Received: 20 November 2016 | Revised: 23 April 2017 | Accepted: 23 April 2017

DOI: 10.1002/cpe.4179

WILEY

RESEARCH ARTICLE

A colored Petri net model for DisCSP algorithms

Carlos Pascal | Doru Panescu

Automatic Control and Applied Informatics, Gheorghe Asachi Technical University of Iasi, Iasi, Romania

Correspondence
Carlos Pascal, Automatic Control and Applied Informatics, Gheorghe Asachi Technical University of Iasi, Iasi, Romania.
Email: cpascal@ac.tuiasi.ro

Summary

The aim of this paper is to present a colored Petri net that models a system applying a DisCSP method. Our study considered 3 representative DisCSP algorithms: synchronous backtracking, asynchronous backtracking, and weak-commitment search. To obtain the model, it was necessary to transpose the operation of a DisCSP-based system into a discrete event system. The constructed colored Petri net is presented, with details on the information stored in places, procedures associated with transitions, and communication between agents. Through the performed trials (with the n-queens problem), it was proved that the model is correct, easy to use, and adaptable to different operation conditions. The conducted simulations revealed new results about the way the 3 analyzed algorithms compare one with the other. It was showed that the performance of weak-commitment search is degrading in cases close to real conditions, namely, when agents use un-updated information about the state of system. We could also study 3 strategies for improving the performance of DisCSP algorithms, making them more practicable. The proposed model is available for open use, it is independent of the software that carries out a DisCSP method, and it can be enhanced for other DisCSP algorithms and diverse communication schemes.

KEYWORDS

colored Petri nets, constraint satisfaction problems, multiagent systems, software evaluation

1 | INTRODUCTION

The method of distributed constraint satisfaction problem (DisCSP) possesses some advantages that make it relevant for certain applications.[1-9] Besides the benefits obtained by distributed operation, DisCSP is characterized by a clear and easy to express description of the problem. Namely, according to the CSP formalism,[10-13] which is the support for DisCSP, the problem to be solved must refer a set of variables x_1, x_2, ..., x_n that have attached the corresponding domains D_1, D_2, ..., D_n (D_i must be finite). Relations that variables have to satisfy—constraints—can be represented by a finite set of predicates. Such a predicate $P_k(x_{k1}, ..., x_{km})$ is defined on the Cartesian product D_{k1} x ... x D_{km}, and the predicate is true if the assignment of variables x_{k1}, ..., x_{km} satisfies the desired restriction. Thus, the problem is solved when an assignment of variables is found so that all imposed constraints are satisfied. In the case of DisCSP, a distributed search is conducted in a finite network of processes (or nodes). This distributed system is usually developed as a multiagent system (MAS), where variables are handled by agents. The goal of an agent is to communicate and negotiate with the other agents until its variables have values consistent with the constraints of problem.

As one can see in the next section, several methods were developed in order to materialize DisCSP, which use different specifications. In our approach, the following hypotheses have been taken into account:

- Each agent is handling a single variable.
- Agents communicate by sending messages, and they know each other (ie, all agents are neighbors according to the terminology of DisCSP); this means that in our approach, a binary constraint exists between any 2 agents. Thus, the case when additional links must be added between certain agents during the search process is not considered in our research.
- The message communication is an asynchronous one, and messages are received and handled in the order they are sent (no message is lost).
- Any agent knows all constraints that are relevant to its variable.

Considering all these simplifications, we were able to construct a model useful for the analysis of 3 DisCSP algorithms that are between the most used ones (see the next section); these are

asynchronous backtracking (ABT), synchronous backtracking (SBT) and weak-commitment search (WCS). Thus, although some remarks are made regarding other cases, only ABT, SBT, and WCS are thoroughly treated in this paper. The 3 selected algorithms have some common features: All are based on backtracking and use the same mechanism for agents' communication with 2 types of messages: An OK message is sent when an agent changes the value of its variable and this is announced to other agents, and *nogood* messages are used when an agent detects a dead end in its search process (more details about these issues appear in the next sections). Asynchronous backtracking and SBT can be considered subclasses of WCS with respect to the agents' communication and the priorities established for agents; consequently, a model for WCS is described in detail, and then comments are made to clarify the differences for the other 2 algorithms.

DisCSP proves to provide solutions for problems on resource allocation,[14-16] planning/scheduling and timetabling/meeting problems,[17-26] biochemistry,[27] multirobot planning and exploration,[28-32] and distributed configuration.[33,34] In the last years, an application area for DisCSP that had an important development regards various types of networks; one can find results on the use of DisCSP in cognitive radio networks,[35] wireless communication networks,[6,7] and sensor networks.[16,23,36-39]

An important point for application of artificial intelligence techniques in general and especially for DisCSP is the existence of an analyzing mechanism that should be used to check the characteristics and usefulness of a proposed method; a model creates such a possibility, and it can also increase the beneficiaries' confidence in DisCSP. Being known that CSP is NP-complete,[11,40] heuristics are needed, and for such approaches, models allowing performance evaluation can be useful when trying to improve the search process; this is even more important for DisCSP, due to the inherent difficulties of evaluating distributed algorithms.[41] In this context, the purpose of this paper is to provide a model that should enable us to make simulation experiments and analyze different DisCSP algorithms.

In the next sections, after presenting the related work (Section 2), we describe the discrete event system (DES) that operates according to a DisCSP algorithm (Section 3). Then, details of the developed colored Petri net (CPN) model are given in Section 4, which is followed by the results obtained with the CPN model for the n-queens problem (Section 5); the conducted experiments allowed us to analyze the behavior of 3 DisCSP algorithms, to reveal new ideas about their efficiency and to propose some strategies for their operation improvement. Finally, in Section 6, a few conclusions and ideas for future work are presented.

2 | RELATED WORK

2.1 | Classification of DisCSP algorithms

There are more classifications of DisCSP algorithms as one can find in the literature.[7,13,23,42-45] Thus, the algorithms are complete and incomplete, synchronous and asynchronous; in our study, we considered only complete algorithms and both synchronous and asynchronous ones. Some algorithms are based on backtracking and different priorities assigned to agents, but with differences in the use of these mechanisms. In asynchronous forward checking, backtracking is done sequentially,[23] which is not the case for ABT and WCS. While ABT uses fixed priorities for agents, in WCS, these are changed dynamically during the search process.[12,13] In Zivan and Meisels,[45] a classification is done according to how agents perform a single search process (like in SBT, ABT, and WCS) or a multiple one. In the last case, an agent maintains more assignments for its variable and thus more backtracks can appear for different assignments; an example of such an algorithm is the concurrent dynamic backtracking.[45] This case was not considered by the developed model. In all the above mentioned algorithms, agents keep one or more partial solutions and stop when succeed to transform (extend) a partial solution into a complete one, making use of backtracking. This class of methods can suffer from the possibility of a trashing behavior,[46] namely, a repeated search in a subtree of the search space that is of no utility for finding the solution. A way to avoid this drawback, by using Petri nets, is presented in Portinale,[46] for the case of CSP. If we compare Portinale[46] with our research, the following differences have to be mentioned: In Portinale,[46] ordinary Petri nets are used to find a solution to CSP problems, while our approach regards the development of a CPN model useful for simulation and analysis of different DisCSP algorithms. Some methods do not imply backtracking.[11,43] One such approach includes the distributed breakout algorithms,[11,47] in which agents improve their tentative and flawed variables' assignments through communication, until getting the solution. Distributed breakout algorithms are incomplete. Another possibility for not involving backtracking is offered by algorithms that are based on merging more partial solutions. Such an algorithm is the asynchronous partial overlay,[3] and in Grinshpoun and Meisels,[43] this is enhanced so that to be sound and complete. The developed model regards backtracking-based algorithms. Another classification can be done depending on how each agent involved in the DisCSP mechanism is managing one or more variables. Some of the above enumerated algorithms, although being initially designed so that each agent encapsulates a single variable, were enhanced for multivariable problems[19]; for example, ABT and WCS were extended for cases when an agent manages more variables.[8,21] Moreover, one can also find DisCSP methods specifically conceived for the case of multivariable agents, as is the case of distributed backtracking with sessions.[19] In our paper, we considered only the situation of agents with a single variable, and all constraints were binary, which is the common case for the DisCSP algorithms. Finally, all the above approaches regard static problems, namely, the ones that do not change during solution search. Besides this case, there is the field of dynamic DisCSP that addresses changing problems, which is tackled in Mailler and Zheng.[48]

From the broad range of DisCSP algorithms, ABT is considered to be the reference for many researches in DisCSP and we can speak about a family of ABT based techniques.[19,28,49-51] A problem for these algorithms regards the possible explosion of *nogood* messages, which can appear for complex problems. These messages are generated when a backtracking process is started, a *nogood* representing a new constraint discovered during the search process. More enhancements were proposed to diminish this effect. In Brito and Meseguer,[42] ABT

is modified so that to include a kind of synchronism for the backtracking phase, in order to avoid the transmission of redundant messages. In Zivan and Meisels,[45] ABT is adapted so that an agent proposes a dynamic reordering of lower priority agents, and this can conduct to an efficient use of *nogoods*. In Muscalagiu,[51] a classification of *nogoods* is used so that an agent should not take into account those *nogoods* that are not currently useful. In our research, we also tackled the problem of an efficient employment of *nogoods*, by using the minimum *nogoods*, which speeds up the search process; a quite similar idea is stated in Zivan and Meisel,[44] namely, to involve minimum *nogoods*, instead of using the complete state of an agent that discovers an inconsistency. A distinct possibility of improving ABT is presented in Hamadi and Ringwelski,[49] namely, the involvement of more parallel search processes, each using a distinct order between agents.

2.2 | Assessment of DisCSP algorithms

Our paper brings new aspects in the assessment of some DisCSP algorithms. Comparisons between DisCSP algorithms can be found in the literature.[6,13,28,42-45,52,53] These either analyze various distinct algorithms or variants of the same method. In evaluating DisCSP approaches, the computational and communication load must be taken into account. For these 2 performance indicators, the main measures are the total search time and the communication load.[44] In literature, one can find more evaluating parameters. From the computational point of view, there were used: number of cycles,[11,13,28] nonconcurrent constraint checks,[45,54] concurrent constraint checks,[44,49] total CPU time,[28] runtime,[49] parallel runtime,[49] mean time needed to get the solution,[35] and number of backtracks.[49] The number of exchanged messages was used for the communication load.[28,36,44,45,49] In our paper, we worked with the number of cycles and number of messages as main metrics, but the number of backtracks and time needed in simulation were used, too. In addition, we used the number of *nogoods*, which we did not remark in other researches; this parameter can be useful in comparing the number of messages of different types (when knowing the total number of messages, too) and in studying the possibility of grouping messages in agents' communication. It is to notice that when DisCSP is applied to a specific domain, the evaluation process may be conducted with specific metrics, like the total traveled distance of robots for the case of multirobot exploration.[28]

A comparison between the performance of ABT and WCS is made in Bejar et al,[36] the focus being on the communication part of algorithms. Authors study how random delays influence the results of DisCSP algorithms. It is to notice that they arrive at the same conclusion as us, namely, WCS is not always producing better results than ABT; it is observed that for unsatisfiable instances of a problem, WCS is less performant than ABT. Our research shows more than that, namely, even for cases of satisfiable instances, it happens that ABT determines better results than WCS. In Grinshpoun and Meisels,[43] a comparison between ABT, WCS, and asynchronous partial overlay is done. The analysis is made on different types of graph—sparse and dense, with the conclusion that WCS is inefficient for the case of dense graphs. In Wahbi and Brown,[6] the operation of ABT is studied for different network topologies. The conclusion is that the network topology has a great influence for the performance of ABT, mainly with regard to the communication load, but also for the computational load. Moreover, it is highlighted the difference between the constraint graph and communication graph for the practical case of agents within wireless sensor networks. In our paper, we studied only the behavior of DisCSP algorithms for problems that conduct to complete constraint graphs.

In Zivan and Meisels,[44,45] comparisons between ABT, SBT, WCS, and some of their variants are done. One main result of these studies is consistent with the outcome of our paper; namely, the performance of DisCSP algorithms decreases when agents use obsolete information on system state. For example, in Zivan and Meisels,[44] it is shown how this applies for ABT. Different conditions of operation for agents are created by introducing delays in message transmission. When message delays appear, it happens that ABT has a worse performance than SBT, and the worst operation of ABT is obtained when communication functions with random delays. The explanation for this, which is supported by our paper, is as follows. In the case of random delays, agents frequently operate with un-updated information, because data from different neighbors arrive at an agent at distinct times and it proceeds to a decisional step, without being completely informed. This in accordance with our research, which showed that the worst results (not only for ABT, but for WCS, too) are obtained for agents using out of date information. A difference between our approach and those used in Zivan and Meisels[44,45] is that we did not consider message delays, but we could observe the behavior of DisCSP algorithms when tuning the internal agents' operation with respect to how they react to internal and external events. Zivan and Meisels[45] study the performance of ABT and WCS, as well as of a variant of ABT—dynamic ordering ABT. As in Zivan and Meisels,[44] authors are interested in seeing the influence of message delays for DisCSP. In order to better evaluate a real DisCSP system that can have delays in its transmission, an asynchronous message delay simulator is presented. The developed experiments conduct to the conclusion that delays have a strong effect on WCS. Again, these results are consistent with ours, although being obtained through different means.

In Monier et al,[28] a comparison is done between 3 algorithms: ABT, WCS, and distributed backtracking with sessions. The authors are interested in evaluating the 3 algorithms when they have to solve a specific problem, namely, multirobot exploration. If this analysis is related with ours (with respect to ABT being judged against WCS), one can notice a comparable conclusion: WCS can have a worse performance than ABT. While the authors of the literature[28] are focused on relating the behavior of DisCSP algorithms with their application field, we are concerned in our evaluation with discovering the intrinsic merits of each DisCSP variant.

2.3 | Simulation and modeling approaches for DisCSP algorithms

The evaluation of DisCSP algorithms is dependent on the use of appropriate software instruments. In literature, one can find various distributed constraint reasoning simulation tools, which operate according to DisCSP algorithms and allow their analysis and comparison[19,27,55-59]; in these settings, agents have their own execution thread and

communicate via message exchange. In the early stage, Yokoo and collaborators[11,13] evaluate the efficiency of DisCSP algorithms according to a discrete event simulation. This works as follows: All agents react from the initial state, produce the second state, and so on; in each step, all agents read their received messages, update their world model, and react by producing the next state. This approach misses a real system behavior in 2 points: Agents have to react in each step and their reasoning is always done with the actual state. One must avoid assessing a DisCSP algorithm in an ideal operation; namely, many researchers have criticized evaluations done with perfect simulators, which mean an instantaneous message delivery.[44,45] In this criticism, it is emphasized that such simulators, in a wrong way, provide agents of a DisCSP algorithm with the advantage of operating with full information on the current state of the search process. To overcome this, Zivan and Meisels[44,45] introduce random message delays; thus, they could study the impact of agents' reasoning with un-updated information on the current state. The approach of our paper, by using an appropriate model, captures both points: cases when all agents react in the actual state, or some agents cannot react in the current step and they handle one or more messages (but not all) before their reasoning phase. In Bejar et al,[36] a discrete-event simulator is used, which models in a realistic way various communication environments. The basic idea is to test different delay distributions applied to each agent's delivery message queue, when running various DisCSP algorithms. Moreover, authors use in their study a benchmark problem close to reality, namely, the distributed sensors network. Other problems frequently used in assessing DisCSP methods concern: the n-queens problem,[11-13,42] graph coloring,[11,12,43] meeting scheduling,[18,53] sensor networks,[36,37,53] and protein-folding problem.[27] In our study, the n-queens problem was utilized.

