A Novel Constraint Satisfaction Problem Solver for Self-Configuring Distributed Systems with Highly Dynamic Behavior

Benjamin Satzger, Faruk Bagci, and Theo Ungerer

***Abstract*— The increasing complexity of distributed computer systems requires new control mechanisms. The behavior of future systems should be defined by high-level goals, with the system itself being responsible to maintain them. This paper proposes to use the constraint satisfaction problem (CSP) paradigm to realize such self-configuring systems. This allows to specify the desired system behavior as constraints and generic domain independent algorithms can be used to enforce these constraints. We present a novel algorithm called System-Driven Search (SDS) for maintaining constraints in highly dynamic distributed environments, like wireless sensor networks. It is not susceptible to message loss and piggybacking may be used for information dissemination instead of sending explicit messages. These features make SDS suitable especially for decision making tasks like self-configuration in battery-operated wireless sensor networks. Partitioning into coordinating cliques and channel allocation, two typical problems in that area, are used to evaluate the proposed algorithm.**

Fig. 1. Principal states and territories of Australia

I. Introduction

Constraint satisfaction problems (CSPs) [5] are subject to research in the areas of artificial intelligence and operations research. They are defined as a set of objects that must satisfy a number of constraints. CSPs conform to a simple standardized representation and allow the application of generic, domain-independent solution algorithms.

Distributed constraint satisfaction problems (DCSPs, also DisCSPs) represent a formalism for describing a problem that involves multiple participants, called agents. DCSPs were first investigated by Yokoo et al. [9]. They can be used to model many real-world problems with distributed nature, such as meeting scheduling problems [7] and self-configuration in networks [3]. DCSPs allow to formulate distributed problems, like configuration problems in wireless sensor networks (WSNs), in a standardized comprehensible way. Off-the-shelf algorithms can be used to solve them. This helps to focus on the problem itself and avoids developing single purpose algorithms for every particular problem.

Current modern DCSP solvers have difficulty in coping with highly dynamic environments and message loss. In this paper we present a new algorithm, which tries to maintain a best-effort solution in constantly changing environments with lossy communication like WSNs. For evaluation purposes we apply SDS to typical configuration problems in WSNs: Partitioning into coordinating cliques, which arises if collections of sensor nodes are responsible for the joint execution of tasks, and channel allocation for conflict free distribution of channels.

The paper is organized in six sections. Section II gives a short overview of classical constraint satisfaction problems, while Section III introduces distributed constraint satisfaction problems and related algorithms to solve them. Then, Section IV describes the proposed algorithm and Section V presents evaluation results. Finally, Section VI concludes the paper.

II. Constraint satisfaction problems

A CSP is defined by a set of variables, $V = \{X_1, X_2, \ldots, X_n\}$, and a set of constraints, $C = \{C_1, C_2, \ldots, C_m\}$. Each variable X_i has a nonempty domain D_i of possible values. Each constraint C_i involves some subset of the variables and specifies the allowable combinations of values for that subset. A state of the problem is defined by an assignment of values to some or all of the variables, $\{X_i = v_i, X_j = v_j, \ldots\}$. An assignment that does not violate any constraint is called *consistent* or *legal*. A *complete* assignment is one in which every variable is mentioned; a solution to a CSP is a complete and consistent assignment. The following example of a CSP has been taken from [5]. Figure 1 shows a map of the states and territories of Australia. The task, which is to be modeled by a CSP, is coloring each region either red, green, or blue, in such a way that no neighboring regions have the same color. To model this example as a CSP, the variables are set to the regions $\{WA, NT, QLD, NSW, VIC, SA, TAS\}$. The domain of each variable is $\{red, green, blue\}$. Based on these definitions, the coloring problem can be expressed by the following set of constraints: $\{WA \neq NT, WA \neq$

Benjamin Satzger is with the International Computer Science Institute, 1947 Center Street, Berkeley, CA 94704, USA (phone: +1 510 666 2900; email: satzger@icsi.berkeley.edu).

Faruk Bagci and Theo Ungerer are with the Department of Computer Science, University of Augsburg, Universitätsstraße 6a, 86159 Augsburg, Germany (phone: +49 821 598 2351; email: {bagci,ungerer}@informatik.uni-augsburg.de).

