(c) is 2-restricted and 1-enhanced. Similarly, thinking of the rectangular mobile in Fig. 8(d) as of the only mobile, the visibility graph corresponding to Fig. 8(d) is 2-restricted and 0-enhanced.

[1] We adopt a general position assumption that the objects are not located on the edges of the grid, but in the cells formed by the grid.

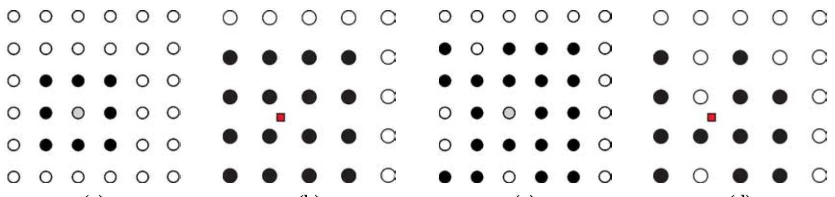


Fig. 8. k-compatibility and k-visibility windows. Locality of communication and visibility.

6.2. Connecting locality and constrainedness

While physical limitations of the sensors in GSensorDCSP problems are modeled via the locality windows, terrain limitations are modeled via incompleteness of compatibility and visibility within the windows. This part of modeling is done in a way very similar to this for the general SensorDCSP: Within a particular problem class (k_c, k_v) representing problems with k_c-restricted compatibility graph and k_v-restricted visibility graphs, the problems can be ordered according to the *local constrainedness*, i.e., the average number of sensors that a sensor can communicate with and the average number of sensors that can track a mobile object. For the experiments, a random distribution of GSensorDCSP instances for a particular pair of locality parameters (k_c, k_v) is defined as follows. Similarly to the general SensorDCSP, an instance of GSensorDCSP is generated from two different random graphs, the visibility graph and the compatibility graph. Apart of the parameters number of mobiles and number of sensors, we also specify the parameters $P_v, P_c \in (0, 1]$ that control the edge density of visibility and communication graphs, respectively. These parameters specify the independent probability of including a particular edge in the corresponding graph. However, not as for the general SensorDCSP, these parameters have only a local effect: For every pair of sensors s_i and s_j, the probability $\Pr(s_i, s_j)$ for the edge (s_i, s_j) to be a part of the communication graph is given by:

$$\Pr(s_i, s_j) = \begin{cases} 0, & s_j \notin \mathbb{C}^{k_c}(s_i), \\ P_c, & s_j \in \mathbb{C}^{k_c}(s_i). \end{cases} \tag{1}$$

Similarly, for the visibility graph, we have:

$$\Pr(t_i, s_j) = \begin{cases} 0, & s_j \notin \mathbb{V}^{k_v}(t_i), \\ P_v, & s_j \in \mathbb{V}^{k_v}(t_i). \end{cases} \tag{2}$$

Clearly, higher values for P_c and P_v correspond to less problematic terrain conditions for communication and tracking, respectively. To conclude, each problem instance of GSensorDCSP can be characterized by six parameters:

- *Order of the problem*, characterized by both the number of sensors and the number of mobiles (n and m, respectively),
- *Level of decomposition*, modeled via locality of compatibility and visibility, using the corresponding notions of window restrictness (k_c and k_v), and
- *Level of constrainedness*, modeled via the expected fraction of sensors that can communicate with a sensor and the expected fraction of sensors that can track a mobile

object, out of the maximally possible such numbers specified by the level of decomposition. These aspects of the problem instances are modeled using the uniform probability distributions P_c and P_v with their corresponding means.

7. Computational analysis of GSensorDCSP

In this section we present a complexity analysis for GSensorDCSP. Despite the a priori problematic multi-parametric nature of this problem, its concrete definition allows us to perform both analytical and empirical complexity analysis. Our analytical analysis characterizes both tractable and hard subclasses of GSensorDCSP. This classification both guides our empirical evaluation, and describes the connection between various problem parameters and the expected hardness of the problem. In turn, our experimental analysis shed light on both the phase transition in satisfiability of GSensorDCSP and the scalability expected from DisCSP algorithms on spatial SensorDCSP problems.

7.1. Complexity results for GSensorDCSP

In this section we perform an extensive formal complexity analysis of GSensorDCSP, identifying both tractable and hard subclasses of the GSensorDCSP problem.

Lemma 1. *GSensorDCSP is* NP-*complete.*

Proof. GSensorDCSP is a special case of SensorDCSP, thus it is clearly in NP. The proof of hardness is by a straightforward reduction from SensorDCSP. Given a general SensorDCSP problem Π with sensors $S = \{s_1, \ldots, s_m\}$ and mobile objects $T = \{t_1, \ldots, t_n\}$, the corresponding GSensorDCSP problem Π' is defined as follows: Let k be the smallest number such that $k \geqslant m$ and $k = l \times l'$, where $l, l' \in \mathbb{N}$. The sensor set of Π' is $S' = \{s'_1, \ldots, s'_k\}$, where, for $1 \leqslant i \leqslant m$, we have $s'_i = s_i$, and all these k sensors are located on the nodes of a uniform $l \times l'$ grid. The set of mobile objects of Π' is this of Π, and these objects are arbitrarily located within the grid. Finally, the compatibility and visibility graphs of Π' are identical to these of Π. Obviously, there exist a solution for the problem Π if and only if there exist a solution for the problem Π'. □

Note that the notions of locality in the GSensorDCSP problems constructed from the general SensorDCSP problems as in the proof of Lemma 1 are redundant: In general, if $l \geqslant l'$, we can only say that the compatibility and visibility graphs of the generated problems Π' are $(l-1)$-restricted. However, later we discuss GSensorDCSP with constantly bounded compatibility and visibility windows.

Recall that SensorDCSP is polynomial for complete compatibility graphs [18]. The corresponding notion in GSensorDCSP is this of *locally complete* compatibility graphs, as it is summarized by Lemma 2.

Lemma 2. *Given an GSensorDCSP problem instance Π with an i-enhanced compatibility graph, and j-restricted visibility graph, if $i \geqslant 2j - 1$, then Π is solvable in polynomial time.*

Proof. Without loss of generality, assume that, for every sensor s, we have $\mathcal{C}(s) = \mathbb{C}^i(s)$. In this case, if $i \geqslant 2j - 1$, then, for every mobile t, we have that every pair of sensors in $\mathcal{V}(t)$ can communicate one with the other. Therefore, this problem can be presented as a feasible integral flow problem in a bipartite graph, similarly to the way it is done for SensorDCSP problems with complete compatibility. □

Observe that, while Lemma 1 shows the general hardness of GSensorDCSP, the practical relevance of this result is extremely limited. Recall that the central motivation for specifying a spatial model for SensorDCSP was that, in real world, both the communication and tracking abilities of the sensors are spatially limited. Therefore, the complexity analysis of GSensorDCSP would be helpful only if it will be parametrized by the problem's level of decomposition. Below we perform such an analytical analysis, parametrized by the restrictness of visibility.