Although being flexible instruments, the existing software platforms (eg, DisChoco,[29] which was frequently used to evaluate new DisCSP algorithms and to find the suitable variant for a specific distributed constraint problem) reveal only the output obtained after running a concurrent algorithm, without details about the evolution between the initial and final states. Such information can be provided by a model. For example, as illustrated in the real project at Ericsson Telebit A/S,[60,61] a developed Petri net model of an edge router discovery protocol for mobile ad hoc networks helped to identify several issues in design. Similarly, in tests made by us, the developed CPN model made evident details on the operation of DisCSP algorithms, which would be hard to discover in other way (eg, the ending condition of a DisCSP algorithm for unsatisfiable problems has to be correspondingly adapted depending on how *nogoods* are or not obtained through hyperresolution).

As an additional drawback, the above discussed simulation instruments do not allow the testing of different hypotheses at any level of a DisCSP algorithm and they are dependent on the programming environment (eg, thread-based operation), which would not be the case for a DisCSP model. Our proposal, ie, a CPN model, if compared with the existing simulation instruments and software platforms allowing DisCSP implementation, is between these 2 cases. It allows a more profound evaluation of DisCSP algorithms' operation than a simulation environment, and the model is also close to a prototype so that a user can easily pass from model to real implementation.

In Smith[62] the importance of how a problem is modeled according to the CSP formalism is underlined and an analysis is presented regarding the analytical issues encountered when specifying a problem in the CSP framework. What is lacking in literature is a model independent of implementation aspects and easy to share. An approach that can conduct to such a model is offered by CPNs,[60] which were used in this paper, too. A review on the advantages and usefulness of CPNs for modeling and simulation of concurrent and distributed systems can be found in Jensen and Kristensen.[63] To the best of our knowledge, there is no result about using CPNs for modeling DisCSP algorithms. As expressed in Barták and Salido,[64] establishing of modeling tools for DisCSP is an important as well as difficult and open issue. The goal would be to obtain a model that should be useful independent of the software used to carry out a DisCSP method; the result proposed by our paper has this feature.

The DisCSP algorithms have been implied in manufacturing control architectures,[27,31,32,34,65-68] mainly in the research field; even so, they were rarely applied in real-life problems, the weak adoption of multiagent control systems being discussed in Leitão.[69] About this, the use of a CPN model can facilitate the design and implementation of multiagent and holonic control schemes.[31,32,65-67,70] In a holonic system, the coordination problem is often solved by using the contract net protocol (CNP).[31,65,66,70,71] In such a case, the CPN model allows to test the way negotiation between holons conducts to the needed holarchies. For example, in our research, a CPN model was used for holonic schemes that combined CNP with DisCSP; the model permitted us to test different coordination strategies by involving a DisCSP-based negotiation either between managers or between contractors of the CNP.[66,72] To conclude, CPN models of distributed manufacturing control schemes create an important advantage for the next generation of manufacturing solutions' deployment.

3 | THE DES FOR DisCSP

3.1 | Considered DisCSP formalism

DisCSP can be materialized under the form of an MAS, where each agent manages one or more variables. The goal for such an agent is to find the right values for its variables, consistent with the constraints of problem and the assignments made by other agents. This implies a communication and negotiation between agents until all of them find right values for their variables. Agents use their knowledge bases to keep information on constraints and values chosen by other agents. This information is to be updated according to decisions taken by agents. As already told, we chose in our study three of the most known versions of DisCSP, namely, SBT, ABT, and WCS.[12,13] These algorithms vary on how they operate in a sequential, synchronous manner—SBT, or an asynchronous one—ABT and WCS. Asynchronous backtracking and WCS differ on how agents' priorities are used: In ABT, agents' priorities are fixed, while in WCS, these are dynamically changed during the operation of algorithm. In our study, WCS is considered as the prime algorithm and the differences appearing when SBT or ABT are used are discussed, too.

The common formalism for DisCSP is used (see the literature,[1,4,5,7,12,13] meaning each agent possesses a model for the states of other agents, named *agent view* (abbreviated *aview*), and the constraints generated during the search process are named *nogoods*; this information is stored as explained below. When WCS is used, the *aview* must be divided in 2 subsets: *high agent view* (*hview*) and *low agent view (lview)*. The elements of *aview* for agent a_i (the one that is handling the variable x_i) are 3-tuples:

$$aview_i = \{(x_j, d_j, p_j) \mid j \neq i; d_j \in D_j; p_j \in \mathbb{N}\} \quad (1)$$

where the 3 elements represent the name of variable, its assigned value, and agent's priority. In this way, the name of an agent coincides with the name of variable being handled by it. D_j is the domain for the variable of agent handling the variable x_j, and d_j is the current value assigned to x_j. *hview* is the subset of *aview* defined by

$$hview_i = \{(x_j, d_j, p_j) \mid p_j > p_i\} \quad (2)$$

where p_i is the priority of agent handling variable x_i ($hview_i$ is the part of $aview_i$ containing information on agents that have a higher priority in comparison with the current agent). Similarly, *lview* is defined as

$$lview_i = \{(x_j, d_j, p_j) \mid p_j < p_i\} \quad (3)$$

In the above expressions, when 2 agents have the same priority, the decision about belonging to *hview* or *lview* is taken according to the alphabetical order of their identifiers.

Another part of the model that an agent is keeping during its operation regards the *nogood* constraints. Such a restriction specifies a combination of values for variables of some agents that is discovered by another agent as prohibiting it from assigning a value for its own variable and is sent as a constraint to certain agents. Thus, a *nogood* (abbreviated *ng*) is represented as a list of pairs:

$$nogood = ((x_{j1}, d_{j1}), \ldots, (x_{jn}, d_{jn})) \quad (4)$$

It means that for the combination of values $x_{j1} = d_{j1}, \ldots, x_{jn} = d_{jn}$, the current agent cannot assign a value so that to satisfy the constraints of problem. An agent stores a list of all received *nogoods* (in the following explanations this is called *received_nogoods*, abbreviated *l'rng*) and a list of all transmitted *nogoods* (marked as *sent_nogoods*, abbreviated *l'sng*). Besides this, an agent considers from the *received_nogoods* specific, relevant subsets: only those *nogoods* that are compatible with *aview* or *hview*. A *nogood* is compatible with *aview*/*hview* if variables appearing in *nogood* have the same values in *nogood* and *aview*/*hview*.[13]

As already told, we consider as test problem the case of n-queens. In order to apply DisCSP, each agent represents a queen that is placed in one row. The goal of problem is to find the n queens' positions (the columns where they are placed) so that they do not threaten each other. In Figure 1, the case of the 4-queens problem appears. Each agent chooses a value for its variable and announces the selected value to all agents. In the case illustrated in Figure 1, the agent of variable x_4 has the $aview_4$ as: $hview_4 = \{(x_1, 1, 0), (x_2, 4, 0), (x_3, 2, 0)\}$ and $lview_4 = \{\}$ (here, we refer to the initial stage of solution search, when all agents have the same priority—0). This agent discovers 2 *nogoods* (see Equation 4): $ng_1 = ((x_1, 1), (x_3, 2))$ and $ng_2 = ((x_2, 4), (x_3, 2))$ and sends them

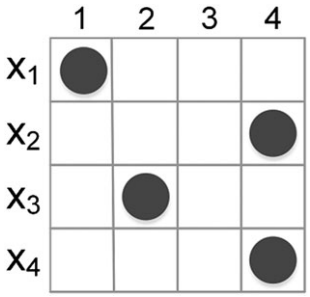

FIGURE 1 Snapshot of 4-queens problem

to the agent 3. Thus, the agent 3 has $hview_3 = \{(x_1, 1, 0), (x_2, 4, 0)\}$, $lview_3 = \{(x_4, 4, 0)\}$, and $l'rng_3 = \{((x_1, 1), (x_3, 2)), ((x_2, 4), (x_3, 2))\}$. The same principle is applied by the other agents.

3.2 | Events to be handled in a DisCSP-based system: WCS algorithm case

Considering the DisCSP mechanism as described in Yokoo and Ishida[12] and Yokoo et al,[13] a first point for obtaining a CPN model of agents is to transpose the algorithm into a DES. By analyzing the operation of WCS, one finds the following types of events:

- *Ok event* appears when the agent handles an *Ok* message received from another agent, which happens when that agent assigned its variable. The effect of this event is an updating of *aview*.
- *Nogood event* appears when the agent handles a *Nogood message* sent by another agent that discovered a *nogood* (the agent receiving the message is referred in the *nogood* constraint). The effect regards the updating of *received_nogoods*.
- *Choose event* appears when 2 conditions are met. First, it happens when either the agent's variable is unassigned (this is the case at the beginning of search) or the current value is not consistent with *hview* and *received_nogoods*.[13] To express the condition of consistency with *hview* and *received_nogoods*, we will briefly call them agent's *knowledge_base*. Second, the event happens only if there is a value in the agent's domain that is consistent with the *knowledge_base*; it means that value does not violate any constraint of the problem imposed for the current agent by agents included in *hview* and does not infringe any *nogood* from the *received_nogoods* being compatible with *hview* and referring only agents of *hview*. The effect of this event is the assignment of a new value that satisfies the above mentioned condition and which minimizes the constraint violations with *lview*, according to the min-conflict heuristics[13]; the chosen value is announced to all agent's neighbors by means of *Ok* messages.
- *Backtrack event* appears when 3 conditions are met. The first one is the same as in the case of *choose*, namely, either the agent's variable is unassigned or the current value is not consistent with the *knowledge_base*. The second condition is the opposite of the case that triggers the *choose* event (it means that the events *backtrack* and *choose* can never have their launching conditions satisfied in the same time). Namely, *backtrack* happens when the agent cannot choose a value for its variable consistent with the *knowledge_base*,

ie, for each value of the agent's domain, there is a constraint (either of the problem or a *nogood*) that is referring the agents in *hview* and that is violated. The third condition is that the agent can generate at least one new *nogood*. There are 2 types of effects for this event:

1. Generating new *nogoods* based on *hview* and *received_nogoods* and their announcement to agents appearing in the produced *nogoods*. For the case when the empty *nogood* is generated, a message announcing that the problem has no solution is sent to all other agents (the empty *nogood* can appear only for a problem with no solution[12,13]).
2. Calculating a new agent's priority so that to be higher than the priorities of the other agents and assigning of a new value for the agent's variable that minimizes the number of constraint violations with *lview* (in fact, here, *lview* ≡ *aview*, because the agent becomes the one with the highest priority); the new assigned value and priority are announced to all neighbors.

Using the above description, with the focus on the underlined items (when and effect), one can materialize the Algorithms 1, 2, and 3.

Algorithm 1. HandleMsgEvent

Condition

1: **when** there is a *message* to be handled

Effect

2: **if** *message* is Ok **then**
3: $aview_i \leftarrow$AddOrModify(x_j, d_j, p_j)
4: **if** *message* is Nogood **then**
5: $received\_nogoods_i \leftarrow$ Add((x_{j1}, d_{j1}), ..., (x_{jm}, d_{jm}))

Algorithm 2. ChooseEvent

Condition

1: **when** there exists a set $D_i^* \subseteq D_i$ of values consistent with $knowledge\_base_i$
2: **and** (x_i = un-assigned **or** $d_i \notin D_i^*$)

Effect

3: Find the set $D_i^* \subseteq D_i$ of values consistent with $knowledge\_base_i$ **and** that minimizes the number of conflicts with $lview_i$ and $received\_nogoods_i$
4: Choose randomly a value $d_i \in D_i^*$
5: Send (Ok, x_i, d_i, p_i) message to all agents

Algorithm 3. BacktrackEvent

Condition

1: **when** there is **no** value in D_i consistent with $knowledge\_base_i$
2: **and** there is a new *nogood*

Effect

3: Generate *new_nogoods* by resolvent-based *nogood* learning
4: Update *sent_nogoods*
5: **if** *empty_nogood* ∈ *new_nogoods* **then**
6: Send *empty_nogood* message to all agents
7: **else**
8: **for each** *nogood* **in** *new_nogoods*
9: Send *nogood* message to all agents in *nogood*
10: Find the set $D_i^\# \subseteq D_i$ of values that minimizes the number of conflicts with $aview_i$ **and** $received\_nogoods_i$
11: Choose randomly a value $d_i \in D_i^\#$
12: $p_i \leftarrow \max(p_j) + 1$; $j \in Id$
13: Send (Ok, x_i, d_i, p_i) message to all agents

Comparing the WCS algorithm described in Yokoo et al[13] with the above set of events, it is to notice that the procedure that checks the consistency of the agent's current value with its *aview* (the procedure named *check_agent_view* in Yokoo et al[13]) is materialized in the proposed DES by means of 2 events: *Choose* and *Backtrack*.

About the procedure of generating new *nogoods*, this can be developed in 2 ways: either getting a *nogood* as a subset of *hview* that does not allow the agent to assign a value for its variable or constructing new *nogoods* by means of the resolvent-based *nogood* learning, also known as hyperresolution.[12,73,74] We choose the last form and materialized it as a specific procedure that generates *nogoods* by considering the agent's domain and the *received_nogoods* set. The details of this procedure are beyond the purpose of this paper. When generating *nogoods*, we used only the minimum *nogoods*; for example, if we get as ng_1 = ((x_1, 1), (x_2, 3)) and ng_2 = ((x_2, 3)), then only the ng_2 is used.

3.3 | Differences for SBT and ABT algorithms

The above description is for WCS, and the differences for SBT and ABT are as follows[12,13] (see also Section 2). As already told, in SBT, agents are ordered and act sequentially. The first agent is choosing a value that is communicated to the next agent. The second agent continues with either a *choose* or *backtrack* event. If it carries on with *choose*, then it announces the assigned value not to all neighbors as in WCS, but only to the next agent according to the established order (line 5 is changed in Algorithm 2). Moreover, the issued *Ok* message includes not only the value for the variable of current agent but also the values of variables for all previous agents. If the agent continues with *backtrack*, then the generated *nogood* is sent only to the previous agent and agent's variable becomes unassigned (lines 8 and 9 are changed, and lines 10-13 are removed in Algorithm 2). Thus, in the case of SBT at each instant, only an agent is acting: It either succeeds to add the partial solution got from the previous agent with a value for its variable, or it cannot do this and generates *nogoods*, asking the previous agent to make a change.