978-1-4244-8126-2/10/$26.00 ©2010 IEEE

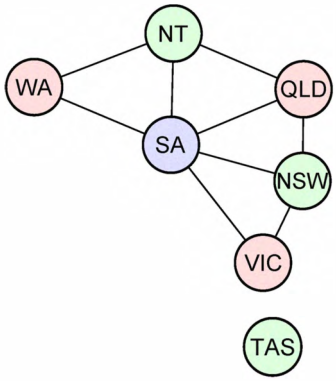


Fig. 2. Map-coloring as constraint graph. One possible solution to the CSP is indicated by the nodes' colors.

$SA, NT \neq SA, NT \neq QLD, QLD \neq SA, QLD \neq NSW, NSW \neq VIC, SA \neq VIC, SA \neq NSW\}$. As shown in Figure 2, a CSP can be visualized as a constraint graph. The vertices of such a graph correspond to variables and the edges correspond to constraints. A possible solution to this problem would be: $\{WA = red, NT = green, QLD = red, NSW = green, VIC = red, SA = blue, TAS = green\}$.

III. Distributed constraint satisfaction problems

A DCSP is a generalization of a CSP that distributes variables among agents $A = \{A_1, \ldots, A_l\}$, i.e., each agent *owns* some variables. Constraints based on variables of a single agent are called *intra-agent* constraints, *inter-agent* constraints involve variables of different agents. DCSPs have been first discussed in [6], [9], as a formalization of a generic way for distributed problem solving. Solving a DCSP requires agents to communicate, interact, and negotiate via message passing.

The most trivial approach to solve a DCSP is to select a leader agent, which is responsible to collect all information about variables, their domains, and all constraints. In this case the leader is able to use a standard centralized CSP algorithm to solve the DCSP. This approach, however, suffers from two major issues [10]. Firstly, it can be very costly to collect all information about the DCSP and secondly, it may be undesirable for security and privacy reasons to allow one agent to gather all information.

Synchronous backtracking adapts the standard backtracking algorithm for CSPs to a distributed setting. In the run-up, all agents must agree on an order on the variables. When an agent owning the variable X_i receives information about the instantiation of the variables $X_1, \ldots, X_{i-1}$, it searches for a consistent value for X_i. If such a value is found, the agent forwards the variables $X_1, \ldots, X_i$ and their current values to the owner of variable X_{i+1}. If no consistent value can be found, a backtracking message is sent to the previous agent. The major drawback of this approach lies in its sequential nature. Only one agent is active at a particular point in time while all others are waiting. The establishment of an order on the variables introduces additional overhead.

The probably most well known algorithm for solving DCSPs was introduced by Yokoo et al. [9]. The authors introduce the asynchronous backtracking (ABT) algorithm, an asynchronous implementation of a backtracking search. Without loss of generality, for the sake of simplicity, they assume that each agent has exactly one variable, all constraints are binary, and each agent knows all constraints relevant to its variable. In an initial phase, ABT establishes an order on all agents. Each agent assigns a random value to its variable and broadcasts it to all neighbors with lower priority. The receiving agents store this information in a container called *agentview*. If its own assignment is not consistent with the agentview, the agent tries to change it to make it consistent. If this is impossible, a backtracking is initiated by sending a *nogood* message to a higher priority agent. This roughly described algorithm is considered to be one of the most efficient approaches to solve DCSPs. Several improvements of the original ABT have been proposed, e.g., [2], [8], [11].

Mailler and Lesser [4] devise a new algorithm for solving DCSPs, called Asynchronous Partial Overlay (APO). This algorithm uses a mediator to resolve conflicts and hence centralizes small portions of the DCSP. Compared to ABT, APO agents broadcast value assignments to all neighbors, not only to lower priority neighbors. Agents update their agentview similarly to ABT. If an agent finds a conflict with one or more of its neighbors and has not been told by a higher priority agent that it wants to mediate, it assumes the role of the mediator. The mediator conducts a centralized search within the scope of participants of the mediation process. If no satisfying assignments can be found, the mediator announces that the problem is unsatisfiable.

IV. System-driven search

Classical DCSP solvers are designed to find a solution to a problem in a static setting. However, in dynamic distributed environments, agents may join and leave the system and change their spatial position. A solution fulfilling all constraints at one time may become invalid instantly. In such a scenario it is necessary to constantly adapt the system to the current conditions. A solver must rather "chase" a permanently changing goal than compute a solution once. The DCSP algorithms presented above and all solvers known to the authors are not designed to perform in such a way. Furthermore, the loss of a single message could cause these algorithms to fail.