Theorem 3. *Any GSensorDCSP problem instance with* 1*-restricted visibility graph $\mathcal{V}$ is solvable in polynomial time.*

The proof of Theorem 3 by reduction to the problem of feasible integer flow appears in Appendix A. In turn, Theorem 4 shows that extending restrictness of visibility to $\mathbb{V}^2$ makes the GSensorDCSP problem hard, and this result is independent of the restrictness of compatibility between the sensors.

Theorem 4. *GSensorDCSP with* 2*-restricted visibility is* NP*-complete.*

The proof of Theorem 4 by reduction from 3-SAT appears in Appendix A.

7.2. Complexity profiles of the AWC algorithm on GSensorDCSP

For the first experiment with the AWC algorithm, we consider different sets of instances with 25 sensors (grid 5×5) and 5 mobiles, with every set generated with different values for the parameters P_c and P_v with respect to Eqs. (1) and (2). The parameters P_c and P_v are ranging from 0.1 to 1 with an increment of 0.1, giving a total number of 100 data sets, where every set contains 50 instances. Recall that by k_v and k_c we refer to the parameters controlling the restrictness of the visibility graph and compatibility graph, respectively. It is worth to mention that in contrast to the time model assumed in Section 4, the forthcoming experiments in the GSensorDCSP domain (but in the dynamic case) assume a random negative exponential distributed delay, with a mean of 1 time unit for both inter-agent and intra-agent communication. The reason for such an assumption is that we are no longer interested in time performance but in complexity analysis of the problem. Obviously, such scenario changes if dynamics of the mobiles is considered, shifting back to the time model adopted in Section 4.

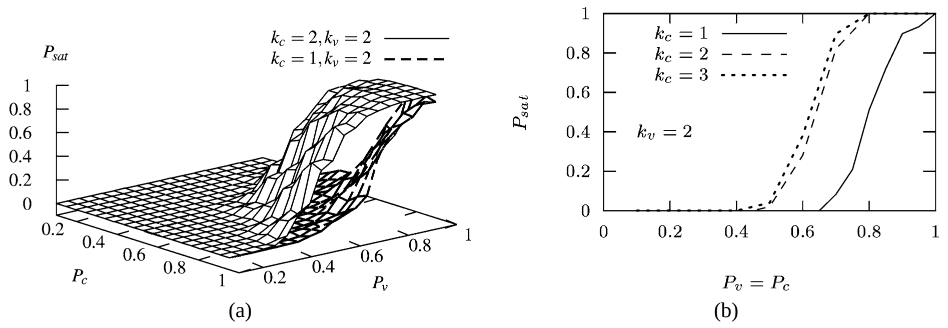


Fig. 9. Percentage of satisfiable instances depending on density parameters for the visibility graph (P_v) and the compatibility graph (P_c). (a) Plot for different values of P_v and P_c. (b) Plot when P_v and P_c are equal.

Given the results provided by Theorems 3 and 4, we consider three hard subclasses of GSensorDCSP, corresponding to $k_v = 2$ and $k_c \in \{1, 2, 3\}$. Fig. 9(a) shows[2] the percentage of satisfiable instances as a function of P_c and P_v for $k_v = 2$ with $k_c = 1$ and $k_c = 2$. As in the case of general SensorDCSP (see Fig. 2), when both probabilities are low, the instances generated are mostly unsatisfiable, while for high probabilities most of the instances are satisfiable. Both for $k_c = 1$ and $k_c = 2$, the transition between the satisfiable and unsatisfiable regions occurs within a narrow range of the density parameters. Observe that, for $k_c = 1$ this range corresponds to significantly higher values of P_c and P_v, comparatively to these for $k_c = 2$ and $k_c = 2$. However, the form of the transition for various values of k_c is very similar (see Fig. 9(b)), showing a similar phase transition behavior for various subclasses of the GSensorDCSP problem with $k_v = 2$.

Consistently with the general SensorDCSP, we observe that the phase transition coincides with the region where the hardest instances occur. For instance, Fig. 10 shows the mean solution time with respect to the density parameters P_v and P_c for the problem instances with 25 sensors, 5 mobiles, $k_c = 1$, and $k_v = 2$. Somewhat less expected result is depicted in Fig. 11 for the case of $k_v = 1$ (and $k_c = 1$), which is shown in Theorem 3 to be polynomial by a reduction to the problem of feasible integral flow in bipartite graphs. Despite the fact that AWC has no explicit connection with the algorithms for the latter problem, Fig. 11(b) shows that these instances are practically easy for AWC as well.

For the second experiment with the AWC algorithm, we consider different sets of instances for several orders of the problem (size of the grid), and several levels of decomposition (visibility and compatibility restrictness). In particular, we consider grids of 25, 36, 49, 64, 81, and 100 sensors ($N = 5, 6, 7, 8, 9, 10$), tracking 5, 7, 9, 12, 15 and 18 mobiles, respectively, giving us an approximately constant ratio between the number of mobiles and the number of sensors for each case. Note that $N = 10$ was the largest problem size we were able to deal with using the CNCL simulator. The restrictness of visibility and compatibility graphs is kept equal ($k_c = k_v = k$), and different sets correspond to k equal 2, 3, 4, and 5. Each set of problem instances corresponding to a particular pair of

[2] The case of $k_v = 2$ and $k_c = 3$ is not depicted in Fig. 9(a) as it is very close to this for $k_v = 2$ and $k_c = 2$ (see Fig. 9(b)).

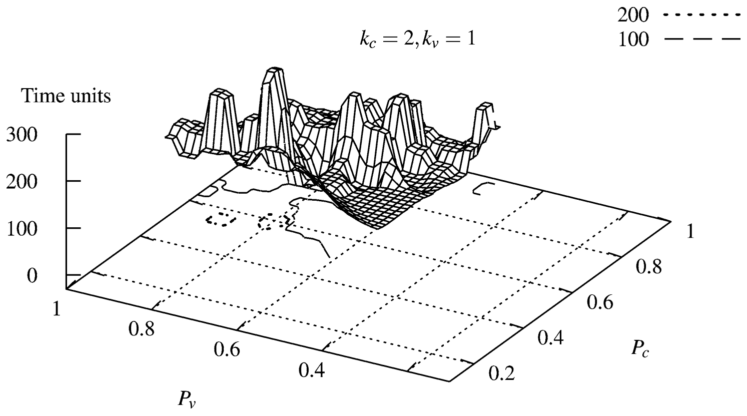
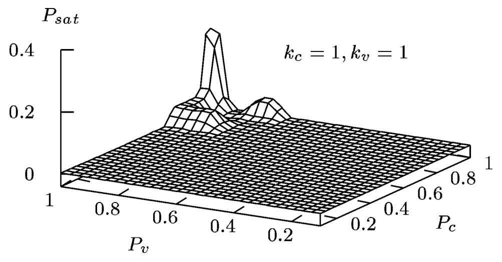
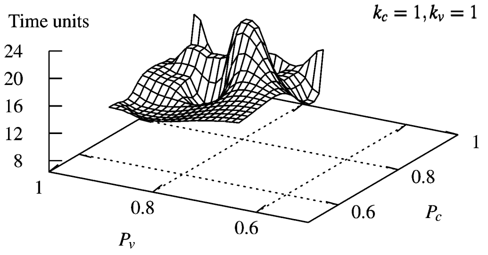


Fig. 10. Mean solution time with respect to P_v and P_c for the AWC algorithm on instances with 25 sensors, 5 mobiles, $k_c = 1$ and $k_v = 2$.