In the case of ABT, agents act concurrently and in an asynchronous manner. A first difference with the above presented operation of WCS is that agents have their priorities a priori established and these do not change during the algorithm run. Then, instead of using *hview*, an agent uses *aview*. When a *choose* event is fired, the issued *Ok* message is sent not to all neighbors, but only to agents having lower priorities in comparison with the current agent (line 5 is changed in Algorithm 3). When a *backtrack* event is fired, each obtained *nogood* is sent to only one agent, namely, that agent appearing in *nogood* and having the lowest priority (line 9 is changed in Algorithm 3). Moreover in the case of a *backtrack* event, the agent does not choose a new value for its variable, but it keeps the previous value (lines 10-13 are removed, meaning that the set $D_i^\#$ is not calculated any longer and the current

value of variable is kept in Algorithm 3). The ending is the same for all algorithms, namely, when the empty *nogood* is generated or when no message is issued (this means that a solution was found).

4 | DisCSP COLORED PETRI NET MODEL

Our goal was to get a model for DisCSP algorithms that should allow the understanding of the behavioral performance of these algorithms and to allow the testing of new assumptions for improving the operation of studied algorithms. Using the above presented elements, such a CPN model was obtained (see Figure 2), by using the CPN Tool.[60] The common notation for Petri nets is used with rectangles representing transitions and ellipses for places; information about the formalism of CPNs can be found in Jensen and Kristensen.[60] The model in Figure 2 contains a number of places that store different information. A place includes a token for each agent, excepting for *dom1* and *dom2* places. For example, in the case of n-queens problem, the number of tokens in each place is *n*, one token for each agent representing a queen. Place *current value* stores the present values of variables handled by agents, while *current priority* keeps the priorities of agents. The default value of a token in the place *current value* is the empty list; as soon as a value is assigned, the list has one element. The default value for a token in the place *current priority* is zero. The place *agent view* keeps the information about values assigned by other agents; data is stored in a form that complies with expression 1. The received and sent *nogoods* are stored in places *received* and *sent nogoods*. Each *nogood* is represented as a list (according to Equation 4) so that a token for *nogoods* contains a list of lists. The messages being received and to be sent are kept in places *In/Out Msgs*. Such a place has the role of a buffer (FIFO), and it is marked with double lines because it makes the connection with the upper level of CPN model, being used to transfer messages between agents. The places *dom1* and *dom2* store temporary information representing domains D_i^* and $D_i^{\#}$, as explained in previous section.

Depending on the content of tokens, different transitions (representing corresponding events) are enabled. The received messages are handled by transition *handle msg*, which updates the information of the places *agent view* or *received nogoods*. About the previously presented event *choose*, this was split in 2 transitions. *Choose_consist_dom* is fired first (when the condition for the appearance of *choose* event is satisfied), and it calculates the subset D_i^* of agent's domain, as explained in previous section. Then, the transition *choose value* is fired, and it randomly selects a value from the set provided by the previous transition. The chosen value is stored in the token of place *current value*. The decided value is also used in a corresponding *Ok* message, which is transferred in place *Out Msgs*. The token in the place *priority* is used by transitions *choose value* and *backtrack* because the agent has to consider its priority with respect to other agents in calculating the effects of these transitions.

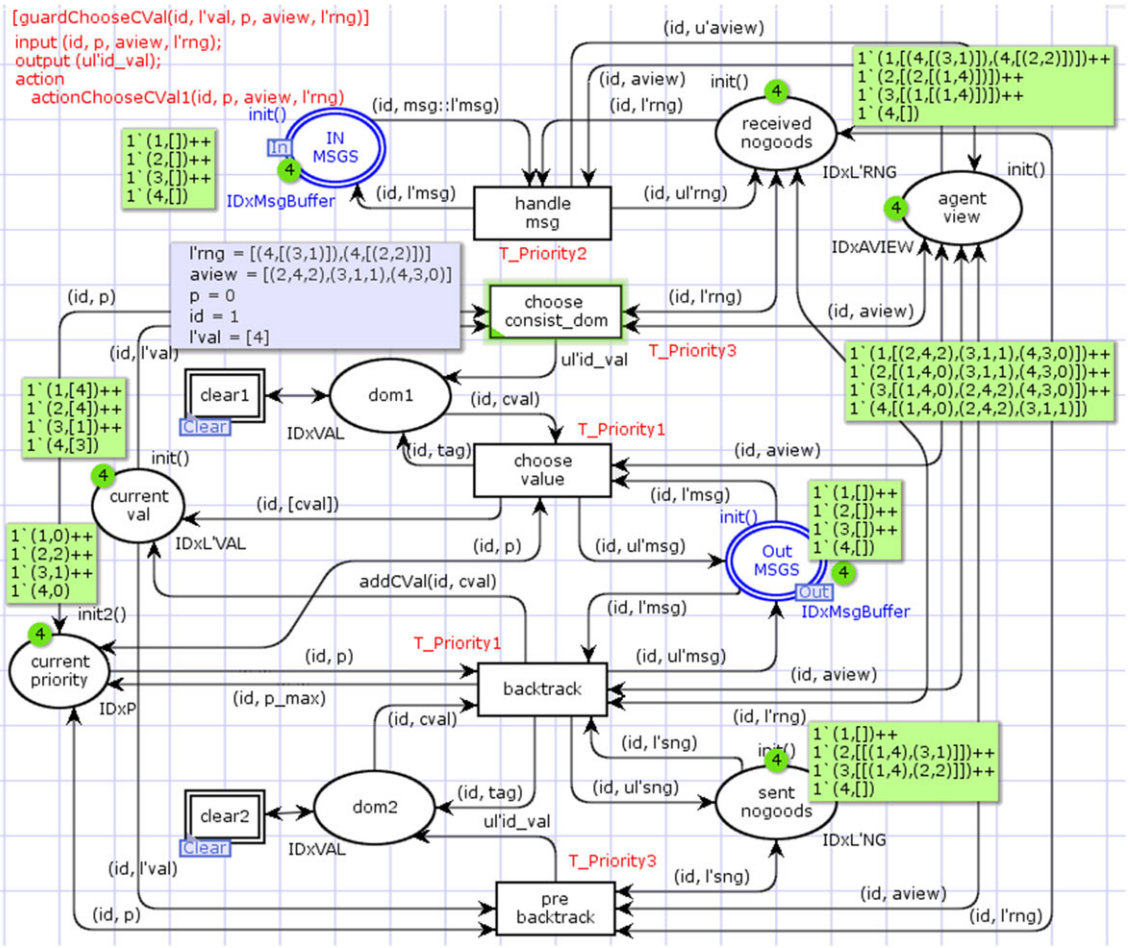

FIGURE 2 Colored Petri net model for agents that apply a DisCSP algorithm

The transition *clear1* is needed for deleting the tokens in place *dom1* after the transition *choose value* is executed. The details about implementing this transition are not represented in Figure 2.

In the same way as for the *choose* event, the phase of backtracking was divided in 2 subevents. First, *pre backtrack* is executed in order to calculate the subset $D_i^{\#}$ from the agent's domain containing values that minimize the constraint violations with *aview* and the received *nogoods*, as detailed in previous section. After that, the transition *backtrack* randomly chooses a value from the determined set and correspondingly updates the token in place *current value* and the token in place *current priority* (the priority is updated according to line 12 in Algorithm 3). Corresponding *Ok* and *Nogood* messages are placed in the output buffer (*Out Msgs*), and the new generated and sent *nogoods* are kept in place *sent nogoods*. The transition *clear2* has the same purpose as *clear1*, but for tokens in place *dom2*.

The separation of *choose* and *backtrack* events in 2 phases had not only an implementation reason, but it was also decided in order to obtain a model efficient for simulation. Thus, the set determined by the first phase can be completely run in simulation in order to highlight all possible cases. Besides this, the nondeterministic behavior is supported by model, as it allows a random selection of values. It is to remark that even 2 phases exist, they carry out an atomic action that is obtained by means of priorities attached to certain transitions. Thus, after the firing of transition *choose consist_dom/pre backtrack*, the following fired transition is always *choose value/backtrack*, even if other transitions are also enabled to fire. In the same way, *clear1* and *clear2* follow after *choose value* and *backtrack*, respectively, according to their priority, in order to remove the unselected values (tokens) that are no longer needed.

The priorities attached to transitions (marked in red in Figure 2) allowed the study of different behaviors with respect to the order between events. Figure 2 illustrates the structural part of CPN model that is the same for all considered DisCSP algorithms, while the behavioral part is hidden in functions attached to transitions; details on these functions are presented in Appendix A. Figure 2 also shows part of the information on an intermediate state for the application of WCS to a case with 4 agents. In the green rectangles, the content of the 4 tokens is presented. One can see that according to the tokens of the place *current value*, the 4 agents assigned to their variables the following values: $x_1 = 4$, $x_2 = 4$, $x_3 = 1$, $x_4 = 3$. By applying WCS, agents have arrived to the priorities displayed in the tokens of place *current priority* (agent of variable x_2 has the maximum priority, followed by the agents of x_3, x_1, and x_4). Each agent knows the other agents' state as shown in place *agent view*. For the state in Figure 2, the transition *choose consist_dom* is the one activated for agent of x_1 due to the fact that its *current value* is not consistent with *hview* (a constraint between x_1 and x_2 is violated, because it cannot be $x_1 = x_2 = 4$) and *received_nogoods* (the agent of x_1 received a *nogood*—see place *received nogoods*—ng = $((x_1, 4), (x_3, 1))$. Moreover, this transition is activated because agent of x_1 can choose a value from its domain that will violate no constraint. Some details of model are hidden in order to make it readable; for example, in the top left part of Figure 2, marked in red, the guard and action functions for transition *choose consist_dom* are presented (such functions exist for each transition but are not displayed in Figure 2). According to Section 3.2, the guard function tests the conditions of *Choose event*, while action function is executed at the firing of transition (the action function is *Choose_consist_dom* of Appendix A). In the gray rectangle nearby transition *choose consist_dom*, the parameters that are checked by the guard function appear; thus, in case of Figure 2, these values are id = 1 being the agent id, $x = 4$ (this appears as *l'val*), $p = 0$, *aview* with information on agents a_2, a_3, a_4, and *l'rng*, which represents the list of *received nogoods* (in this case, there are 2 *nogoods*). These parameters are used both in the guard and action functions. The transition *choose consist_dom* determines as result the set D_i^*, which in the considered case will contain only the value 2. The parameter *ul'id_val* in action function is a list containing the values of D_1^*. The state following the one of Figure 2 will be a final one, with the variables' values: $x_1 = 2$, $x_2 = 4$, $x_3 = 1$, $x_4 = 3$.

When the CPN model is used in simulation, in each step, all the validated transitions are detected. If more transitions are active, then their priorities are taken into account. This helped us to materialize different simulation scenarios (by changing the priorities), which are closer to an ideal or a real operation (see the explanations in the next section). For the case when more transitions of the same priority are simultaneously activated, a random choice is implemented. For example, about the transition *choose value*, it can happen that more values of the agent's variable are consistent with the current constraints; this means that the transition *choose value* is validated more times for each token in position *dom1*. The selection of the transition *choose value* to be fired will be randomly made according to the operation of the CPN tool.

The Petri net in Figure 2 is the model devoted to all agents. The model of entire system is obtained by integrating the models of agents and communication network as shown in Figure 3, according with the capability of CPN Tool.[60] The connection between agents is supported by places *In/Out Msgs* as well as by an appropriate model of the communication network (see Figure 4); this is hidden by the transition *Network* in Figure 3. The transition *Agents* hides the model of Figure 2. The transfer of tokens from buffer *Out Msgs* to *In Msgs* is carried out by a single transition as shown in Figure 4. This transition has the maximum priority because delays produced by the communication network were not considered in our analysis. Our CPN model is accessible for open use at.[75] This can be modified so that new problems to be considered by adapting the function that test constraints.

The developed model can be used with minimal changes for different problems. In Figure 5, two instances of problems are introduced that can be treated with our model, besides the already discussed n-queens

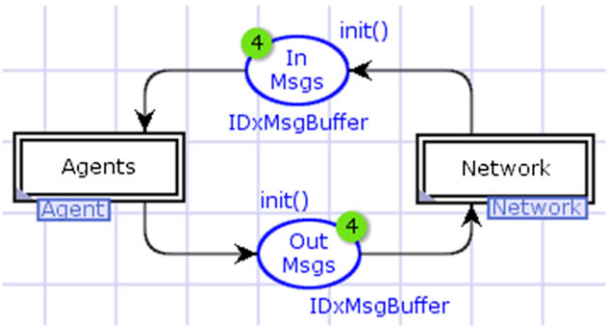

FIGURE 3 Colored Petri net model of a system operating according to DisCSP

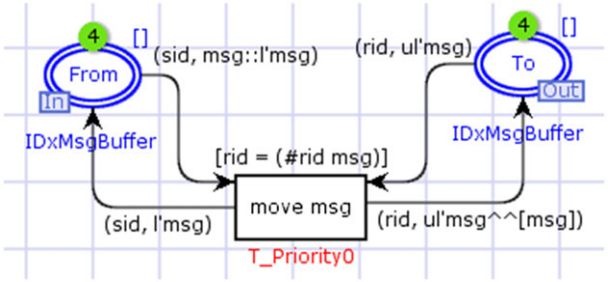

FIGURE 4 Details of the communication network model

problem. The CPN model contains declarations for types, variables, constants, and functions. Thus, in Appendix B, some declarations for 4-queens, graph coloring, and jobshop problems are presented. In the case of 4-queens problem, in the first 9 lines, type declarations appear; one can see that the type for *aview* results as a list of tuples (see line 9) containing the name of variable (this coincides with agent's id and has to be an integer—line 1), the value that can be assigned to it (lines 3 and 4) and agent's priority (line 6); this complies with expression 1. Lines 10 to 12 are for some declarations of variables that are used by the model presented in Figure 2. Line 13 achieves the binary constraint function for the n-queens problems: d_i must be different from d_j because 2 queens must not be placed in the same column and the second restriction avoids the case when 2 queens are in the same diagonal. In line 14, the name of constraint function is assigned to a generic variable that is used by the model. Lines 15 to 19 establish the values for agents' id and domains for their variables.

If one wants to use the model for a graph coloring problem (see the example in Figure 5A), the needed changes are presented in Appendix B. The only changes regard the constraint definition (line 13), number of agents (line 15), and values of variables (lines 16-18).