The algorithm proposed in this section represents a best-effort approach to keep a dynamic system close to user-defined constraints in a robust way. As mentioned above, a DCSP consists of a set of agents, $A = \{A_1, \ldots, A_l\}$, a set of variables, $V = \{X_1, X_2, \ldots, X_n\}$, and a set of constraints, $C = \{C_1, C_2, \ldots, C_m\}$. Each variable X_i has a non-empty domain D_i of possible values and is owned by a single agent. The functionality of SDS is presented as pseudo code in Algorithm 1. The basic idea is that agents, upon certain events, simply communicate their variable assignments to the

agents they share constraints with, i.e., the direct neighbors of the constraint graph. This implies the requirement of having a communication channel between any two agents sharing constraints. The sending of variable assignment information is triggered periodically. For faster adaptation that event may be triggered after updating variable assignments, too. Periodical triggering makes the approach less susceptible to message loss and accounts for environment dynamics, such as nodes entering the network. When an agent receives a message it analyzes the sender's status and adapts its variable assignments in order to satisfy all constraints based on knowledge gathered by all messages received so far. Every agent needs to know its variables V and their corresponding domains, its constraints C, and the current values assigned to its variables VA. The $agentview$ container serves as store for information received from other agents and is initially empty. There are three events, on which the algorithm reacts. On start-up, agents assign random values to their variables that are consistent with its intra-agent constraints C_{intra}. The symbol $\stackrel{r}{\leftarrow}$ stands for a random assignment, cw. is an abbreviation for "consistent with". After expiration of a certain time-out, an event is triggered, upon which a message is sent to all agents sharing inter-agent constraints. This message contains the assignments of variables relevant to the respective constraints. Upon receipt of a message the agent updates its $agentview$ with the appended data. If it receives a message from another agent for the first time, a new entry (V, C, VA) is inserted into $agentview$. Next time only the variable assignments are updated. After that, the agent checks whether the current assignments VA are consistent with all values in $agentview$ and its own constraints. Nothing needs to be done if that is the case. If the current assignment is inconsistent, the agent searches for a random consistent assignment. After finding a solution, the assignment is adjusted and the procedure terminates. If $agentview$ does not allow a consistent variable assignment, $agentview$ is cleared except for the latest entry and the agent tries to find a solution which at least conforms to the latest entry. Thus, SDS favors recent information over assignments received sooner. In the worst case, the latest entry of $agentview$ is also deleted and a random assignment is applied which is only consistent with the intra-agent constraints. Note that finding solutions consistent with $agentview$ involves a search comparable to solving classical CSPs. However, $agentview$ only contains local constraints, i.e., constraints concerning the agent itself and the agents it is sharing constraints with. In order to derive a random solution consistent with $agentview$, a random search strategy may be applied.

In order to explain the basic functionality of SDS, it is applied to the simple scenario illustrated in Figure 3. All three examples consist of four agents, $A_1, \ldots, A_4$, and each agent has one variable X_i with the domain $\{0, 1, 2\}$. Agents connected by an edge share the constraint that their variables have to be distinct from each other. Additionally, A_4 has the (intra-agent) constraint that its variable must be greater than one. Initially, the system is consisting only of A_1, A_2, and A_3, which are being in a consistent configuration. When A_4 is joining, the configuration is becoming invalid, as shown in the top part of the figure. After A_4 has sent its configuration, A_2 is looking for a variable assignment that satisfies the constraints shared with A_1 and A_3, as stored in $agentview$. As a result, A_2 sets X_2 to 0 (see middle part of figure). The bottom part assumes that all variables X_i are based on the domain $\{1, 2\}$. In that case, A_2 is not able to find a solution consistent with all entries in $agentview$ and sets X_2 to a value just consistent with X_4, which is one. The next message of A_2 to A_1 and A_3 results in a consistent configuration as shown in the bottom part.