(a)



(b)

Fig. 11. (a) Percentage of satisfiable instances and (b) Mean solution time for the AWC algorithm on (polynomial) instances with 25 sensors, 5 mobiles, $k_c = 1$ and $k_v = 1$.

values (N, k) contains 30 instances. The important point is that all the problem instances, in all the sets (N, k), have been selected from the corresponding phase transition regions with respect to the density parameters P_c and P_v, representing the regions of the hardest problem instances (as it was shown in Figs. 9(a) and 10).[3]

[3] The phase transition regions for every pair (N, k) have been determined in advance.

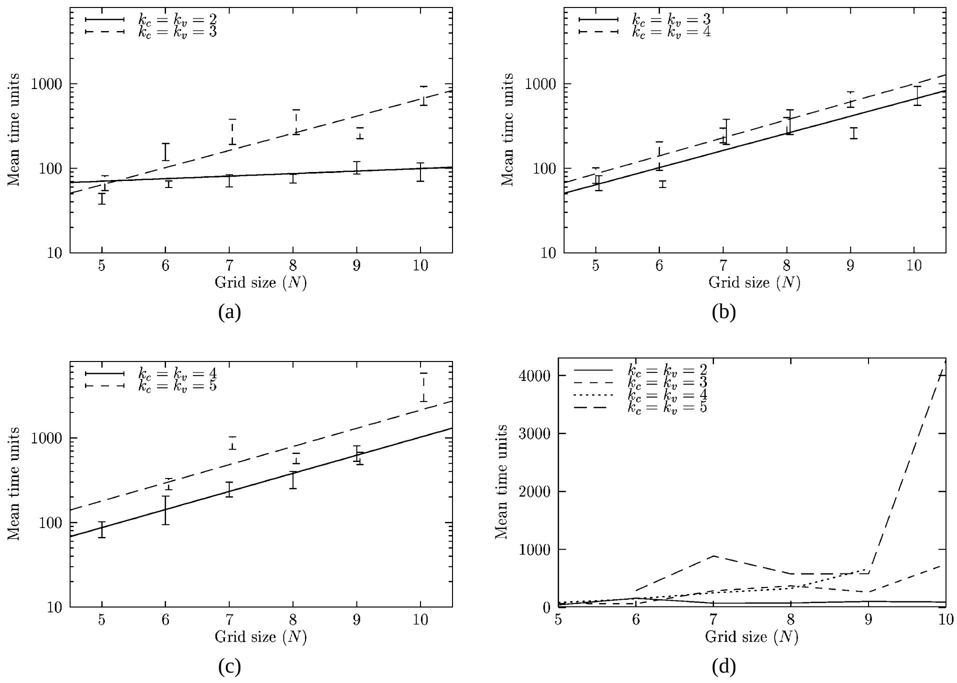


Fig. 12. Mean solution time with respect to the order of the problem (size of the grid) for AWC algorithm on problem instances from the phase transition regions for $k_c = k_v = 2, 3, 4, 5$.

The mean solution time for satisfiable instances in this experiment is plotted in Fig. 12 as a function of N, where Figs. 12(a), (b) and (c) depict this graphs in logarithmic scale for the problem instances with $k = 2, 3$, $k = 3, 4$, and $k = 4, 5$, respectively, while Fig. 12(d) presents the whole picture in the linear scale. We observe that the problem scalability with N degrades dramatically as k increases, but it can be considered as reasonable for $k = 2$ and $k = 3$. In order to capture the exponential behavior of AWC on these problems, Figs. 12(a)–(c) depicts the obtained measures, showing 95% confidence interval of the samples in logarithmic scale, as well as their corresponding linear regression plots. These plots have been represented in three different interrelated pictures in order to facilitate a pair-wise comparison. Two conclusions can be drawn from Fig. 12. First, it is easy to see that the slopes of the regression lines increase with k. For our set of results, the obtained slopes are 0.03, 0.202, 0.213 and 0.293 for $k = \{2, 3, 4, 5\}$, respectively. Second, the exponential dependence of the mean solution time on N seems to fit well according to the experiments. In particular, the obtained mean square error of the regressions is 0.039, 0.04, 0.002 and 0.11 for k equal to 2, 3, 4 and 5, respectively.

7.3. Exploiting solution repairing in dynamic GSensorDCSP

Considering the scalability of the DCSP algorithms on GSensorDCSP, our main concern was about feasibility of striving to optimality in problems with real-life sensor/mobiles settings, where time deadlines play a crucial role, and the objects being tracked are moving.

More formally, the task of a tracking system can be specified as a *dynamic GSensorD-CSP* problem $\boldsymbol{\Pi}$, which consists of an ordered sequence $\Pi_1, \ldots, \Pi_N$ of regular (static) GSensorDCSP problems, that are:

(1) Defined over the same set of sensors $S_i = \{s_1, \ldots, s_m\}$, and having the same compatibility graph,
(2) Possibly differ in their sets of mobile objects and/or visibility graphs, where $T_i = \{t^i_1, \ldots, t^i_{n_i}\}$ is the set of mobiles associated with the problem Π_i, and
(3) Each problem instance should be solved within a certain time window.

Attempting to address this problem, we conducted an experiment with 100 sensors that suppose to track over time a continuously changing set of moving mobiles. The parameters used in this experiment have been chosen to represent a network of radars controlling some part of the airspace. As written, we considered a 10×10 uniform grid of sensors, with the distance of 10 miles between any two adjacent sensors, and the tracking area covered by these sensors is defined by the square of 8100 square miles enclosed by the grid. The compatibility graph of $\boldsymbol{\Pi}$ is 4-restricted, and the visibility graphs of all the sub-problems of $\boldsymbol{\Pi}$ (see below) are 4-restricted as well ($k_c = k_v = 4$).