As a final example, Appendix B presents the case of a jobshop problem (see Figure 5B).[26] In this case, we need to change the type that can be assigned to variables. Namely, for this problem we use 3-tuples of integers representing the resource id, the start time for the task carried out by the resource, and the duration of its activity (see line 3). The binary constraint presented in line 13 regards ordering and overlapping restrictions to be considered between 2 tasks. According to line 15, in our jobshop problem, we have 4 agents and the domains for their variables are presented in lines 16 to 19. The constant defined in line 20 is used by the ordering function to check the ordering constraint. It is to notice that the graph in Figure 5B is not a complete one (no constraint appears between x_1 and x_3). Even so, the model can be used, with the remark that some messages between agents are not used when the constraints are calculated.

A similar adaptation was materialized for a manufacturing problem, as presented in Pascal and Panescu[32]; this regards the values of variables which are plans. Constraints are expressed as predicates with arguments being components of plans. Here, an early version of CPN model was presented. While in Pascal and Panescu,[32] we were focused on managing the coordination of holonic agents by using ABT; in the present research, the CPN model is described with all its details and this is used for evaluating some DisCSP algorithms and finding the means to improve their behavior. For the problem considered in Pascal and Panescu,[32] the reachability graph was obtained which signifies that the proposed manufacturing planning mechanism is sound and complete.

5 | EXPERIMENTS

The CPN model was used to make simulations with chosen DisCSP algorithms (SBT, ABT, and WCS) for the n-queens problem.[12,13] For the number of queens varying from 3 to 13, series of 1000 simulation experiments were carried out. First, the correctness of model was confirmed by obtaining the reachability graph[60] for certain instances of n-queens problem; analyzing the tokens of place *current value* in the final states, the content corresponded to a correct solution, which further endorses that the studied algorithms are sound and complete.

A comparison was made, concerning the considered DisCSP algorithms. This regards the number of messages exchanged between agents and the number of decisional cycles. A cycle represents a decisional phase by which an agent settles the value of its variable and/or establishes new *nogoods* in accordance with the received messages. Both criteria depend on the number of *choose* and *backtrack* events and on the number of new discovered *nogoods* according to the resolvent-based *nogood* learning; these parameters will be considered in the next discussion. The analysis of a DisCSP algorithm can be conducted in 3 cases:

1. Decisions are taken only after reading all incoming messages;
2. after each received message, a decision is established; and
3. treating of received messages is combined with decisions.

The difference between case (1) and the other two is that the last two can determine decisions based on un-updated information regarding the state of system. For example, it can happen that an agent does

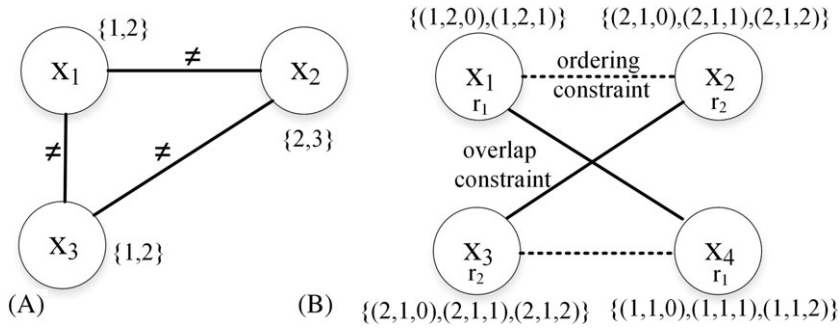

FIGURE 5 Instances of CSP: A, graph coloring and B, jobshop problem

not know that a higher priority agent has already changed its value and thus it produces an obsolete decision (like uselessly changing its value and notifying this to its neighbors). The case (2) is the one when an agent is always forced to take a rush decision; this situation reflects the algorithm presented in Yokoo et al.[13] Although the case (1) is the ideal one, it is important to find those means that should conduct the operation of DisCSP algorithm closer to this behavior. The experiments for the 3 cases could be carried out by establishing an appropriate priority for the transition *handle msg* with respect to the transitions *choose consist_dom* and *pre backtrack* (see Figure 2). The case (3) will produce intermediate results, and it is not further considered in this analysis.

5.1 | Default behavior of DisCSP algorithms

Table A1 from Appendix C presents the results in cases (1) and (2) for the 3 DisCSP algorithms. For each algorithm and instance of problem, the number of exchanged messages, cycles, *backtrack* events, and sent *nogoods* are provided. For SBT, the number of *backtrack* events is equal to the number of sent *nogoods*, while in the case of ABT and WCS, there are more sent *nogoods*, according to the resolvent-based *nogood* learning method that is used. For each parameter, the minimum, maximum, and average values for 1000 simulations are given. About the marked lines (*) in Table A1, first the number of solutions is shown, as obtained from final states. Beginning with the 8 queens, final states did not make evident all possible solutions. Next to this information, the time needed for 1000 simulations (in second) is displayed. Using data of Table A1, Figure 6 is obtained. The average number of messages and of cycles are displayed for the analyzed algorithms. For SBT, there is a single case due to its sequential operation.

About messages, SBT produces the minimum number of messages because each agent outputs a single message after its decision, namely, a message sent to its successor/predecessor, while WCS is producing the greatest number of messages, as the *Ok* messages are sent to all neighbors, and *Nogood* messages are sent to all agents involved in *nogood*.

About the number of cycles for case (1), as expected WCS(1) produces the best results. This can be explained by the fact that an exhaustive search made by an agent is mainly avoided.[13] Comparing ABT(1) with SBT, there are different behaviors for problems with fewer or more agents (more or less than 10 queens). Namely, until the 9-queens problem, ABT has a worse performance (more cycles) because a decision is issued towards more agents, which deal with more *choose* events and may decide wrong values. Thus, one can calculate the number of *choose* events as the difference between the number of cycles and the number of *backtrack* events and notice (according with the data of Table A1 from Appendix C) that until the 9-queens problem it happens that ABT(1) determines more *choose* events in comparison with SBT. Beginning with the problem of 10-queens, the performance of ABT(1) becomes better in comparison with SBT. This can be explained by the fact that the number of *nogoods* produced for each *backtrack* event becomes significant (in average, more than 2 *nogoods* for each *backtrack* event in the case of the 10-queens problem and almost 4 *nogoods*/*backtrack* event for the 13-queens problem, which does not happen for the cases of problems with fewer agents). The important number of received *nogoods* makes the lower priority agents to handle less *choose* events and to avoid mistakes when they carry out such events. Thus, as it can be seen from Table A1, the number of *choose* events for ABT(1) becomes less than the number of *choose* events for SBT, beginning with the 10-queens problem.

The results for case (2) have an important connotation, as they highlight some specific issues of the analyzed algorithms. These data are about the situation when a decision is taken after each received message (*OK* or *Nogood* message). Synchronous backtracking is mainly keeping its performance due to its sequential way of taking decisions. An explosion of the number of messages for WCS(2) and of number of cycles for ABT(2) and WCS(2) appears (see Figure 6). This can be explained as follows. When an agent jumps to taking a decision with un-updated information about the other agents, it is a great chance to produce a wrong (un-useful) result (reaction). More than that, for WCS(2), such an agent propagates the wrong decision to more agents in comparison with the case of ABT(2), which can explain why the results of WCS(2) become worse with respect to ABT(2) (see Figure 6); for example, in the case of 10 queens, the evolution of system can conduct to more than 70 000 messages and 5000 cycles (see Table A1 in Appendix C).

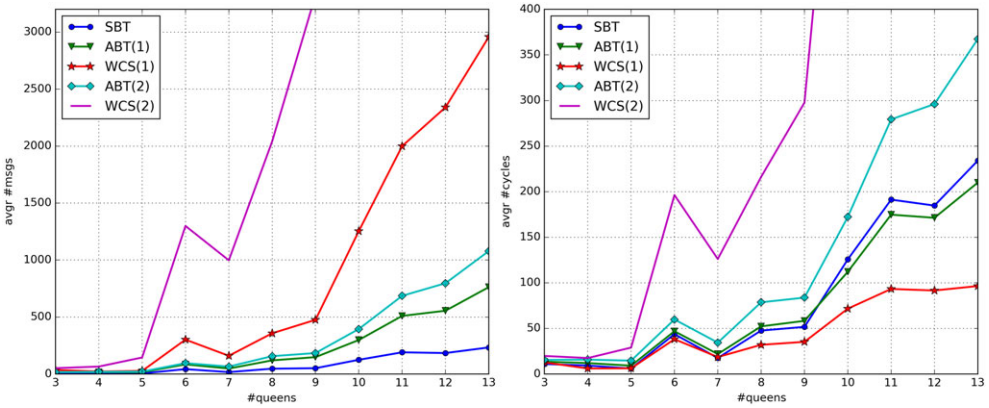

FIGURE 6 Behavior of DisCSP algorithms: average number of messages (left) and of cycles (right). ABT, asynchronous backtracking; SBT, synchronous backtracking; WCS, weak commitment search

5.2 | On applying some strategies for improving DisCSP algorithms

As stated above, WCS determines an increased computational as well as communication load. Thus, we tried to get some mechanisms to diminish these effects. About communication, a possibility to reduce the number of messages is to group them. Consequently, we made a new series of experiments. Message grouping means that when an agent carries out backtracking, it transmits a single message for each neighbor, including both the new chosen value and the set of *nogoods* relevant for the respective neighbor. As an example, for the case of 6-queens problem, it happens that agents arrive in the state of Figure 7A. The agent of x_6 has no choice, and it backtracks. The agent produces the following 3 *nogoods*: $ng_1 = ((x_1, 6), (x_2, 2), (x_5, 4))$; $ng_2 = ((x_2, 2), (x_4, 1), (x_5, 4))$; $ng_3 = ((x_1, 6), (x_3, 5), (x_5, 4))$. The agent of x_6 chooses the value that minimizes the number of conflicts, namely, $(x_6, 4)$, and its priority is settled to be the highest priority. Then, it sends messages with the following content: towards agent of x_1, the chosen value and ng_1 & ng_3; towards x_2, the chosen value and ng_1 & ng_2; towards x_3, the chosen value and ng_3; towards x_4, the chosen value and ng_2; towards agent of x_5, the chosen value and ng_1 & ng_2 & ng_3. By message grouping, agent of x_6 sends only 5 messages instead of 14. In this way, the number of messages for an agent is kept constant, being equal with the number of cycles multiplied with the number of neighbors (in our case, when all agents are neighbors, the number of messages is the number of cycles multiplied with n-1, for n agents). Thus, the system depends only on a single parameter, the number of cycles. In Appendix C, the data for this strategy and the same series of problem instances are presented in Table A2 for case (1) and Table A3 for case (2), being marked with WCSg.

Another possibility to make WCS more efficient is about sending a single *nogood* instead of a whole set. The *nogood* to be sent is chosen as follows. From the set of generated *nogoods*, it is chosen the one that has the minimum number of involved agents. If more generated *nogoods* have the same minimum length (as it happens in the above case of Figure 7A), the second criterion for selecting a single *nogood* is to consider the *nogood* having the maximum priority.[74] For the presented case, the chosen *nogood* will be ng_1. It is normal to use the minimum *nogood* because this is the most powerful with respect to the search process,[13] and we need a further discrimination criterion for cases when there are more *nogoods* of the same length. The obtained results for this strategy are also presented in Appendix C in Tables A2 and A3, being labeled WCSs.

A further possibility to improve a DisCSP algorithm is to make agents use some global information. Namely, an agent can take into account some global constraints that it knows to be applied by other agents (if it knows about them) so that to better decide when to react or not to events. This is the case of agent of x_6, whose *aview* is presented in Figure 7B. Normally, the agent reacts in this state by generating a *nogood*, both when it applies ABT or WCS. If agent of x_6, according to its *aview*, notices that x_1 with x_4 and x_4 with x_5 are in conflict, then it is better for it to wait until these constraints are solved. Otherwise, the agent of x_6 takes a decision using information that will be changed by other agents (x_1, x_4, and x_5 in case of Figure 7B). If agent of x_6 does not wait, its decision is based on data that will be changed in the future so that its reaction can launch a useless search. The influence of a bad decision taken by agent of x_6 is greater for WCS in comparison with ABT, because in WCS, after deciding, agent of x_6 becomes the one with the highest priority and thus its wrong choice can complicate the search. To conclude, a better operation is obtained if agent of x_6 waits until the other agents solve their conflicts. It must be mentioned that with respect to the conflicts between x_1 and x_4, respectively, x_4 and x_5, these will be solved in a certain order, by applying the same principle, namely, agent of x_5 waits until the agents of x_1 and x_4 solve their conflict. These remarks are confirmed by data in Tables A2 and A3 from Appendix C, where the algorithms that apply this strategy are labeled with ABTgc and WCSgc.

Figure 8 shows a comparison between all discussed algorithms for case (1). About the number of messages, one can see that ABTgc determines better results than common ABT. The proposed strategies conduct to an important reduction of the message number for WCS (eg, for 10 queens, the common WCS uses 1250 messages, as average value, while 635 messages were obtained for WCSs, 628 for WCSg, and 535 for WCSgc, see Appendix C). About the number of cycles, again the method of using global constraints determines the best results; its application improves both the performance of ABT and

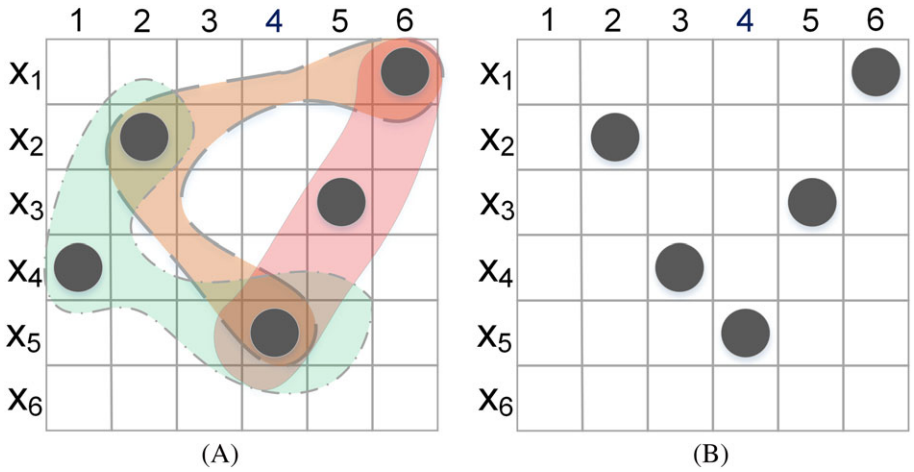

FIGURE 7 Snapshots from the problem of 6 queens

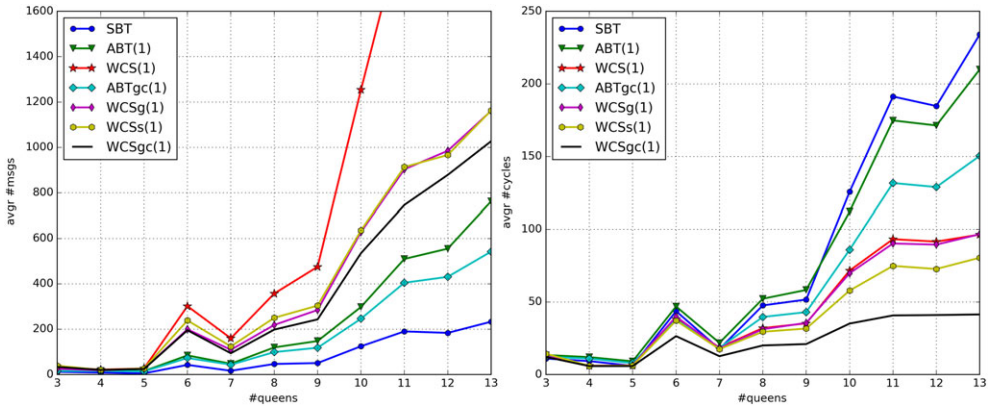

FIGURE 8 Behavior of DisCSP algorithms in case (1) for different strategies. ABT, asynchronous backtracking; SBT, synchronous backtracking; WCS, weak-commitment search

WCS. Weak-commitment searches has a lower number of cycles than common WCS, while WCSg does not improve the performance of WCS in case (1), because WCSg influences only the number of messages.