In wireless domains, under the assumption that (1) constraints involve only agents which are within each others transmission radius and (2) that every message sent may be

Algorithm 1 System-Driven Search (SDS)

```
1:  V                                   ▷ Variables with domain information
2:  C = C_intra ∪ C_inter               ▷ Constraints
3:  VA                                  ▷ Variable assignments
4:
5:  agentview ← ∅                       ▷ Information about other agents
6:
7:  procedure ONINIT
8:      VA ←r consistent with (cw.) C_intra
9:  end procedure
10:
11: procedure ONEVENT
12:     send respective subsets of V, C, and VA to direct
    neighbors of constraint graph
13: end procedure
14:
15: procedure ONMESSAGERECEIVE(m)
16:     update agentview with information from m
17:
18:     if VA cw. agentview and C then
19:         return
20:     end if
21:
22:     if consistent assignment exists then
23:         VA ←r cw. agentview and C
24:         return
25:     end if
26:
27:     clear agentview, except latest entry
28:
29:     if consistent assignment exists then
30:         VA ←r cw. agentview and C
31:         return
32:     end if
33:
34:     clear agentview
35:     VA ←r cw. agentview and C
36:
37: end procedure
```

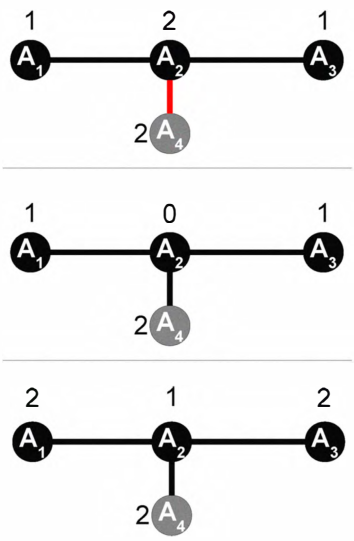


Fig. 3. Simple distributed problem to explain the functionality of SDS. Each agent A_1 to A_4 owns one variable with possible values in $\{0, 1, 2\}$. Constraints forbid connected agents having equal settings for their variables. Agent A_4 has an additional constraint, which states that its variable needs to be greater than one. The top part shows the situation of A_4 joining the network, which results in the violation of the inter-agent constraint between A_2 and A_4. The middle part shows the system configuration after A_4 sends its status to A_2. At the bottom, the resulting configuration can be seen, assuming that the variables' domain is $\{1, 2\}$ instead of $\{0, 1, 2\}$.

received by all devices within the sender's reach, the algorithm can be optimized in order to avoid sending messages. Wireless networks like WSNs typically have strict energy constraints and have to deal with limited battery capacity. Communication amongst nodes consumes the largest part of energy [1]. Under assumptions (1) and (2), it is sufficient for each agent to send a single message containing the variable assignments instead of sending a message to all direct neighbors of the constraint graph, i.e., all agents with shared constraints (see Algorithm 1, Line 12). Under the assumption that the wireless nodes send application messages frequently, it may be sufficient to simply append the variable assignments to these messages and completely avoid active sending of messages. Thus, communication introduced by solving DCSPs is avoided, which results in less energy consumption and improved network lifetime. Furthermore, radio traffic is minimized, which in turn minimizes transmission collisions. In this case, the functionality of the system itself is used as infrastructure to solve DCSPs. This property originally led to the name *System-Driven Search* (SDS). Algorithm 2 shows how the procedure ONEVENT (lines 11-13) can be replaced by ONMESSAGESEND, which appends the current variable assignments to application messages, instead of sending messages actively.

Algorithm 2 SDS with piggybacking

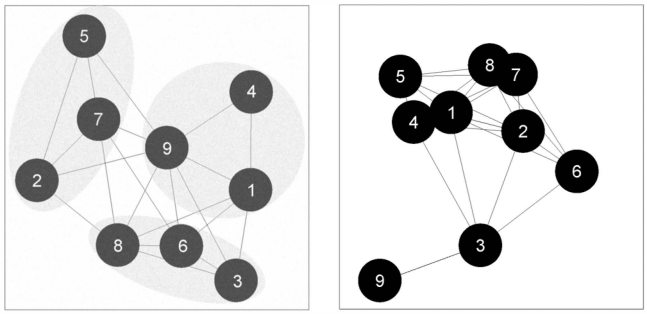
procedure ONMESSAGESEND(m)
append V,C, and VA to m
end procedure

V. EVALUATION

For evaluation purposes, we apply SDS to specific configuration problems typically arising in wireless sensor networks. In this paper we focus on the usage of application messages as a vehicle for information dissemination. The problems considered are partitioning into subgroups of completely interconnected nodes and channel assignment for shared medium access. The former problem, which is NP-hard in its general form, arises for instance if collections of sensor nodes are responsible for the joint execution of certain tasks, like tracking an object [3]. The latter is basically an instance of a graph coloring problem. For both problems, constraints involve only agents which are within each others transmission radius. We assume that every message may be received by every node within the sender's reach and that the wireless nodes send application messages frequently. Therefore, we use the modification of SDS, as presented in Algorithm 2, for our experiments.