The mobiles are assumed to move according to some independently chosen linear trajectories, where the velocity of all the mobiles is 2 Mach (1500 miles/hour). Our intention was to keep a controlled, relatively tight ratio between the number of mobiles and the number of sensors, thus we strived to keep the (now expected) number of 18 mobiles inside the grid. On the other hand, we want to model both the mobiles leaving the grid, and the mobiles entering the grid, while keeping the movement of the mobiles independent one of another. To achieve it, we extended the number of mobiles to 36, setting this mobiles to move in (randomly initialized) linear trajectories inside an area larger than our sensor grid. The area is modeled by a square of 16200 square miles (twice as big as the square defined by the grid), and the center of this extended area is exactly the center of the grid. For the first sub-problem Π_1, each mobile is located at a randomly chosen point inside this extended area, and is annotated with a randomly chosen linear trajectory, that will determine the position of this mobile in Π_2 and so on. If, at some point, a mobile reaches the border of the extended area, it *reflects* from the border at a randomly chosen angle, which determines a new linear trajectory for this mobile. Such modeling of the mobile dynamics provides us with a continuously changing set of mobiles inside the grid, while the expected size of this set is known (and is 18 mobiles in our experiment). The time window available to solve each sub-problem Π_i is set to 1.2 seconds, i.e., the minimum time spent by a mobile inside a cell given a speed of 2 Mach, providing us at least 20 snapshots of a mobile during its presence in a particular cell.

Fig. 13 depicts the results for two dynamic GSensorDCSP problems $\boldsymbol{\Pi}_1$ and $\boldsymbol{\Pi}_2$, each consists of 100 static GSensorDCSP sub-problems, where the subproblems for $\boldsymbol{\Pi}_1$ (Figs. 13(a)–(b)) and $\boldsymbol{\Pi}_2$ (Figs. 13(c)–(d)) were selected from the regions of $P_{sat} \approx 0.7 (P_c = P_v = 0.47)$ and $P_{sat} \approx 0.5 (P_c = P_v = 0.45)$, respectively. Recall that $P_{sat} \approx 0.5$ corresponds to the region of the hardest instances. The dashed lines in Figs. 13(a), (c) depict the cumulative probability distributions of solving Π_i within a time window of t

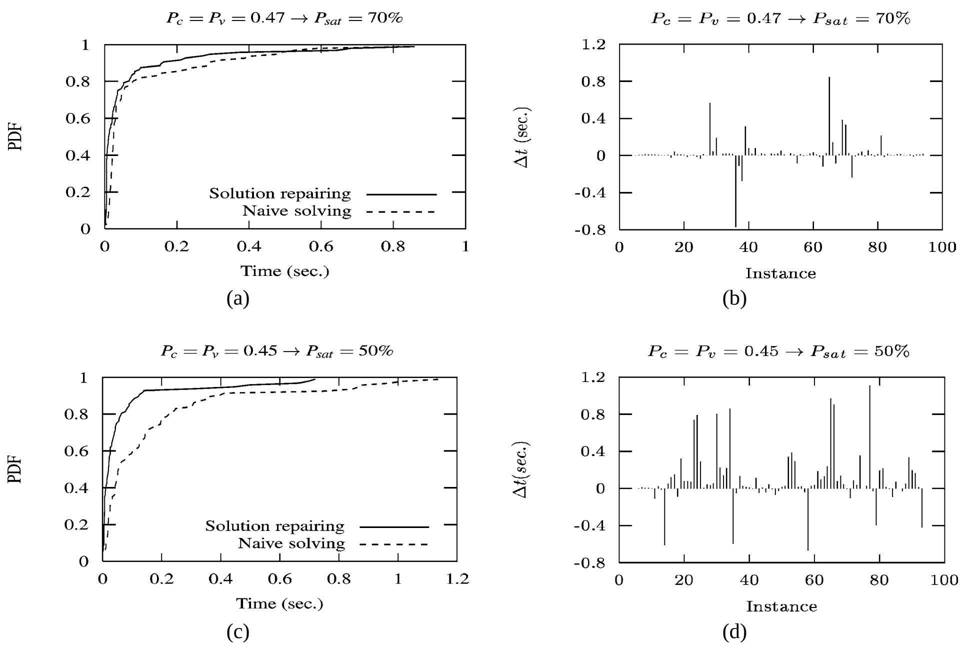


Fig. 13. Dynamics of one problem, located at 70% of satisfiability ratio; (a) show the cumulative probability distributions for the solution repairing and the naive solving approach; (b) plot time differences to solve between the two approaches.

seconds. In $\boldsymbol{\Pi}_1$, all the solvable sub-problems were solved in less than 0.9 seconds, while in $\boldsymbol{\Pi}_2$ all except to one sub-problems were solved within the time limit of 1.2 seconds.

Observe that, if no assumptions can be made about the connection between the mobiles in T_i and T_{i+1}, there is no particular reason to treat $\boldsymbol{\Pi}$ differently than just solving its static sub-problems $\Pi_1, \ldots, \Pi_N$ one by one *independently*, using one of the DCSP algorithms. In what follows, we refer to this approach as to *naive* solving of dynamic CS problems, and the results depicted by the dashed lines in Figs. 13(a), (c) correspond to this straightforward approach. However, mobile dynamics are typically far from being chaotic (linear trajectories in our experiment), i.e., the changes between the subsequent sub-problems are governed by some clear model of mobile dynamics. For instance, consider a network of radars controlling some airspace region. In such an application, it is reasonable to assume that if an aircraft becomes trackable by a sensor, then this aircraft is likely to remain trackable by this sensor in some near future.

One of our hypotheses was that continuity of mobiles movement can be exploited in improving the performance of the tracking systems. An approach that a priori seems to be promising for dealing with such a problem $\boldsymbol{\Pi} = \{\Pi_1, \ldots, \Pi_N\}$ is to initialize the search for Π_i, $1 < i \leqslant N$, by the solution already achieved for Π_{i-1} (comparatively to starting from an random assignment in AWC used in the naive approach). In what follows, we refer to this approach as to *solution repairing*. Note that in this approach, nogoods are not kept and are removed once a solution is obtained, so no additional synchronization is required between agents.

The central question is whether the contribution of solution repairing (versus the naive approach) is expected to be significant in real-life settings of both the mobiles dynamics, and the time available to solve each one of the static sub-problems. One experiment provides a positive evidence to this question: The solid lines in Figs. 13(a)–(c) depict the cumulative probability distributions of solving Π_i within a time window of t seconds using the solution repairing approach. It is easy to see that solution repairing clearly outperforms the naive approach, and Figs. 13(b)–(d) illustrate this even better: For each sub-problem Π_i, these graphs plot the difference between the times required to solve Π_i using AWC from scratch and starting from the solution for P_{i-1}, if this exists (Δt). More interestingly, the results of our experiment show that the *relative attractiveness of solution repairing is higher in the region of harder instances*. For instance, using solution repairing, all the sub-problems of $\boldsymbol{\Pi}_2$ were solved in less than 0.75 second. The reason could be that small changes in the problem setting (as the changes between Π_i and Π_{i+1} are expected to be) usually will not change significantly the placement of the solutions in the search tree. If so, then adopting solution repairing is likely to initialize the search at a node that is close to a solution node in the search tree. Likewise, the contribution of this property is likely to be more significant for sequences of harder problems, i.e., problems that a priori have less alternative solutions.