Figure 9 presents the results for case (2). About the number of messages, WCSg(2) and WCSs(2) do not prevent a message explosion; only WCSgc(2) and ABTgc(2) determine a significant reduction. The effect is similar with respect to the number of cycles, but it is to highlight that ABTgc(2) and WCSgc(2) produce better results than SBT, even in this situation—case (2).

Another idea was to see the results of combinations of strategies. These are displayed in Figure 10. About the number of messages, this is diminished as soon as the global constraints strategy is applied. An important remark is that when this method is used, the difference between cases (1) and (2) is insignificant. When the number of cycles is analyzed, again the use of global constraints mechanism is important; furthermore, when this is combined with message grouping, it conducts the operation of WCS in case (2) close to its operation in case (1). The data obtained in Tables A1, A2, A3 in Appendix C are significant, meaning the same trend is kept for a higher number of queens.

Regarding the simulation time in CPN Tool, Figure 11 shows simulation durations for the above described cases (1) and (2). As expected, SBT conducts to a value of simulation time similar with WCS in case (1), which can be explained by the fact that in each step of applying SBT, a single transition is valid and the computation load is low. One can see that the developed model, when applied to problems until 13 queens, determines a reasonable simulation time. About scalability, our experiments showed an exponential increase of duration with the number of agents. Even so, the utility of CPN model is about understanding the behavior and quality of different DisCSP algorithms. Nevertheless, one must make the distinction between the simulation time (which is obtained with the CPN model) and the real, distributed operation of the MAS.

As shown in Figure 12, the obtained results allowed us to track the computational load of each agent within the system when solving a problem, the instance of 13 queens being chosen. For WCS, the decisional load is homogenously shared between agents, as expected for this algorithm. In the case of SBT, a greater decisional load appears at the middle priority agents, which we suppose that is a characteristic specific for the problem of n-queens. For ABT, an increased decisional load appears for lower priority agents, which can be explained according to the operation of this algorithm; namely, agents with lower priorities are forced to handle more *choose* and *backtrack* events, according to the great number of messages received from agents with higher priority. The computational load flattening that appears for ABT(1) in

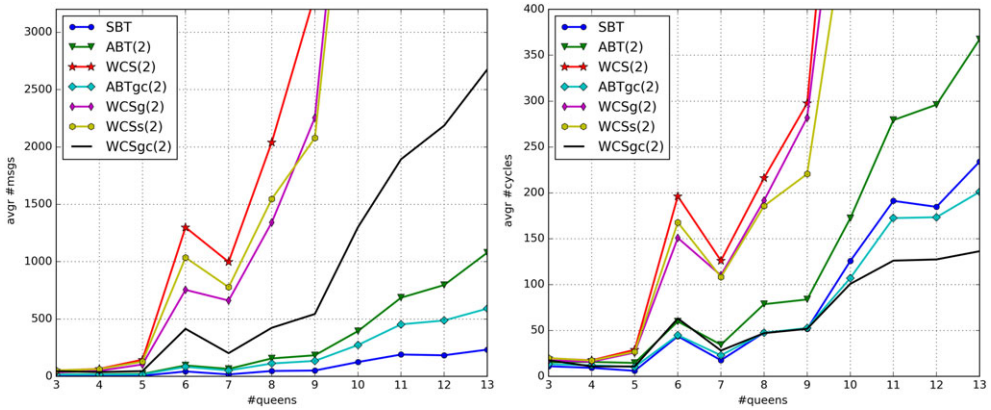

FIGURE 9 Behavior of DisCSP algorithms in case (2) for different strategies. ABT, asynchronous backtracking; SBT, synchronous backtracking; WCS, weak-commitment search

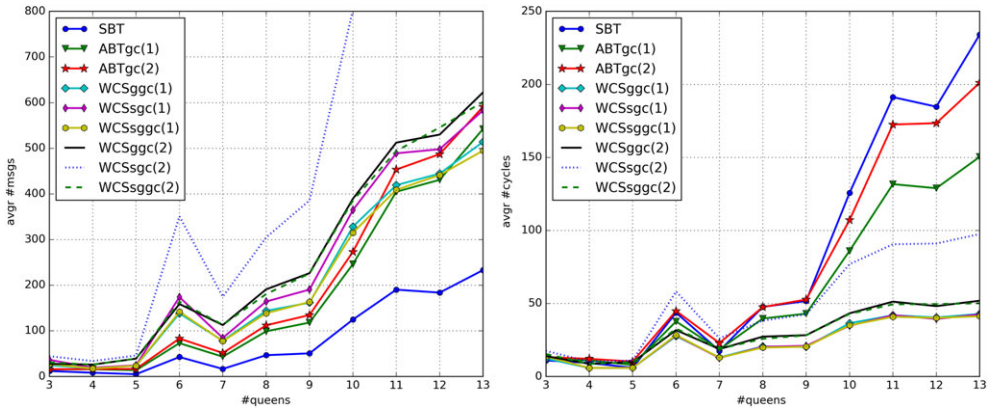

FIGURE 10 Behavior of DisCSP algorithms when combining different strategies. ABT, asynchronous backtracking; SBT, synchronous backtracking; WCS, weak-commitment search

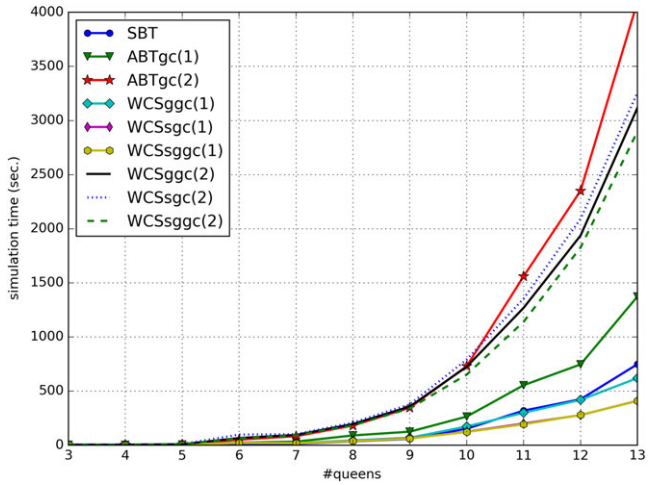

FIGURE 11 Time needed for 1000 simulations for certain strategies (in second). ABT, asynchronous backtracking; SBT, synchronous backtracking; WCS, weak-commitment search

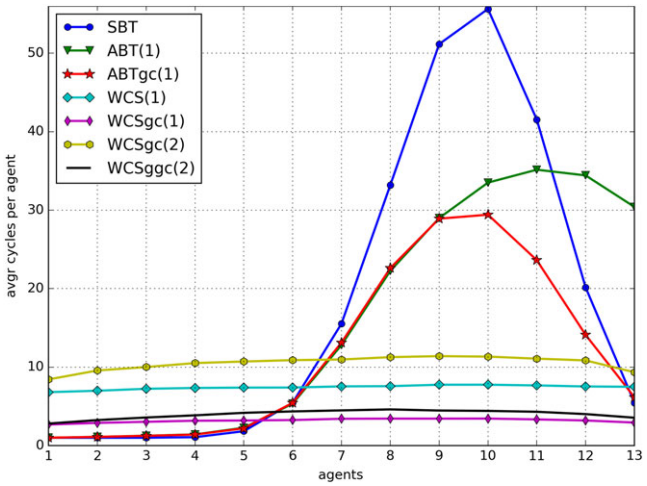

FIGURE 12 Computational load for agents solving the 13-queens problem. ABT, asynchronous backtracking; SBT, synchronous backtracking; WCS, weak-commitment search

comparison with SBT is explained by the exploitation of the greater number of *nogoods* that are used. For SBT and ABT, as shown in Figure 12, the first agents have a small computational load. This comes from the fact that they have few constraints with the other agents and so they get right choices from the beginning. When comparing the cases of ABT(1) and ABTgc(1), the better behavior of the last one is explained by the avoidance of useless reactions of agents with lower priorities, for the states when higher priority agents did not solve their conflicts, yet. Figure 12 confirms the already stated assertion, namely, when adding the common operation of WCS with global constraints and message grouping, its operation in case (2) becomes close to that of case (1). It is to mention that graphs similar to the one in Figure 12 were obtained for other instances of the n-queens problem.

6 | CONCLUSIONS AND FUTURE WORK

Colored Petri net model presented in this paper possesses certain qualities. It was developed for the class of backtracking-based algorithms, including SBT, ABT, and WCS. It can be used for different problems, with diverse number of agents and distinct constraints. Moreover, the CPN model could be used for solving coordination in holonic manufacturing systems.[31,32,65-67,72] The model could be employed in checking different properties (eg, the set of existent solutions, absence of deadlocks, communication, and computation load) as well as in testing diverse manufacturing scenarios for distributed manufacturing systems. Being similar to a prototype, the CPN model is a valuable tool that connects the distributed system design with its implementation, and thus a better deployment of agent-based manufacturing systems can be obtained.

The adaptation of model is simple to be done, by changing the constraint functions between agents. We tested it for 3 variants of DisCSP algorithms, by only modifying the effects of certain transitions. Finally, the developed model allowed an important range of simulation experiments to be carried out by varying the priorities attached to different events (transitions in the CPN). Thus, one can simulate diverse cases with different orders between distinctive types of events: decisional versus communication events.

An important result of our analysis is about the quality of different DisCSP algorithms. The results show that the performance (computational load and number of messages) of a DisCSP algorithm is highly

dependent on the quality (degree of updating) of information used by agents. Namely, there is a high difference between the behavior of the same algorithm if it operates only with updated information and the case when this condition is not met; thus, WCS has a better behavior in comparison with ABT only when agents acquire all information about the system's state first, and then they take decisions. Unfortunately, this is an almost ideal case; without any synchronization between agents, an agent often decides using knowledge on other agents that is not a current one. In this case, what WCS gains by using the min-conflict heuristics does not compensate what it loses by propagating a bad decision based on un-updated information to a large number of agents.

In order to be used, WCS must be endorsed with strategies preventing as much as possible the agents' reactions in unstable states of the search process. We studied 3 strategies consisting in message grouping, sending a single *nogood* and considering global constraints. The experiments showed that WCSg and WCSs are necessary, but not enough with respect to the computational load. From this point of view, WCSgc determines an evident improvement; thus, this is to be used (and it provides better results when combined with message grouping or single *nogood*) whenever it is possible. Namely, WCSgc is applicable for those problems when agents know the constraints of their neighbors. When this approach is not possible, our analysis shows that ABT or SBT are to be preferred.

To conclude, the main contribution of this paper is a first CPN model for DisCSP algorithms, which is easy to understand and use. This model is close to a prototype, and the details provided in Section 3 may be a guide for a fast implementation of tailored DisCSP algorithms. It allowed an analysis that revealed new results about how the behavior of a system that applies DisCSP is dependent on the order of certain events. The proposed model proved to be an important tool for analyzing new strategies that can improve the performance of DisCSP algorithms, too.

As future work, the model can be enhanced to include the case not considered by now, namely, when a connection between certain agents must be established during the algorithm run, as determined by new generated *nogoods*. Then, the model can be enhanced to comprise: other types of DisCSP algorithms, different *nogood* learning techniques, and diverse cases regarding the communication and synchronization between agents. Other strategies, besides those already presented, could be developed in order to tune WCS so that to make its real operation as close as possible to the ideal one.