For our measurements concerning network partitioning, we restrict the problem to the formation of cliques of size three. More formally this means, given $n = q \cdot 3$ wireless nodes with communication radius R, partition the network into q cliques of size 3. Within a clique each node must be able to communicate with all others. As mentioned in [3], this problem can be formulated as a DCSP as follows: Each node is represented as an agent $i \in A = \{1, \ldots, n\}$ and each agent has a set of two variables $X_{i,1}$ and $X_{i,2}$, which can take on values within $\{1, \ldots, n\}$. The variables represent the other two members of the three-clique. The constraints state that the other members have to be distinct from each other and cannot be the agent itself, i.e., $X_{i,1} \neq X_{i,2}$, $X_{i,1} \neq i$, and $X_{i,2} \neq i$. Further constraints are that agent i considers j_1 and j_2 to be in its clique if and only if j_1 has its variables set to i and j_2 and j_2 to i and j_1. Furthermore, a variable can only point to an agent if it is a direct neighbor. Two values representing clique members are appended to messages in this scenario.

Figure 4(a) shows a solvable instance of the partitioning problem into coordinating cliques of size three, Figure 4(b) illustrates an unsolvable instance. The latter is obviously unsolvable because node 9 (lower left corner) has only one neighbor.



(a) Solvable (b) Unsolvable

Fig. 4. Instances of the problem of partitioning the network into coordinating cliques of size three.

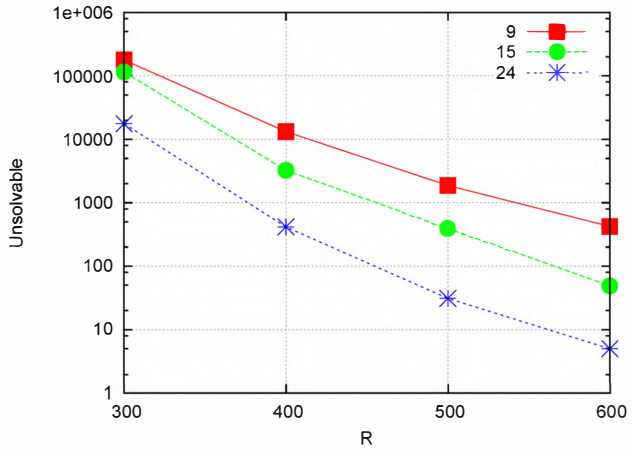


Fig. 5. Unsolvable instances of clique partitioning problems. Each scenario, illustrated by a point in the graph, is tested on 1000 solvable instances. The figure shows the number of unsolvable instances generated during evaluation for different settings. The nodes' transmission radius R is shown on the horizontal axis. The red line represents networks of size 9, the green line of size 15, and the blue line of size 24. The number of unsolvable instances decreases with higher transmission radius and bigger networks.