8. Conclusions

We introduced SensorDCSP, a benchmark that captures some of the characteristics of real-world distributed applications that arise in the context of distributed networked systems. The two control parameters of our SensorDCSP generator, sensor compatibility (P_c) and sensor visibility (P_v), result in a zero-one phase transition in satisfiability. We tested two complete DisCSP algorithms, synchronous backtracking (ABT) and asynchronous weak commitment search (AWC). We show that the phase transition region of SensorDCSP induces an easy-hard-easy profile in the solution time, both for ABT and AWC, which is consistent with CSPs. We found that AWC performs much better than ABT on satisfiable instances, but worse on unsatisfiable instances. This differential in performance is most likely due to the fact that on unsatisfiable instances, the dynamic priority ordering of AWC slows the completion of the search process.

In order to study the impact of different network traffic conditions on the performance of the algorithms, we used a discrete-event network simulator. We found that random delays can improve the performance and robustness of AWC. On hard satisfiable instances, however, the performance of the basic ABT deteriorates dramatically when subject to random link delays. However, we developed a decentralized dynamic restart strategy for ABT, which results in an improvement and shows robustness with respect to the variance in link delays. Most interestingly, our results also show that the active introduction of message delays by agents can improve performance and robustness while reducing the overall network load. These results validate our thesis that when considering networking applications of DisCSP, one cannot afford to neglect the characteristics of the underlying network conditions. The network-level behavior can have an important, algorithm-specific, impact on solution time.

In a more focused attempt to study practical applicability of the DisCSP algorithms for various distributed tracking systems, we introduced and analyzed GSensorDCSP, a variant of SensorDCSP that induces a spatial structure on the problem constraints. We performed both analytical analysis of GSensorDCSP and an empirical study of DisCSP algorithms on various instances of this problem. First, we analyzed the performance of AWC on the tractable subclasses of GSensorDCSP, showing that AWC scales perfectly on such instances. Next we have tested scalability of AWC on various hard subclasses of GSensorDCSP. While in general GSensorDCSP induces phase-transition in satisfiability with respect to the density parameters of the problem, we showed that AWC scales nicely on a wide (a priori hard) subclass of GSensorDCSP. Finally, we analyzed the AWC algorithm on a sequence of GSensorDCSP problems that represent a system of sensors tracking a set of moving objects. We discussed some properties of such a dynamic GSensorDCSP, and showed how these properties could be exploited in the dynamic tracking systems.

We believe that our study makes it clear that DisCSP algorithms are best tested and validated on benchmarks based on real-world problems, using network simulators. We hope our benchmark domains will be of use for the further analysis and development of DisCSP methods.

Appendix A. Proofs

Theorem 3. *Any GSensorDCSP problem instance with* 1*-restricted visibility graph* $\mathcal{V}$ *is solvable in polynomial time.*

Proof. Consider such a problem instance Π, where S and T stand for the sets of sensors and mobiles in Π, respectively. Let $\mathsf{cell}(t_i)$ denote the cell of the grid in which object t_i is located. First, given that, for every object $t_i \in T$, we have $\mathcal{V}(t_i) \subseteq \mathbb{V}^1(t_i)$, we determine the following disjoint partition of T into $T = T' \cup T''$:

(i) $t_i \in T'$ if and only if can be potentially tracked by any possible (out of four) triplet of sensors from $\mathbb{V}^1(t_i)$, and
(ii) $t_i \in T''$ if and only if there exists a pair of sensors in $\mathsf{cell}(t_i)$ $(= \mathbb{V}^1(t_i))$ that have to be a part of any solution for t_i in Π.

To show feasibility of such a partition, consider a mobile $t_i \in T$. If the compatibility graph restricted to the four sensors in $\mathbb{V}^1(t_i)$ is complete (i.e., all four sensors of $\mathsf{cell}(t_i)$ can communicate one with each other), then we have $t_i \in T'$, and this corresponds to locally complete compatibility of sensors relevant to t_i. Alternatively, if the communication between the sensors in $\mathbb{V}^1(t_i)$ is not complete, then there exist at least one pair of mutually incompatible sensors $s, s' \in \mathbb{V}^1(t_i)$. Clearly, these two sensors cannot be simultaneously a part of a solution for t_i. Therefore, two other sensors $\mathbb{V}^1(t_i) \setminus \{s, s'\}$ have to be a part of any solution for t_i in Π, if one exists, and thus we have $t_i \in T''$. Clearly, this partition of T can be performed in linear time.

Observe that, for each object $t_i \in T''$, the pair of sensors $\mathbb{V}^1(t_i) \setminus \{s, s'\}$ can be *preassigned* to t_i. Now, let $S' \subseteq \{s_1, \ldots, s_m\}$ be the set of sensors that were not preassigned to the objects in the first stage. First, we construct a directed graph $G = (S' \cup T, E)$, such that there is an edge from a sensor $s \in S'$ to an object $t \in T$ if and only if $s \in \mathbb{V}^1(t)$. Likewise, we add a super-source node **s**, and for each sensor $s \in S'$ we put an edge from **s** to s. Similarly, we add a super-sink node **t**, and for each object $t \in T$ we add an edge from t to **t**.

Now we construct a feasible integer flow problem based on this graph. For each edge $(\mathbf{s}, s)$ and for each edge (s, t) we place a lower bound flow of 0 and an upper bound flow of 1. For each edge $(t, \mathbf{t})$, if $t \in T'$, then we place a lower bound flow of 3 and an upper bound flow of 3. Otherwise, if $t \in T''$, we place a lower bound flow of 1 and an upper bound flow of 1. It is easy to see that our GSensorDCSP problem Π has a solution if and only if this directed graph has a feasible flow of $3 \cdot |T'| + |T''|$. As feasible-flow problems are known to be polynomial-time solvable [6], so is this special case of GSensorDCSP. □

Theorem 4. *GSensorDCSP with* 2*-restricted visibility is* NP*-complete.*

Proof. The membership in NP is straightforward, since the general SensorDCSP is in NP. The proof of hardness is by reduction from 3-SAT. Let $\mathcal{F}$ be a 3-cnf formula specified by the clauses $\{c_1, \ldots, c_m\}$ over the variables $\{x_1, \ldots, x_n\}$. An equivalent GSensorDCSP problem $\Pi_{\mathcal{F}}$ with 2-restricted visibility and 1-restricted compatibility can be constructed as follows.