REFERENCES

1. Bessiere C, Bouyakhf EH, Mechqrane Y, Wahbi M. Agile asynchronous backtracking for distributed constraint satisfaction problems. *23rd IEEE International Conference on Tools with Artificial Intelligence*, 2011; 777-784. https://doi.org/10.1109/ICTAI.2011.122
2. Faltings B, Yokoo M. Introduction: special issue on distributed constraint satisfaction. *Artific Intel*. 2005;161:1-5. https://doi.org/10.1016/j.artint.2004.10.001
3. Mailler R, Lesser V. Asynchronous partial overlay: a new algorithm for solving distributed constraint satisfaction problems. *J Artific Intel Res*. 2006;25:529-576. https://doi.org/10.1613/jair.1786
4. Meisels A. *Distributed Search by Constrained Agents: Algorithms, Performance, Communication*. Springer Science & Business Media; 2008.
5. Shoham Y, Leyton-Brown K. *Multiagent Systems: Algorithmic, Game-theoretic, and Logical Foundations*. Cambridge University Press; 2008.
6. Wahbi M, Brown K. The Impact of Wireless Communication on Distributed Constraint Satisfaction. In: O'Sullivan B ed. *Principles and Practice of Constraint Programming, Lecture Notes in Computer Science*, Springer. 2014;8656:738-754. https://doi.org/10.1007/978-3-319-10428-7
7. Yeoh W, Yokoo M. Distributed problem solving. *AI Magazine*. 2012;33(3):53-65. https://doi.org/10.1609/aimag.v33i3.2429
8. Yokoo M. Distributed constraint satisfaction: foundations of cooperation in multi-agent systems. *Springer Sci Bus Med*. 2012. https://doi.org/10.1007/978-3-642-59546-2
9. Zanker M, Jannach D. Modeling and solving distributed configuration problems: a CSP-based approach. *IEEE Transact Knowl Data Eng*. 2013;25(3):603-618. https://doi.org/10.1109/TKDE.2011.236
10. Freuder E, Mackworth A. Constraint Satisfaction: An Emerging Paradigm. In: Rossi F, Van Beek P, Walsh T, eds. *Handbook of Constraint Programming*. New York: Elsevier; 2006:11-25.
11. Yokoo M, Hirayama K. Algorithms for distributed constraint satisfaction: a review. *Autonom Agents Multi-Agent Syst*. 2000;3:85-207. https://doi.org/10.1023/A:1010078712316
12. Yokoo M, Ishida T. Search algorithms for agents. In: Weiss G, ed. *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*. Cambridge: The MIT Press; 2001:165-199.
13. Yokoo M, Durfee E, Ishida T, Kuwabara K. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transact Knowl Data Eng*. 1998;10(5):673-685. https://doi.org/10.1109/69.729707
14. Conry SE, Kuwabara K, Lesser VR, Meyer RA. Multistage negotiation for distributed constraint satisfaction. *IEEE Transact Syst Man Cybernet*. 1991;21(6):1462-1477. https://doi.org/10.1109/21.135689
15. Modi PJ, Jung H, Tambe M, Shen W, Kulkarni S. A dynamic distributed constraint satisfaction approach to resource allocation. *Proc. of International Conference on Principles and Practices of Constraint Programming (CP-2001)*. 2001;685-700. https://doi.org/10.1007/3-540-45578-7_56
16. Petcu A, Faltings B. A value ordering heuristic for distributed resource allocation. *Proc. of Joint Annual Workshop of ERCIM/CoLogNet on CSCLP'04*. 2004;86-97. https://doi.org/10.1007/11402763_7
17. Bartak R, Salido M, Rossi F. Constraint satisfaction techniques in planning and scheduling. *J Intell Manuf*. 2010;21(1):5-15. https://doi.org/10.1007/s10845-008-0203-4
18. Hassine AB, Ho TB, Ito T. Meeting scheduling solver enhancement with local consistency reinforcement. *Appl Intel*. 2006;24(2):143-154. https://doi.org/10.1007/s10489-006-6935-y
19. Mandiau R, Vion J, Piechowiak S, Monier P. Multi-variable distributed backtracking with sessions. *Appl Intel*. 2014;41(3):736-758. https://doi.org/10.1007/s10489-014-0532-2
20. Meisels A, Kaplansky E. Scheduling agents–distributed timetabling problems, *Practice and Theory of Automated Timetabling IV*, Springer 2003; 2740:166-177. https://doi.org/10.1007/978-3-540-45157-0_11
21. Meisels A, Lavee O. Using additional information in DisCSP search. *Proc. of 5th Workshop on Distributed Constraints Reasoning*. 2004;4.
22. Nissim R, Brafman R, Domshlak C. A general, fully distributed multi-agent planning algorithm. *Proc. of AAMAS 2010*. 2010;1:1323-1330.
23. Wahbi M, Ezzahir R, Bessiere C, Bouyakhf EH. Nogood-based asynchronous forward checking algorithms. *Constr*. 2013;18(3):404-433. https://doi.org/10.1007/s10601-013-9144-4
24. Wallace R, Freuder E. Constraint-based multi-agent meeting scheduling: effects of agent heterogeneity on performance and privacy loss, *Proc. of DCR'02*. 2002;176-182.
25. Xiang Y, Zhang W. Distributed university timetabling with multiply sectioned constraint networks. FLAIRS Conference. 2008;567-571.

26. Pascal C, Panescu D. On applying DisCSP for scheduling in holonic systems. *20th International Conference on System Theory, Control and Computing (ICSTCC)*. 2016;423-428. https://doi.org/10.1109/ICSTCC.2016.7790702
27. Muscalagiu I, Popa HE, Panoiu M, Negru V. Multi-agent Systems Applied in the Modelling and Simulation of the Protein Folding Problem Using Distributed Constraints. In: Klusch M, Thimm M, Paprzycki M eds. *Multiagent System Technologies, Proceedings of MATES 2013, Lecture Notes in Computer Science*, Springer. 2013;8076:346-360. https://doi.org/10.1007/978-3-642-40776-5_29
28. Monier P, Doniec A, Piechowiak S, Mandiau R. Metrics for the evaluation of DisCSP: some experiments on multi-robot exploration. *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*. 2010;2:370-373. https://doi.org/10.1109/WI-IAT.2010.71
29. Monier P, Doniec A, Piechowiak S, Mandiau R. Comparison of DCSP algorithms: a case study for multi-agent exploration. In: Demazeau Y, Pechoucek M, Corchado JM, Perez JB eds. Proc. of the 9th international conference on practical applications of agents and multiagent systems (PAAMS), Springer, *Advances in Intelligent and Soft Computing*. 2011; 88:231-236. https://doi.org/10.1007/978-3-642-19875-5_30
30. Pal A, Ritu RT, Shukla A. Communication constraints multi-agent territory exploration task. *Appl Intel*. 2013;38(3):357-383. https://doi.org/10.1007/s10489-012-0376-6
31. Panescu D, Pascal C. A constraint satisfaction approach for planning of multi-robot systems, *Proc. of 18th International Conference on System Theory, Control, and Computing (ICSTCC)*, Sinaia, Romania; 2014: 157-162. https://doi.org/10.1109/ICSTCC.2014.6982408
32. Pascal C, Panescu D. A Petri net model for constraint satisfaction application in holonic systems. IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), Cluj-Napoca, Romania. 2014. https://doi.org/10.1109/AQTR.2014.6857900
33. Felfernig A, Friedrich G, Jannach D, Zanker M. Towards distributed configuration. Advances in Artificial Intelligence. In: Baader F, Brewka G, Eiter T, eds. *Lecture Notes in Computer Science*. Vol. 2174. Berlin: Springer; 2001:198-212. https://doi.org/10.1007/3-540-45422-5_15
34. Jannach D, Zanker M. Modeling and solving distributed configuration problems: a CSP-based approach. *IEEE Transact Knowl Data Eng*. 2013;25(3):603-618. https://doi.org/10.1109/TKDE.2011.236
35. Sodagari S, 2014. Application of asynchronous weak commitment search in autonomous quality of service provision in cognitive radio networks, arXiv preprint, arXiv:1403.2077
36. Bejar R, Domshlak C, Fernández C, et al. Sensor networks and distributed CSP: communication, computation and complexity. *Artific Intel*. 2005;161 (1-2):117-147. https://doi.org/10.1016/j.artint.2004.09.002
37. Fernandez C, Bejar R, Krishnamachari B, Gomes C. Communication and computation in distributed CSP algorithms. *Principles and Practice of Constraint Programming*, Springer, Berlin 2002;664-679. https://doi.org/10.1007/3-540-46135-3_44
38. Fouchal H, Habbas Z. Distributed backtracking algorithm based on tree decomposition over wireless sensor networks. *Concurr Comput: Pract Exp*. 2013;25(5):728-742. https://doi.org/10.1002/cpe.1804
39. Zhang W, Xing Z, Wang G, Wittenburg L. An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks. Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, New York. 2003;185-192.
40. Bessiere C. Constraint Propagation. In: Rossi F, Van Beek P, Walsh T, eds. *Handbook of Constraint Programming*. New York: Elsevier; 2006:27-82.
41. Brito I, Herrero F, Meseguer P. On the evaluation of DisCSP algorithms. *Fifth International Workshop on Distributed Constraint Reasoning at the Tenth International Conference on Principles and Practice of Constraint Programming (CP-2004)*, Toronto, Canada, 2004.
42. Brito I, Meseguer P. Synchronous, asynchronous and hybrid algorithms for DisCSP. Principles and Practice of Constraint Programming (CP-2004). In: Wallace M, ed. *Lecture Note in Computer Science*. Vol. 3258. Springer; 2004:791-805.
43. Grinshpoun T, Meisels A. Completeness and performance of the APO algorithm. *J Artific Intel Res*. 2008;33:223-258. https://doi.org/10.1613/jair.2611
44. Zivan R, Meisels A. Synchronous vs asynchronous search on DisCSPs. *Proceedings of the First European Workshop on Multi-Agent Systems (EUMA)* 2003.
45. Zivan R, Meisels A. Message delay and DisCSP search algorithms. *Ann Math Artific Intel*. 2006;46(4):415-439. https://doi.org/10.1007/s10472-006-9033-2
46. Portinale L. Modeling and solving constraint satisfaction problems through Petri nets. *International Conference on Application and Theory of Petri Nets*. 1997;1248:348-366. https://doi.org/10.1007/3-540-63139-9_45
47. Hirayama K, Yokoo M. The distributed breakout algorithms. *Artific Intel*. 2005;161:89-115. https://doi.org/10.1016/j.artint.2004.08.004
48. Mailler R, Zheng H. A new analysis method for dynamic, distributed constraint satisfaction. In: Lomuscio A, Scerri P, Bazzan A, Huhns M eds. *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AMAS 2014)*; 2014.
49. Hamadi Y, Ringwelski G. Boosting distributed constraint satisfaction. *J Heurist*. 2011;17(3):251-279. https://doi.org/10.1007/11564751_41
50. Meisels A, Zivan R. Asynchronous forward-checking for DisCSPs. *Constr*. 2007;12(1):131-150. https://doi.org/10.1007/s10601-006-9013-5
51. Muscalagiu I. The effect of flag introduction on the explosion of nogood values in the case of ABT family techniques. In: *Multi-Agent Systems and Applications IV*. Berlin: Springer; 2005:286-295. https://doi.org/10.1007/11559221_29
52. Wahbi M, Brown K. Global Constraints in Distributed CSP: Concurrent GAC and Explanations in ABT. In: *Principles and Practice of Constraint Programming*. Springer International Publishing; 2014:721-737. https://doi.org/10.1007/978-3-319-10428-7_52
53. Wahbi M, Ezzahir R, Bessiere C, Bouyakhf EH. Nogood-based asynchronous forward checking algorithms. *Constr*. 2013;18(3):404-433. https://doi.org/10.1007/s10601-013-9144-4
54. Meisels A, Kaplansky E, Razgon I, Zivan R. Comparing performance of distributed constraints processing algorithms. *Workshop on Distributed Constraint Reasoning (AAMAS-2002)*. 2002.
55. Grubshtein A, Herschorn N, Netzer A, Rapaport G, Yaffe G, Meisels A. The distributed constraints (DisCo) simulation tool. Proceedings of the IJCAI workshop on Distributed Constraint Reasoning. 2011;30-42.
56. Leaute T, Ottens B, Szymanek R. FRODO 2.0: An open-source framework for distributed constraint optimization, *Proceedings of the IJCAI 2009 Workshop on Distributed Constraint Reasoning*, 2009;160-164.
57. Mailler R, Graves J. Solving distributed CSPs using dynamic, partial centralization without explicit constraint passing. In: *Principles and Practice of Multi-Agent Systems*. Berlin, Heidelberg: Springer; 2012:27-41.
58. Sultanik EA, Lass RN, Regli WC. Dcopolis: a framework for simulating and deploying distributed constraint reasoning algorithms. *Proc AAMAS*. 2008;2008:1667-1668.
59. Wahbi M, Ezzahir R, Bessiere C, Bouyakhf EH. DisChoco 2: a platform for distributed constraint reasoning. *Proceedings of the IJCAI workshop on Distributed Constraint Reasoning* 2011;112-121.
60. Jensen K, Kristensen LM. *Coloured Petri nets: Modeling and Validation of Concurrent Systems*. New York: Springer-Verlag; 2009.
61. Kristensen LM, Jensen K. Specification and validation of an edge router discovery protocol for mobile ad-hoc networks, *Proceedings of Integration of Software Specification Techniques for Applications in Engineering*, Lecture Notes in Computer Science, Springer, Berlin 2004;3147:248-269.
62. Smith B. Modelling. In: Rossi F, Van Beek P, Walsh T, eds. *Handbook of Constraint Programming*. Elsevier: New York; 2006:375-404.

63. Jensen K, Kristensen LM. Colored Petri nets: a graphical language for formal modeling and validation of concurrent systems. *Comm ACM*. 2015;58(6):61-70. https://doi.org/10.1145/2663340
64. Barták R, Salido M. Constraint satisfaction for planning and scheduling problems. *Constr*. 2011;16(3):223-227. https://doi.org/10.1007/s10601-011-9109-4
65. Panescu D, Pascal C. An extended contract net protocol with direct negotiation of managers. In: *Service Orientation in Holonic and Multi-Agent Manufacturing and Robotics*. Springer International Publishing; 2014:81-95.
66. Panescu D, Pascal C. Holonic coordination obtained by joining the contract net protocol with constraint satisfaction. *Comp Ind*. 2016;81:36-46. https://doi.org/10.1016/j.compind.2015.08.010
67. Pascal C, Panescu D. A Petri net model for constraint satisfaction application in holonic systems. IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), Cluj-Napoca, Romania; 2014. https://doi.org/10.1109/AQTR.2014.6857900
68. Wahbi M. Algorithms and ordering heuristics for distributed constraint satisfaction problems. Doctoral dissertation, Université Mohammed V-Agdal; 2012.
69. Leitão P. Agent-based distributed manufacturing control: a state-of-the-art survey. *Eng Appl Artif Intel*. 2009;22(7):979-991.
70. Hsieh FS. Developing cooperation mechanism for multi-agent systems with Petri nets. *Eng Appl Artif Intel*. 2009;22(4):616-627.
71. Borangiu T, Gilbert P, Ivanescu NA, Rosu A. An implementing framework for holonic manufacturing control with multiple robot-vision stations. *Eng Appl Artif Intel*. 2009;22(4):505-521.
72. Pascal C, Panescu D. On applying DisCSP for scheduling in holonic systems, *Proc. of 20th International Conference on System Theory, Control, and Computing (ICSTCC)*, Sinaia, Romania. 2016;423-428.
73. Hirayama K, Yokoo M. The effect of nogood learning in distributed constraint satisfaction. *Proceedings of the 20th International Conference on Distributed Computing Systems*. 2000;169-177. https://doi.org/10.1109/ICDCS.2000.840919
74. Lee J, Shi Y. Removing redundant conflict value assignments in resolvent based nogood learning. *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems, International Foundation for Autonomous Agents and Multiagent Systems*. 2014;1133-1140.
75. DisCSP Models. Available: http://github.com/carlospascal/DisCSP-CPN-Models, 2016.