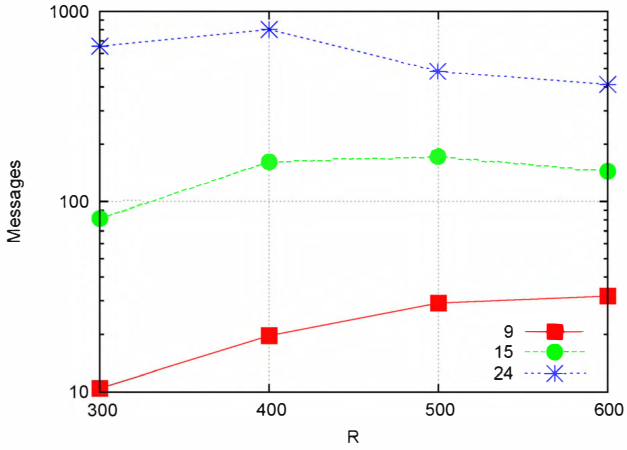


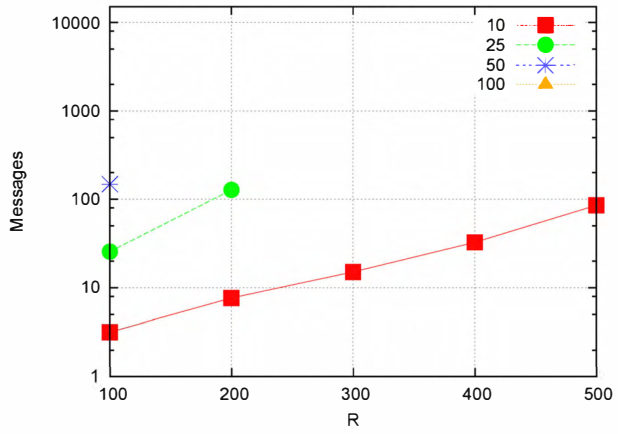
Fig. 6. Number of randomly sent application messages until finding a valid solution to the respective clique partitioning problem. The transmission radius R ranges from 300 to 600 and networks of size 9, 15, and 24 are considered.

For measurements we consider arbitrarily positioned nodes within a plane of size 1000×1000. Hereby we vary the number n of nodes (9, 15, 24) and their transmission radius R (300, 400, 500, 600). Each single scenario is repeated 1000 times using the same setting to obtain representative averaged results. SDS uses application messages for information dissemination. The simulation is conducted in rounds and in each round the probability of a node to send an application message is 10%. Note that SDS does not rely on a round based system nor takes advantage of that fact. Every experiment starts with the random placement of nodes. Then, a centralized algorithm checks whether the generated problem instance is solvable or not. SDS is only applied to solvable problem instances. The simulation environment counts the number of sent application messages until a valid solution is found using SDS. Figure 5 shows the number of unsolvabe problem instances until 1000 solvable problems are generated that can be used as an input for SDS. The number of unsolvable problems decreases with increasing transmission radius R and a higher number of nodes n. Every solvable problem instance is successfully solved by SDS. Figure 6 shows the number of randomly sent application messages until SDS finds a solution to the DCSP.

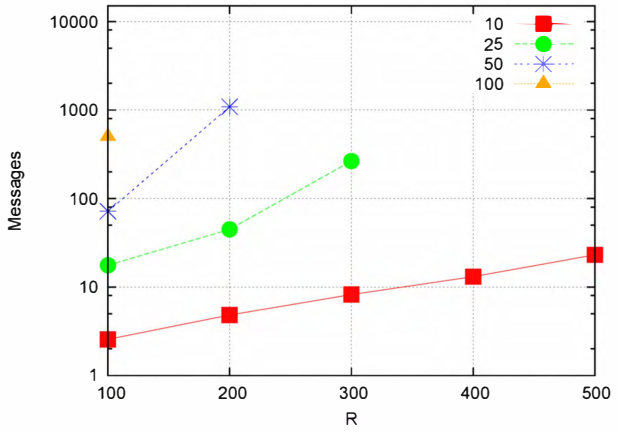
The second problem used to evaluate SDS is channel allocation. A channel may be a time or frequency channel and the goal is to ensure that direct neighbors do not share the same channel. This problem corresponds to a graph coloring problem. The formulation of this problem as CSP is quite straightforward. Each node owns one variable holding the value of its allocated channel. The domain is determined by the available channels. The inter-agent constraints state that the allocated channel must be distinct from the values direct neighbors have allocated. In this case, SDS needs to append only one value to outgoing messages. The experiments for channels allocation are conducted in a similar way as described above, i.e., arbitrarily positioned nodes within a plane of size 1000×1000, varying numbers of nodes n (10, 25, 50, 100), and values for transmission radius R (100, 200, 300, 400, 500). Additionally, the availability of channels varies (3, 5, 10). Each scenario is repeated 1000 times with communication probability of a node again being 10%. As in the former experiments, unsolvable problems are skipped. Unlike clique partitioning, channel assignment problems are more likely to be unsolvable with increased number of nodes and transmission radius. Figure 7(a) (for 3 channels), Figure 7(b) (for 5 channels), and Figure 7(c) (for 10 channels) show the number of messages until a valid solution is found. Missing data points indicate that the experiment is not feasible because corresponding problem instances are very likely unsolvable. Again, SDS is able to find a solution to every solvable problem instance generated by the simulation environment.