The grid of $\Pi_{\mathcal{F}}$ is an $(4n + 2) \times (9m + 3)$ grid of sensors; in what follows, by the rows and columns of the grid we refer to the rows and columns of the grid cells, in terms of which we have an $(4n + 1) \times (9m + 2)$ grid of cells. Except for the first row, the rows of the grid can be considered in ordered quadruples, where the ith quadruple of rows corresponds to the variable x_i in $\mathcal{F}$. For simplicity of presentation, we denote the rows of the grid as

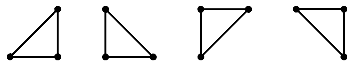
$$0_0, 1_0, 1_1, 1_2, 1_3, 2_0, 2_1, 2_2, 2_3, \ldots, n_0, n_1, n_2, n_3.$$

For an illustration, see Fig. A.3. Similarly, the columns of the grid (except for the first and the last ones), can be considered in ordered nine-tuples

$$0_0, 1_0, \ldots, 1_8, 2_0, \ldots, 2_8, \ldots, m_0, \ldots, m_8, (m + 1)_0$$

where the jth nine-tuple of columns corresponds to the clause c_j in $\mathcal{F}$.

Since we construct $\Pi_{\mathcal{F}}$ with 1-restricted compatibility between the sensors, every triplet of mutually compatible sensors in $\Pi_{\mathcal{F}}$ has to lie on the vertexes of a certain grid cell. It is easy to see that such a triplet of sensors will form a cell-embedded triangle of one of the four forms:



To illustrate the notation we are using, the cell formed by the row i_3 and the column j_5 is denoted by (i_3, l_5), and $\langle i_3, l_5, B\rangle$ denotes the fact that the three sensors forming the triangle of type B in cell (i_3, l_5) are mutually compatible.

First, for each variable $x_i \in \mathcal{F}$, the compatibility of the sensors in rows $i_0, \ldots, i_3$ is defined by the following template (for an illustration see Fig. A.1, where triangles depict the compatible triplets of sensors):

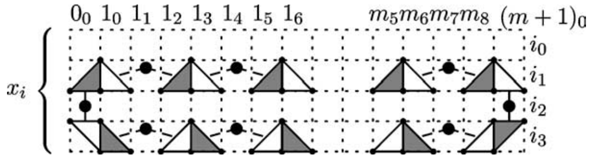


Fig. A.1. Mobiles, sensors, compatibility and visibility for a variable $x_i \in \mathcal{F}$.

- In rows i_0 and i_2 we have no compatible triplets of sensors.
- In row i_1, for all $1 \leqslant l \leqslant m$, we have $\langle i_1, 0_0, A\rangle$, $\langle i_1, l_0, B\rangle$, $\langle i_1, l_2, A\rangle$, $\langle i_1, l_3, B\rangle$, $\langle i_1, l_5, A\rangle$, $\langle i_1, l_6, B\rangle$, $\langle i_1, l_8, A\rangle$, and $\langle i_1, (m+1)_0, B\rangle$.
- In row i_3, for all $1 \leqslant l \leqslant m$, we have $\langle i_3, 0_0, D\rangle$, $\langle i_3, l_0, B\rangle$, $\langle i_3, l_2, A\rangle$, $\langle i_3, l_3, B\rangle$, $\langle i_3, l_5, A\rangle$, $\langle i_3, l_6, B\rangle$, $\langle i_3, l_8, A\rangle$, and $\langle i_3, (m+1)_0, C\rangle$.

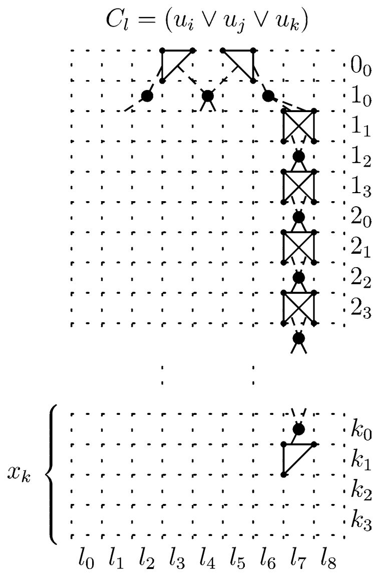
For each variable $x_i \in \mathcal{F}$, we have $6m + 2$ mobiles, located and visible by the sensors according to the following template (see Fig. A.1, where dashed lines connect between the mobiles and the compatible triplets of sensors that can track these mobiles):

- The first $3m$ mobiles are located in row i_1, in cells (i_1, l_1), (i_1, l_4), (i_1, l_7), $1 \leqslant l \leqslant m$, where the mobile in cell (i_1, l_k), $k \in \{1, 4, 7\}$, can be tracked only by the two sensor triplets $\langle i_1, l_{k-1}, B\rangle$ and $\langle i_1, l_{k+1}, A\rangle$.
- The next $3m$ mobiles are located similarly in row i_3, in the cells (i_3, l_1), (i_3, l_4), (i_3, l_7), $1 \leqslant l \leqslant m$, where again the mobile in cell (i_3, l_k), $k \in \{1, 4, 7\}$, can be tracked only by the two sensor triplets $\langle i_3, l_{k-1}, B\rangle$ and $\langle i_3, l_{k+1}, A\rangle$.
- The last two mobiles are located in row i_2, in cells $(i_2, 0_0)$ and $(i_2, (m+1)_0)$. The mobile in $(i_2, 0_0)$ can be tracked only by the two sensor triplets $\langle i_1, 0_0, A\rangle$ and $\langle i_3, 0_0, D\rangle$, and the mobile in $(i_2, (m+1)_0)$ can be tracked only by the two sensor triplets $\langle i_1, (m+1)_0, B\rangle$ and $\langle i_3, (m+1)_0, C\rangle$.

Observe that the mobiles corresponding to a particular variable x_i are “circularly” constrained: if all these mobiles are tracked, then either they are all tracked by the white sensor triangles, or they are all tracked by the shadowed sensor triangles. Intuitively, this construction will ensure that x_i takes the same value with respect to all the clauses in $\mathcal{F}$.

Now we extend both the set of the mobiles, and the compatibility between the sensors with respect to the clauses of $\mathcal{F}$. For each clause $c_l = (u_i, u_j, u_k)$, $i < j < k$, where each $u_t \in \{x_t, \overline{x_t}\}$, the construction is as follows (see Fig. A.2):

- In the special row 0_0, we have $\langle 0_0, l_3, C\rangle$ and $\langle 0_0, l_5, D\rangle$.
- For each literal $u_t \in c_l$, let h be 1, 4, or 7, when u_t is the first, second and third literal of c_l, respectively. Now, for each literal $u_t \in c_l$:
  - For $1 \leqslant s \leqslant t-1$, we have $\langle s_1, l_h, C\rangle$, $\langle s_1, l_h, D\rangle$, $\langle s_3, l_h, C\rangle$, and $\langle s_3, l_h, D\rangle$.
  - If $u_t = x_t$, then we have $\langle t_1, l_h, D\rangle$. Otherwise, if $u_t = \overline{x_t}$, we have $\langle t_1, l_h, C\rangle$.
  - For $1 \leqslant s \leqslant t-1$, we have two mobiles located in the cells (s_2, l_h) and $((s+1)_0, l_h)$. Each one of these two mobiles can be tracked only by all (one or two) compatible



$C_l = (u_i \vee u_j \vee u_k)$
0_0
1_0
1_1
1_2
1_3
2_0
2_1
2_2
2_3
x_k
k_0
k_1
k_2
k_3
l_0 l_1 l_2 l_3 l_4 l_5 l_6 l_7 l_8

Fig. A.2. Mobiles, compatible triplets of sensors, and visibility constraints in the construction corresponding to a clause $c_l \in \mathcal{F}$. In this case, $u_k = \overline{x_k}$.

triplets of sensors in the cell immediately above it, and by all (one or two) compatible triplets of sensors in the cell immediately below it.