How to cite this article: Pascal C, Panescu D. A colored Petri net model for DisCSP algorithms. *Concurrency Computat: Pract Exper*. 2017;29:e4179. https://doi.org/10.1002/cpe.4179

APPENDIX A

FUNCTIONS FOR TRANSITIONS

function Choose_consist_dom(x_i, p_i, $aview_i$, $received\_nogoods_i$) **returns** D_i^*

1: $D_i \leftarrow$ Get_domain(x_i)
2: $hview \leftarrow$ Get_hview(x_i, p_i, $aview_i$)
3: $dom\_consist\_hview \leftarrow$ Get_ dom_consist_hview(x_i, D_i, $hview$)
4: $dom\_consist\_hview\_and\_Ng \leftarrow$ Get_dom_consist_hview_and_Ng(x_i, $dom\_consist\_hview$, $hview$, $received\_nogoods_i$)
5: $lview \leftarrow$ Get_lview(x_i, p_i, $aview_i$)
6: $D_i^* \leftarrow$ Get_MinConflict(x_i, $hview$, $lview$, $dom\_consist\_hview\_and\_Ng$, $received\_nogoods_i$)
7: **return** D_i^*

// For SBT and ABT, $D_i^* \leftarrow dom\_consist\_hview\_and\_Ng$

function Choose_val(x_i, d_i^*, p_i, $aview_i$) **returns** msgs

1: $receivers\_ids \leftarrow$ Get_SentOk(x_i)
2: $msgs \leftarrow$ Compose_msgs(x_i, $receivers\_ids$, Ok, (x_i, d_i^*, p_i))
3: **return** $msgs$

//* For ABT, Get_SentOk returns the ids of agents having lower priorities; for SBT it returns the id of the agent having the next lower priority

//** For SBT, Compose_msgs is adapted to include $aview_i$ in the Ok message

function Prebacktrack(x_i, d_i, $aview_i$, $received\_nogoods_i$) **returns** $D_i^\#$

1: $D_i \leftarrow$ Get_domain(x_i)
2: $D_i^\# \leftarrow$ Get_MinConflict(x_i, $aview_i$, $aview_i$, D_i, $received\_nogoods_i$)
3: **return** $D_i^\#$

// for ABT and SBT $D_i^\# \leftarrow \{d_i\}$

function Backtrack(x_i, $d_i^\#$, p_i, $aview_i$, $received\_nogoods_i$, $sent\_nogoods_i$) **returns** p_max, $msgs$, $sent\_nogoods$

1: $D_i \leftarrow$ Get_domain(x_i)
2: $hview \leftarrow$ Get_Hview(x_i, p_i, $aview_i$)
3: $nogoods \leftarrow$ Generate_Nogoods(x_i, D_i, $hview$, $received\_nogoods_i$)
4: $new\_nogoods \leftarrow$ Get_New_Nogoods($nogoods$, $sent\_nogoods_i$)
5: $sent\_nogoods_i \leftarrow new\_nogoods$ **U** $sent\_nogoods_i$
6: **if** $new\_nogoods$ contains the *empty* nogood **then**
7: $msgs\_Nosolution \leftarrow$ Compose_msgs_Nosolution(x_i)
8: **return** (p, $msgs\_Nosolution$, $sent\_nogoods_i$)
9: $msgs\_nogoods \leftarrow$ Compose_msgs_nogoods(x_i, $new\_nogoods$)
10: $p\_max \leftarrow$ 1+ Get_max_priority($aview_i$)
11: $receivers\_ids \leftarrow$ Get_SentOk(x_i)
12: $ok\_msgs \leftarrow$ Compose_msgs(x_i, $receivers\_ids$, Ok, (x_i, $d_i^\#$, p_max))
13: **return** (p_max, $msgs\_nogoods$ **U** ok_msgs, $sent\_nogoods_i$)

// lines 10-12 are removed when SBT and ABT are used

// line 9 is adapted for SBT and ABT, namely the *nogood* should be sent to the lowest priority agent from *nogood*

APPENDIX B

MODEL DECLARATIONS FOR 3 PROBLEMS

Model declarations for 4-queens problem

```
                (* some declarations of types *)
1:    colset ID = INT;
2:    colset L'ID = list ID;
3:    colset VAL_Queen = INT;
4:    colset VAL = VAL_Queen;
5:    colset L'VAL = list VAL;
6:    colset P = INT;
7:    colset IDxVALxP = product ID*VAL*P;
8:    colset L'IDxVALxP = list IDxVALxP;
9:    colset AVIEW = L'IDxVALxP;
                (* some declarations of variables *)
10:   var id:ID;
11:   var cval:VAL;
12:   var aview:AVIEW;
                (* constraints *)
13:   fun isConsistQueen (xi:ID, di:VAL_Q, _) (xj:ID, dj:VAL_Q, _) =
                        di <> dj and also abs(xi-xj) <> abs(di - dj);
14:   val isValConsist = isConsistQueen;
                (* initialization *)
15:   val agents:L'ID = [1,2,3,4];
16:   val D1:L'VAL = [1,2,3,4];
17:   val D2:L'VAL = [1,2,3,4];
18:   val D3:L'VAL = [1,2,3,4];
19:   val D4:L'VAL = [1,2,3,4];
```

Model declarations for coloring graph problem

```
                (* constraints *)
13:   fun isConsistColoringGraph (xi:ID, di:VAL_Q, _)
(xj:ID, dj:VAL_Q, _) = di <> dj;
14:   val isValConsist = isConsistColoringGraph;
                (* initialization *)
15:   val agents:L'ID = [1,2,3];
16:   val D1:L'VAL = [1,2];
17:   val D2:L'VAL = [2,3];
18:   val D3:L'VAL = [1,2];
```

Model declarations for jobshop problem

```
                (* some declarations of types *)
3:    colset VAL_JobShop = product ID*INT*INT;
                (* resource id, start time, duration *)
4:    colset VAL = VAL_JobShop;
                (* constraints *)
13:   fun isConsistJobShop (xi:ID, (ri,di,si):VAL_JobShop, _) (xj:ID, (rj,dj,
sj):VAL_JobShop, _) = ordering(xi,(ri,di,si),xj,(rj,dj,sj)) andalso overlap((ri,
di,si), (rj,dj,sj));
14:   val isValConsist = isConsistJobShop;
                (* initialization *)
15:   val agents:L'ID = [1,2,3,4];
16:   val D1:L'VAL = [(1,2,0),(1,2,1)];
17:   val D2:L'VAL = [(2,1,0), (2,1,1), (2,1,2)];
18:   val D3:L'VAL = [(2,1,0), (2,1,1), (2,1,2)];
19:   val D4:L'VAL = [(1,1,0), (1,1,1), (1,1,2)];
20:   val ordering_constraints = [(1,2), (3,4)];
```

APPENDIX C

TABLE A1 Results of series of 1000 simulations for common DisCSP algorithms

#queens / Algorithms		3		4		5		6		7	
		min/max	avgr	min/max	avgr	min/max	avgr	min/max	avgr	min/max	avgr
SBT (1)	#msgs	12/12	12.0	3/19	8.3	4/8	5.0	5/139	42.7	6/56	16.6
	#cycles	11/11	11.0	4/20	9.3	5/9	6.0	6/140	43.7	7/57	17.6
	#backs	6/6	6.0	0/8	2.6	0/2	0.5	0/67	18.8	0/25	5.3
	#ngs	5/5	5.0	0/8	2.6	0/2	0.5	0/67	18.8	0/25	5.3
	*	0	2.4	2	2.5	10	2.5	4	15.6	40	14.9
ABT (1)	#msgs	15/20	16.1	6/42	16.4	10/33	15.2	15/253	84.4	21/143	47.7
	#cycles	11/18	13.5	4/31	12.0	5/21	9.1	6/143	46.9	7/69	21.9
	#backs	6/9	6.7	0/13	3.4	0/8	1.2	0/73	21.2	0/34	6.2
	#ngs	5/8	5.7	0/15	4.0	0/9	1.6	0/96	27.8	0/54	9.5
	*	0	2.7	2	3.0	10	3.9	4	25.8	40	32.7
WCS (1)	#msgs	26/47	35.8	12/87	21.4	20/152	27.0	30/1554	301.6	42/1520	159.8
	#cycles	9/17	12.7	4/18	6.1	5/28	6.2	6/180	38.2	7/144	18.6
	#backs	6/13	8.4	0/13	1.2	0/13	0.5	0/115	18.4	0/78	6.0
	#ngs	7/15	10.1	0/33	3.2	0/45	2.1	0/659	110.7	0/656	48.2
	*	0	3.4	2	2.6	10	3.7	4	47.5	40	30.3
SBT (2)	#msgs	12/12	12.0	3/19	8.7	4/8	5.0	5/139	43.5	6/56	16.5
	#cycles	11/11	11.0	4/20	9.7	5/9	6.0	6/140	44.5	7/57	17.5
	#backs	6/6	6.0	0/8	2.9	0/2	0.5	0/67	19.2	0/25	5.2
	#ngs	5/5	5.0	0/8	2.9	0/2	0.5	0/67	19.2	0/25	5.2
	*	0	2.9	2	2.8	10	2.3	4	19.7	40	16.6
ABT (2)	#msgs	16/18	17.3	6/44	18.6	10/38	21.1	15/301	95.4	25/190	65.0
	#cycles	13/17	15.4	4/36	15.7	6/28	14.6	7/186	59.8	10/104	34.4
	#backs	7/8	7.7	0/16	4.4	0/11	3.2	0/102	27.9	0/54	11.1
	#ngs	6/7	6.7	0/17	5.0	0/13	3.9	0/125	33.8	0/75	15.9
	*	0	3.6	2	4.6	10	7.0	4	57.1	40	86.8
WCS (2)	#msgs	35/76	51.1	12/301	65.2	20/678	144.8	45/7708	1299.3	54/7471	997.8
	#cycles	14/29	19.8	4/76	17.5	5/128	29.0	9/1148	196.3	9/913	126.1
	#backs	8/16	10.7	0/36	5.8	0/56	8.8	0/514	76.0	0/338	40.8
	#ngs	7/19	11.3	0/74	12.8	0/166	28.9	0/1968	317.7	0/1993	241.0
	*	0	5.9	2	8.7	10	23.4	4	551.7	40	444.8

Abbreviations: ABT, asynchronous backtracking; SBT, synchronous backtracking; WCS, weak-commitment search.

#queens	8		9		10		11		12		13	
Algorithms	min/max	avgr	min/max	avgr	min/max	avgr	min/max	avgr	min/max	avgr	min/max	avgr
SBT (1)	7/285	46.6	8/342	50.7	9/1029	124.7	10/1298	190.3	11/2523	183.7	12/5062	233.1
	8/286	47.6	9/343	51.7	10/1030	125.7	11/1299	191.3	12/2524	184.7	13/5063	234.1
	0/139	19.8	0/167	21.3	0/510	57.8	0/644	90.1	0/1256	86.4	0/2525	110.5
	0/139	19.8	0/167	21.3	0/510	57.8	0/644	90.1	0/1256	86.4	0/2525	110.5
	91	35.7	297	58.1	450	152.7	732	319.1	935	424.2	985	748.3
ABT (1)	28/617	119.6	36/914	147.8	46/2181	298.1	56/4968	509.9	74/4660	555.3	80/14303	762.5
	8/281	52.2	9/349	58.3	11/828	112.1	12/1714	174.8	16/1372	171.3	14/3672	210.0
	0/150	22.6	0/202	25.1	0/497	55.6	0/1016	92.3	0/833	89.6	0/2241	112.8
	0/257	38.1	0/463	48.7	0/1089	120.8	0/2723	236.0	0/2735	266.5	0/8708	399.3
	87	76.1	282	136.3	444	313.5	739	841.7	930	1159.3	983	2987.5
WCS (1)	56/2360	357.1	72/4889	475.3	90/11923	1254.1	110/17911	2000.1	132/14048	2340.2	156/29753	2958.9
	8/183	32.0	9/328	35.3	10/611	71.6	11/733	93.3	12/524	91.6	13/832	96.5
	0/90	12.0	0/157	12.8	0/298	29.6	0/352	38.6	0/236	36.2	0/373	37.2
	0/1079	133.0	0/2265	193.2	0/6424	609.3	0/10581	1067.3	0/8617	1332.6	0/19769	1801.0
	92	85.6	332	158.1	535	794.2	815	2050.4	972	3007.6	993	6218.3
SBT (2)	7/291	47.5	8/354	55.4	9/1023	116.5	10/2534	195.6	11/2711	190.8	12/5842	249.0
	8/292	48.5	9/355	56.4	10/1024	117.5	11/2535	196.6	12/2712	191.8	13/5843	250.0
	0/142	20.2	0/173	23.7	0/507	53.8	0/1262	92.8	0/1350	89.9	0/2915	118.5
	0/142	20.2	0/173	23.7	0/507	53.8	0/1262	92.8	0/1350	89.9	0/2915	118.5
	92	43.8	291	73.4	453	188.3	734	474.5	921	590.5	973	1266.8
ABT (2)	33/670	156.3	49/1088	183.6	56/2568	394.7	74/7402	686.0	92/9524	796.5	141/24763	1078.9
	12/354	78.6	17/523	83.8	18/1159	172.3	25/3018	279.3	24/3521	296.1	39/8554	367.5
	0/225	37.4	0/338	39.0	1/784	96.4	1/2104	170.2	1/2495	182.5	3/6249	233.3
	0/321	56.9	0/582	65.8	2/1468	179.8	2/4621	357.1	4/6052	430.7	9/16864	623.8
	88	215.2	290	397.6	466	1063.4	719	3071.6	918	4945.2	980	12762.9
WCS (2)	77/15322	2040.7	136/25478	3323.8	171/69641	10521.8						
	11/1604	216.2	17/2255	297.7	19/5273	801.2						
	0/542	67.9	0/742	90.3	0/1630	241.2						
	0/4094	527.4	0/7438	942.4	0/22184	3311.2						
	92	1542.9	321	4099.5	528	43448.0						

TABLE A2 Results of series of 1000 simulations in case (1) for different strategies