The fact that every solvable partitioning and channel allocation is solved by SDS is the main result of the conducted experiments. It is quite astonishing that the simple local working principle of SDS is suitable for finding global solutions in all investigated settings. The piggybacking scheme used for information dissemination with 10% communication probability is comparable to actively sent messages with a message loss probability of 90%.

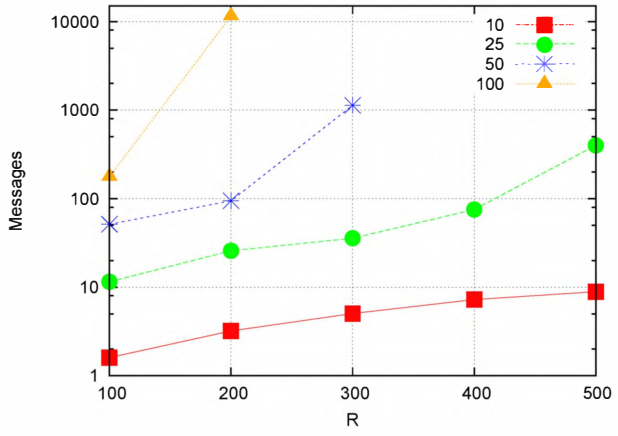
The evaluation results demonstrate that SDS is indeed able to solve interesting DCSPs, but does not really quantify its ability to keep a constantly changing system "close" to specified constraints. While we think the functionality of SDS suggests that is able to cope with such scenarios, concrete measurements will be provided in the future.



(a) 3 Channels



(b) 5 Channels



(c) 10 Channels

Fig. 7. Number of randomly sent application messages for solving channel allocation problems with 3 available channels (a), 5 available channels (b), and 10 available channels (c). The variable R on the horizontal axis stands for the nodes' transmission radius. Colors represent different network sizes of 10, 25, 50, and 100. Missing points indicate, that the corresponding setting is infeasible, i.e., randomly created problem instances are very likely unsolvable. A large radius and many nodes increase the complexity of the problem and thus the number of sent messages.

VI. Conclusion

In this paper, we present a robust algorithm called SDS, for solving DCSPs especially in dynamic environments, in which constant adaptation and control is necessary. Due to its nature, SDS is very robust against message losses, in contrast to traditional DCSP solvers, which may fail if one single message gets lost. For evaluation purposes network partitioning and channel assignment problems are formulated as DCSP. In the investigated settings, SDS is able to solve all solvable problem instances. We plan to apply SDS to further problem instances to evaluate its performance especially in more dynamic scenarios.

Acknowledgement

This work was partially supported by the German Academic Exchange Service (DAAD).

References

[1] S. G. Akojwar and R. M. Patrikar. Improving Life Time of Wireless Sensor Networks Using Neural Network Based Classification Techniques With Cooperative Routing. *International Journal of Communications*, 2(1), 2008.

[2] C. Bessiére, A. Maestre, I. Brito, and P. Meseguer. Asynchronous backtracking without adding links: a new member in the ABT family. *Artif. Intell.*, 161(1-2):7–24, 2005.

[3] B. Krishnamachari, R. Bejar, and S. Wicker. Distributed problem solving and the boundaries of self-configuration in multi-hop wireless networks. *Hawaii International Conference on System Sciences*, 9:297b, 2002.

[4] R. T. Mailler and V. Lesser. Using cooperative mediation to solve distributed constraint satisfaction problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and MultiAgent Systems*, pages 446–453. ACM, 2004.

[5] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2002.

[6] K. Sycara, S. F. Roth, N. Sadeh-Koniecpol, and M. S. Fox. *Distributed Constrained Heuristic Search*, 21(1):1446–1461, December 1991.

[7] R. J. Wallace and E. C. Freuder. Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving. *Artif. Intell.*, 161(1-2):209–227, 2005.

[8] M. Yokoo. Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In *CP '95: Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pages 88–102, London, UK, 1995. Springer-Verlag.

[9] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving. In *ICDCS*, pages 614–621, 1992.

[10] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10:673–685, 1998.

[11] R. Zivan and A. Meisels. Dynamic Ordering for Asynchronous Backtracking on DisCSPs. *Constraints*, 11(2-3):179–197, 2006.