- To accomplish the construction, in row 1_0 we have three mobiles in cells $(1_0, l_2)$, $(1_0, l_4)$, and $(1_0, l_6)$. The mobile in cell $(1_0, l_2)$ can be tracked only by the sensor triplet $\langle 0_0, l_3, C\rangle$, and by all (one or two) compatible triplets of sensors in cell $(1_1, l_1)$. The mobile in cell $(1_0, l_4)$ can be tracked only the sensor triplets $\langle 0_0, l_3, C\rangle$ and $\langle 0_0, l_5, D\rangle$, and by all (one or two) compatible triplets of sensors in cell $(1_1, l_4)$. Finally, the mobile in cell $(1_0, l_6)$ only by the sensor triplet $\langle 0_0, l5, D\rangle$ and by all (one or two) compatible triplets of sensors in cell $(1_1, l_7)$.

The construction is clearly polynomial-time since the number of sensors on the grid is $(4n + 2) \times (9m + 3)$, and the number of mobiles is $\Theta(n(6m + 2) + 3m(2n - 1))$. Fig. A.3 illustrates the construction on a small example of $\mathcal{F} = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$.

Our claim is that a 3-cnf formula $\mathcal{F}$ is satisfiable if and only if there exist a solution to the corresponding GSensorDCSP $\Pi_{\mathcal{F}}$.

($\Rightarrow$) First we show that if $\mathcal{F}$ is satisfiable, then there is a solution for $\Pi_{\mathcal{F}}$. Let ϕ be an assignment to $\{x_1, \ldots, x_n\}$ satisfying $\mathcal{F}$. First, for each variable x_i, we assign the $6m + 2$ mobiles associated with x_i (see the first stage of the construction) to be tracked consistently with $\phi(x_i)$: These mobiles are tracked by the white (shadowed) compatible triplets of sensors if $\phi(x_i) = \mathit{true}$ ($\phi(x_i) = \mathit{false}$), respectively.

Since ϕ is a satisfying assignment, let $u_{l_i} \in \phi$ be a literal satisfying the clause c_l. Consider the vertical sequence of mobiles associated with the literal u_{l_i} in the column l_h (plus one sensor in the row 1_0), and, in particular, consider the lowest such mobile. By the construction, this mobile can be tracked by the compatible triplet of sensors in cell $((l_i)_1, l_h)$,

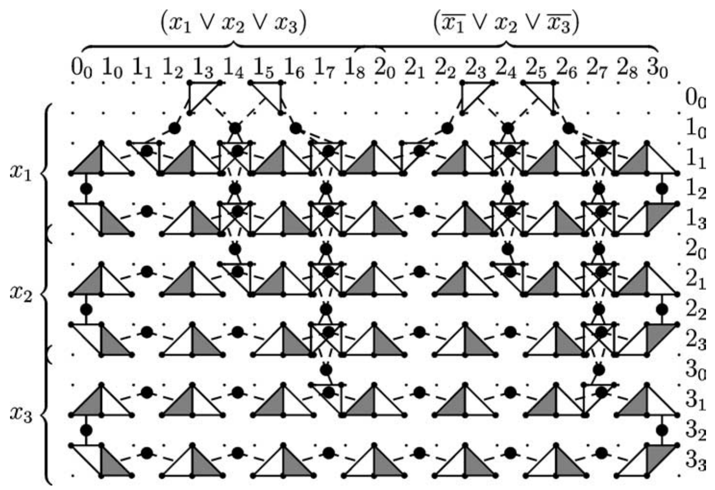


Fig. A.3. Construction of $\Pi_{\mathcal{F}}$ for $\mathcal{F} = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$.

since no sensor in this triplet has been assigned to track mobiles associated with x_{l_i}. By the same inductive argument, every sensor in this vertical sequence can be tracked by a compatible triplet of sensors located immediately *below* it. Therefore, the remaining two mobiles associated with c_l in row 1_0 can be tracked by the sensor triplets $\langle 0_0, l_3, C\rangle$ and $\langle 0_0, l_5, D\rangle$. Hence, all the mobiles are assigned to be tracked by pair-wise disjoint, compatible triplets of sensors, and thus $\Pi_{\mathcal{F}}$ is solved.

($\Leftarrow$) Now we show that if $\Pi_{\mathcal{F}}$ is solvable, then $\mathcal{F}$ is satisfiable. It is easy to see that, if there exist a solution for $\Pi_{\mathcal{F}}$, then, for each clause $c_l \in \mathcal{F}$, there exist at least one literal $u_{l_i} \in c_l$, such that the corresponding mobile in the row 1_0 is tracked by a triplet of compatible sensors located immediately *below* it in the column l_h. In turn, this entails that all the mobiles in the corresponding vertical sequence corresponding to the associated with the literal u_{l_i} will have to be tracked by the sensor triplets located immediately below them.

Given the set of m literals $\{u_{1_i}, \ldots, u_{m_i}\}$ as above, we claim that the assignment $\phi = \bigwedge_{j=1}^{m} u_{j_i}$ satisfies $\mathcal{F}$. Since each literal u_{l_i} satisfies the corresponding clause c_l, the only thing remains to be shown in order to prove the claim is that if $u_{l_i} \in \phi$, then $\overline{u_{l_i}} \notin \phi$. However, this is apparent from the construction of $\Pi_{\mathcal{F}}$, and the choice of u_{l_i} for c_l: Suppose that u_{l_i} corresponds to $x_{l_i} = \mathit{true}$. In this case, u_{l_i} will eliminate a shadowed triangle of sensors in the row $(l_i)_1$, and $\overline{u_{l_i}}$ will eliminate a white triangle from the same row. It is easy to see that in this case some of the mobiles in the rows $(l_i)_1$, $(l_i)_2$, and $(l_i)_3$ will remain untracked, which contradicts our assumption that there is a solution for $\Pi_{\mathcal{F}}$. □

References

[1] AFRL/IFTB, Autonomous Negotiating Teams (ANTs) Program, http://www.rl.af.mil/div/IFT/IFTB/ants/ants.html.