#queens		3		4		5		6		7	
Algorithms		min/max	avgr	min/max	avgr	min/max	avgr	min/max	avgr	min/max	avgr
WCSg(1)	#msgs	18/36	25.4	12/66	18.3	20/100	25.6	30/1115	199.5	42/618	109.1
#ngs = 0	#cycles	9/18	12.7	4/22	6.1	5/25	6.4	6/223	39.9	7/103	18.2
	#backs	6/13	8.4	0/13	1.2	0/14	0.7	0/120	19.4	0/60	5.7
	*	0	3.8	2	2.6	10	3.5	4	41.1	40	26.3
WCSs(1)	#msgs	26/55	39.0	12/83	20.0	20/131	26.6	30/1179	238.6	42/961	124.1
	#cycles	10/20	14.4	4/21	6.0	5/27	6.2	6/176	37.2	7/124	17.6
	#backs	7/15	10.0	0/11	1.1	0/10	0.6	0/101	17.5	0/62	5.2
	#ngs	6/18	9.8	0/20	2.1	0/25	1.6	0/299	52.8	0/217	18.7
	*	0	4.1	2	3.0	10	3.5	4	32.6	40	23.9
ABTgc(1)	#msgs	15/18	15.4	6/36	15.9	10/26	13.8	15/221	73.0	21/122	43.4
	#cycles	11/16	12.2	4/24	11.0	5/16	7.9	6/113	37.7	7/53	18.4
	#backs	6/7	6.1	0/9	2.8	0/4	0.5	0/61	16.0	0/24	4.3
	#ngs	5/5	5.1	0/10	3.4	0/5	0.6	0/79	21.1	0/38	6.4
	*	0	2.7	2	2.9	10	3.9	4	23.6	40	32.8
WCSgc(1)	#msgs	27/46	33.6	60	20.3	20/53	24.5	30/1201	195.4	42/654	93.9
	#cycles	10/17	12.1	14	5.8	5/11	5.9	6/148	26.5	7/66	12.7
	#backs	6/10	7.3	7	0.9	0/3	0.3	0/70	10.3	0/29	2.4
	#ngs	7/12	9.3	18	2.8	0/11	1.1	0/461	63.0	0/258	18.0
	*	0	3.3	2	2.6	10	3.4	4	27.0	40	20.8
WCSggc(1)	#msgs	20/34	24.6	12/48	17.5	20/44	23.5	30/740	138.2	42/288	78.3
#ngs = 0	#cycles	10/17	12.3	4/16	5.8	5/11	5.9	6/148	27.6	7/48	13.1
	#backs	6/10	7.5	0/7	0.9	0/3	0.3	0/75	10.9	0/22	2.5
	*	0	2.9	2	2.7	10	3.3	4	24.8	40	20.2
WCSsgc(1)	#msgs	26/48	36.6	12/54	19.2	20/50	24.4	30/895	174.0	42/339	84.6
	#cycles	10/18	13.7	4/14	5.8	5/11	5.9	6/138	28.1	7/46	12.7
	#backs	7/11	8.7	0/7	0.9	0/3	0.3	0/71	11.3	0/19	2.4
	#ngs	6/12	9.1	0/13	1.8	0/7	0.8	0/205	33.4	0/67	8.4
	*	0	4.1	2	2.7	10	3.3	4	23.3	40	19.1
WCSsggc(1)	#msgs	20/38	27.4	12/48	17.5	20/48	23.6	30/815	141.9	42/324	77.4
#ngs = 0	#cycles	10/19	13.7	4/16	5.8	5/12	5.9	6/163	28.4	7/54	12.9
	#backs	7/11	8.7	0/7	0.9	0/3	0.3	0/78	11.4	0/23	2.5
	*	0	3.9	2	2.8	10	3.9	4	21.3	40	19.2

Abbreviations: ABT, asynchronous backtracking; WCS, weak-commitment search.

#queens Algorithms	8		9		10		11		12		13	
	min/max	avgr	min/max	avgr	min/max	avgr	min/max	avgr	min/max	avgr	min/max	avgr
WCSg(1)	56/1470	218.7	72/1816	284.7	90/3870	628.2	110/450	903.2	132/6226	984.9	156/7320	1160.7
#ngs = 0	8/210	31.2	9/227	35.6	10/430	69.8	11/545	90.3	12/566	89.5	13/610	96.7
	0/115	11.7	0/102	13.0	0/204	28.7	0/267	36.9	0/257	35.3	0/270	37.1
	92	69.2	338	121.4	554	462.1	824	1140.1	963	1770.5	993	3055.9
WCSs(1)	56/1604	250.2	72/2015	304.0	90/4304	635.7	110/5394	913.4	132/6171	966.5	156/7501	1162.4
	8/177	29.5	9/193	31.7	10/383	57.9	11/430	74.9	12/443	72.7	13/505	80.5
	0/89	10.6	0/100	10.9	0/170	22.3	0/194	29.1	0/208	26.9	0/218	29.4
	0/365	43.7	0/471	50.4	0/857	114.9	0/1094	164.5	0/1298	166.4	0/1441	196.2
	92	52.4	329	81.9	528	187.2	828	322.0	962	417.7	998	603.0
ABTgc(1)	28/487	99.0	36/539	118.1	48/1732	246.2	55/3794	404.8	75/3461	431.0	83/8161	542.3
	8/202	39.7	10/197	43.0	11/603	86.0	12/1264	131.7	15/1050	128.9	16/2255	150.4
	0/104	15.3	0/105	16.5	0/332	39.7	0/709	65.0	0/583	62.9	0/1282	74.8
	0/162	24.3	0/180	28.3	0/651	77.9	0/1589	144.0	0/1361	155.1	0/3332	204.6
	90	90.0	287	125.0	464	264.2	742	554.3	926	746.2	979	1373.6
WCSgc(1)	56/1327	198.2	72/1879	243.3	90/3029	535.3	110/4710	746.5	132/8167	878.6	156/7527	1027.1
	8/106	20.0	9/124	21.0	10/167	35.1	11/217	40.7	12/310	41.0	13/240	41.4
	0/46	5.3	0/53	5.0	0/72	10.6	0/91	12.2	0/124	11.5	0/90	11.0
	0/585	57.9	0/887	75.3	0/1670	219.1	0/2540	339.0	0/4757	427.5	0/4647	530.1
	92	47.0	332	79.3	536	213.3	829	403.2	973	650.0	995	974.7
WCSggc(1)	56/777	143.8	72/880	161.7	90/1638	328.4	110/2730	419.2	132/3190	444.6	156/2376	513.9
#ngs = 0	8/111	20.5	9/110	20.2	10/182	36.5	11/273	41.9	12/290	40.4	13/198	42.8
	0/45	5.5	0/45	4.7	0/78	11.2	0/110	12.7	0/110	11.4	0/74	11.6
	92	44.1	327	68.6	540	172.9	826	299.3	964	418.7	995	620.0
WCSsgc(1)	56/768	164.2	72/1052	190.5	90/2057	364.7	110/2465	489.0	132/2551	497.9	156/3792	583.1
	8/90	20.3	9/105	21.0	10/188	34.7	11/203	41.9	12/196	39.4	13/263	42.5
	0/38	5.5	0/48	5.0	0/75	10.4	0/80	12.7	0/66	10.8	0/99	11.4
	0/153	21.9	0/212	22.5	0/372	52.1	0/435	69.5	0/395	64.3	0/636	73.3
	92	39.6	323	63.3	521	124.1	840	199.9	962	277.9	995	410.3
WCSsggc(1)	56/868	138.6	72/688	163.3	90/1926	315.2	110/1970	409.0	132/2343	441.4	156/2124	494.9
#ngs = 0	8/124	19.8	9/86	20.4	10/214	35.0	11/197	40.9	12/213	40.1	13/177	41.2
	0/58	5.1	0/34	4.8	0/87	10.6	0/77	12.3	0/85	11.2	0/68	11.0
	92	37.8	333	61.3	522	122.1	832	193.1	962	279.3	992	408.6

TABLE A3 Results of series of 1000 simulations in case (2) for different strategies

#queens		3		4		5		6		7	
Algorithms		min/max	avgr	min/max	avgr	min/max	avgr	min/max	avgr	min/max	avgr
WCSs(2)	#msgs	35/78	51.5	12/249	63.2	20/694	128.3	40/7358	1036.0	48/4935	777.4
	#cycles	14/30	19.9	4/64	17.2	5/135	26.8	8/1175	167.8	8/677	108.3
	#backs	8/16	10.8	0/30	5.7	0/58	7.8	0/524	64.3	0/239	34.6
	#ngs	7/19	11.5	0/60	11.6	0/154	20.9	0/1483	197.1	0/873	127.7
	*	0	6.9	2	9.8	10	23.6	4	372.8	40	298.5
WCSg(2)	#msgs	24/56	34.8	12/204	47.9	20/580	103.7	45/4160	754.4	54/5148	660.9
#ngs = 0	#cycles	12/28	17.4	4/68	16.0	5/145	25.9	9/832	150.9	9/858	110.2
	#backs	7/15	10.0	0/35	5.4	0/67	7.8	0/375	60.5	0/318	36.6
	*	0	5.5	2	8.4	10	21.5	4	353.4	40	352.2
ABTgc(2)	#msgs	15/17	15.8	6/36	16.2	10/25	16.6	18/228	83.1	23/131	51.5
	#cycles	12/14	13.3	4/25	12.1	5/16	10.2	8/122	44.9	9/60	22.9
	#backs	6/7	6.1	0/9	2.6	0/4	0.5	0/68	19.6	0/28	5.4
	#ngs	5/6	5.1	0/10	3.1	0/4	0.6	0/86	24.6	0/40	7.5
	*	0	3.5	2	4.1	10	6.2	4	51.0	40	82.9
WCSgc(2)	#msgs	33/60	44.6	12/117	37.6	20/164	45.5	40/1983	414.4	54/1032	202.3
	#cycles	13/24	17.5	4/31	10.9	5/33	10.8	8/286	63.2	9/127	28.1
	#backs	7/14	9.1	0/13	2.1	0/10	0.8	0/123	22.2	0/46	5.8
	#ngs	6/13	9.5	0/24	5.0	0/32	2.4	0/563	98.5	0/270	33.5
	*	0	5.7	2	5.7	10	9.4	4	114.5	40	107.4
WCSggc(2)	#msgs	22/36	27.8	12/54	26.0	20/64	39.1	35/715	158.8	48/396	112.7
#ngs = 0	#cycles	11/18	13.9	4/18	8.7	5/16	9.8	7/143	31.8	8/66	18.8
	#backs	6/10	7.6	0/7	1.1	0/5	0.5	0/69	10.9	0/25	2.9
	*	0	4.9	2	4.9	10	8.8	4	66.0	40	94.5
WCSsgc(2)	#msgs	33/61	44.6	12/108	33.8	20/119	45.9	30/1734	351.1	54/954	174.8
	#cycles	13/24	17.5	4/30	10.1	5/26	10.9	6/280	58.2	9/132	26.1
	#backs	7/14	9.1	0/11	1.7	0/7	0.8	0/118	20.0	0/45	5.0
	#ngs	6/13	9.4	0/18	3.4	0/16	2.2	0/339	59.9	0/162	18.0
	*	0	6.6	2	5.8	10	9.8	4	97.8	40	101.6
WCSsggc(2)	#msgs	22/44	30.9	12/51	25.9	20/64	39.2	35/980	163.9	54/330	113.3
#ngs = 0	#cycles	11/22	15.4	4/17	8.6	5/16	9.8	7/196	32.8	9/55	18.9
	#backs	7/14	9.0	0/7	1.1	0/4	0.5	0/94	11.5	0/22	3.0
	*	0	5.3	2	4.8	10	8.9	4	61.6	40	91.7

Abbreviations: ABT, asynchronous backtracking; WCS, weak-commitment search.

#queens / Algorithms	8		9		10		11		12		13	
	min/max	avgr	min/max	avgr	min/max	avgr	min/max	avgr	min/max	avgr	min/max	avgr
WCSs(2)	112/7618	1546.2	125/17500	2078.7	186/39744	5581.2						
	15/901	185.7	15/1839	220.6	20/3744	527.2						
	0/314	58.4	0/582	66.0	0/1185	158.9						
	0/1311	246.5	0/2788	314.0	0/6220	836.0						
	92	796.6	324	1403.7	535	6789.6						
WCSg(2)	70/9436	1341.9	104/14248	2255.5	252/45999	6000.7						
#ngs = 0	10/1348	191.7	13/1781	281.9	28/5111	666.7						
	0/470	62.3	0/595	87.5	3/1657	206.0						
	92	1146.0	323	3192.0	526	25893.3						
ABTgc(2)	30/494	112.2	39/570	134.7	53/1756	273.2	66/4602	453.5	92/3798	487.2	112/11547	590.9
	10/227	47.5	13/248	52.7	15/739	107.1	17/1878	172.5	19/1431	173.5	21/4242	201.1
	0/126	19.1	0/148	21.9	0/472	55.8	0/1245	99.5	0/935	101.0	0/2954	121.2
	0/175	27.3	0/211	32.5	0/745	89.4	0/2097	169.7	0/1630	183.6	0/5473	233.3
	92	181.6	291	346.7	450	730.3	730	1560.8	923	2351.0	981	4096.9
WCSgc(2)	70/2350	423.4	112/3713	543.6	117/11655	1300.9	170/21783	1893.4	176/13023	2186.9	252/14758	2676.5
	10/237	47.1	14/321	51.8	13/850	100.8	17/1382	126.1	16/717	127.4	21/727	136.4
	0/90	12.2	0/118	13.0	0/317	29.5	0/474	37.1	0/247	35.9	0/233	37.6
	0/691	93.8	0/1145	129.0	0/4005	393.9	0/7963	632.2	0/5136	785.1	0/6187	1039.8
	92	250.3	339	454.7	535	1298.5	838	2538.0	969	3894.2	991	6211.1
WCSggc(2)	70/840	191.0	104/776	226.3	117/1836	389.5	170/3180	512.4	198/2662	530.0	228/3192	622.5
#ngs = 0	10/120	27.3	13/97	28.3	13/204	43.3	17/318	51.2	18/242	48.2	19/266	51.9
	0/54	6.1	0/38	5.6	0/83	11.1	0/133	13.5	0/91	11.2	0/92	11.7
	92	195.2	326	357.1	544	725.0	840	1269.3	970	1937.0	992	3117.1
WCSsgc(2)	70/1504	304.7	96/2188	385.9	135/5661	803.7	170/6302	1050.3	187/7601	1154.4	240/8596	1343.5
	10/178	38.2	12/232	42.5	15/519	76.9	17/526	90.5	17/580	91.0	20/602	97.5
	0/63	9.0	0/70	9.7	0/188	21.4	0/181	25.2	0/192	24.4	0/203	25.5
	0/266	37.0	0/332	45.6	0/990	111.7	0/1042	145.0	0/1221	153.1	0/1372	173.2
	92	208.6	333	377.3	548	791.0	837	1354.4	972	2091.4	993	3257.6
WCSsggc(2)	77/707	180.6	104/752	225.6	135/1575	385.6	170/2010	494.2	198/2090	545.4	216/2220	601.7
#ngs = 0	11/101	25.8	13/94	28.2	15/175	42.8	17/201	49.4	18/190	49.6	18/185	50.1
	0/36	5.4	0/36	5.5	0/64	10.9	0/80	12.6	0/71	11.7	0/65	11.1
	92	182.6	330	340.2	518	652.9	833	1140.3	965	1826.5	992	2898.8

- A colored Petri net (CPN) model for DisCSP algorithms is detailed; to the best of our knowledge no similar model has been published, yet;
- A thorough analysis of the existing research on DisCSP (applicability, existing algorithms, how they compare) is provided;
- The CPN model allowed a comparison between 3 representative DisCSP algorithms, with important conclusions regarding the behavior for nonidealized conditions;
- Three strategies are introduced and the model proves how these can make DisCSP algorithms more practicable;
- The proposed model (which is available for open use) is close to a prototype, and thus it can be a valuable tool for deployment of the new manufacturing distributed control shemes (eg, holonic and multiagent systems.)