[2] R. Béjar, B. Krishnamachari, C. Gomes, B. Selman, Distributed constraint satisfaction in a wireless sensor tracking system, in: Proceedings of the IJCAI-01 Workshop on Distributed Constraint Reasoning, Seattle, WA, 2001.

[3] C. Bessière, A. Maestre, P. Meseguer, Distributed dynamic backtracking, in: Proceedings of the IJCAI-01 Workshop on Distributed Constraint Reasoning, Seattle, WA, 2001, pp. 9–16.

[4] M. Calisti, B. Faltings, Distributed constrained agents for allocating service demands in multi-provider networks, J. Italian Oper. Res. Soc. XXIX (91) (2000), Special Issue on Constraint-Based Problem Solving.

[5] S.E. Conry, K. Kuwabara, V.R. Lesser, R.A. Meyer, Multistage negotiation for distributed constraint satisfaction, IEEE Trans. Systems Man Cybernet. (Special Section on DAI) 21 (6) (1991) 1462–1477.

[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 1990.

[7] M. Crovella, A. Bestavros, Self-similarity in world wide web traffic: evidence and possible causes, IEEE Trans. Networking 5 (6) (1997) 835–846.

[8] C. Fernández, R. Béjar, B. Krishnamachari, C.P. Gomes, Communication and computation in distributed CSP algorithms, in: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP-2002), 2002, pp. 664–679.

[9] C.P. Gomes, B. Selman, H.A. Kautz, Boosting combinatorial search through randomization, in: Proceedings of the National Conference on Artificial Intelligence (AAAI-98), Madison, WI, 1998, pp. 431–437.

[10] Y. Hamadi, C. Bessièere, J. Quinqueton, Backtracking in distributed constraint networks, in: Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98), 1998, pp. 219–223.

[11] M. Hannebauer, A formalization of autonomous dynamic reconfiguration in distributed constraint satisfaction, Fundamenta Informaticae 43 (1–4) (2000) 129–151.

[12] M. Helmert, Complexity results for standard benchmark domains in planning, Artificial Intelligence 143 (2) (2003) 219–262.

[13] J. Hoffmann, Local search topology in planning benchmarks: an empirical analysis, in: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01), Seattle, WA, 2001, pp. 453–458.

[14] J. Hoffmann, Local search topology in planning benchmarks: a theoretical analysis, in: Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02), 2002, pp. 379–387.

[15] B. Horling, R. Vincent, R. Mailler, J. Shen, R. Becker, K. Rawlins, V. Lesser, Distributed sensor network for real time tracking, in: Proceedings of the Fifth International Conference on Autonomous Agents, 2001, pp. 417–424.

[16] M. Junius, M. Büter, D. Pesch, et al., CNCL—Communication Networks Class Library, Aachen University of Technology, 1996.

[17] D. Kirkpatrick, P. Hell, On the complexity of general graph factor problems, SIAM J. Comput. 12 (3) (1983) 601–608.

[18] B. Krishnamachari, Phase transitions, structure, and complexity in wireless networks, PhD Thesis, Electrical Engineering, Cornell University, Ithaca, NY, 2002.

[19] B. Krishnamachari, R. Béjar, S.B. Wicker, Distributed problem solving and the boundaries of self-configuration in multi-hop wireless networks, in: Hawaii International Conference on System Sciences (HICSS-35), 2002.

[20] W.E. Leland, M.S. Taqqu, W. Willinger, D.V. Wilson, On the self-similar nature of ethernet traffic (Extended version), IEEE Trans. Networking 2 (1) (1994) 1–15.

[21] S. Macho-Gonzalez, B. Faltings, Open constraint satisfaction, in: Proceedings of the AAMAS-02 Workshop on Distributed Constraint Satisfaction, 2002.

[22] P. Marti, M. Rueher, A distributed cooperating constraints solving system, Internat. J. Artificial Intelligence Tools 4 (1–2) (1995) 93–113.

[23] A. Meisels, E. Kaplansky, Scheduling agents—distributed timetabling problems (DisTTP), in: Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT-02), 2002, pp. 182–190.

[24] A. Meisels, I. Razgon, Comparing performance of distributed constraints processing algorithms, in: Proceedings of the AAMAS-2002 Workshop on Distributed Constraint Reasoning, 2002, pp. 86–93.

[25] P.J. Modi, H. Jung, M. Tambe, W. Shen, S. Kulkarni, A dynamic distributed constraint satisfaction approach to resource allocation, in: Proceedings of International Conference on Principles and Practices of Constraint Programming (CP-2001), 2001, pp. 685–700.

[26] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, L. Troyansky, Determining computational complexity from characteristic phase transitions, Nature 400 (1999) 133–137.
[27] T. Nguyen, Y. Deville, A distributed arc-consistency algorithm, Sci. Comput. Programming 30 (1–2) (1998) 227–250.
[28] V. Paxson, S. Floyd, Wide area traffic: the failure of Poisson modeling, IEEE/ACM Trans. Networking 3 (3) (1995) 226–244.
[29] G. Samorodnitsky, M.S. Taqqu, Stable Non-Gaussian Random Processes, Chapman & Hall, 1994.
[30] SATLIB—The Satisfability Library, http://www.satlib.org/index-ubc.html (created and maintained by H.H. Hoos and T. Stützle).
[31] M. Silaghi, D. Sam-Haroud, B. Faltings, Maintaining hierarchical distributed consistency, in: Proceedings of the CP-00 Workshop on Distributed Constraint Satisfaction, 2000.
[32] K. Sycara, S. Roth, N. Sadeh, M.S. Fox, Distributed constrained heuristic search, IEEE Trans. Systems Man Cybernet. 21 (6) (1991) 1446–1461.
[33] T. Walsh, From P to NP: COL, XOR, NAE, 1-in-k, and Horn SAT, in: Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02), Edmonton, AB, 2002, pp. 695–700.
[34] M. Yokoo, Weak-commitment search for solving constraint satisfaction problems, in: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94), Seattle, WA, 1994, pp. 313–318.
[35] M. Yokoo, Asynchronous weak-commitment search for solving distributed constraint satisfaction problems, in: Proceedings of the First International Conference on Principles and Practice of Constraint Programming (CP-95), 1995, pp. 88–102.
[36] M. Yokoo, E.H. Durfee, T. Ishida, K. Kuwabara, Distributed constraint satisfaction for formalizing distributed problem solving, in: Proceedings of the Twelfth IEEE International Conference on Distributed Computing Systems, 1992, pp. 614–621.
[37] M. Yokoo, E.H. Durfee, T. Ishida, K. Kuwabara, The distributed constraint satisfaction problem: formalization and algorithms, IEEE Trans. Knowledge Data Engrg. 10 (5) (1998) 673–685.
[38] M. Yokoo, K. Hirayama, Algorithms for distributed constraint satisfaction: a review, Autonomous Agents Multi-Agent Syst. 3 (2) (2000) 198–212.