Distributed Constraint Optimization for Task Allocation in Self-Adaptive Manufacturing Systems

Joseph Hirsch, Martin Neumayer, Hella Ponsar, Oliver Kosak and Wolfgang Reif
Institute for Software & Systems Engineering, University of Augsburg, Germany
E-Mail: joseph.hirsch@uni-a.de, {neumayer, ponsar, kosak, reif}@isse.de

***Abstract*—Adaptive manufacturing systems consist of many autonomous agents working together in an ever-changing environment. Therefore, collectively deciding which agent performs what task is a key issue and widely studied. However, many approaches towards this issue assume (partially) centralized control, require implementing proprietary algorithms, or cannot provide any guarantees regarding their runtime or communication overhead. To address these problems, we investigate the use of distributed constraint optimization (DCOP) in this context: We present a DCOP model built on freely available algorithms to distribute the problem among the agents that cooperate to solve it. Furthermore, we compare this decentralized approach to a centralized one by measuring the runtime in a set of system configurations with an increasing number of agents. While the DCOP approach works well in small system configurations, our results indicate poor scalability compared to the central approach when increasing the number of agents. We conclude that, although the DCOP approach has desirable properties, it is unsuitable for larger practical applications with dozens or hundreds of agents.**

***Index Terms*—DisCSP, DCOP, task allocation, self-adaption, flexible manufacturing systems, multi-agent systems**

I. Motivation

Adaptive manufacturing systems are an interesting industrial application for Collective Adaptive Systems (CAS) [1]. They consist of many heterogeneous agents, such as robots and vehicles, working together in a dynamically changing environment to fulfill a common goal, i.e., manufacture the desired products. One major challenge in adaptive manufacturing systems is collectively reconfiguring to deal with changes in the environment or the system itself, e.g., new products to be manufactured or agent breakdown. We focus on reconfiguration in the sense of finding a valid task allocation [2] and routing: The agents have to assign operations required to manufacture a product to agents offering these operations and then ensure products are transported accordingly.

Approaches employing a central controller for decision-making may lack robustness due to a single point of failure [3]–[5] and scalability due to increased communication costs [6]. Therefore, research in CAS focuses on modeling systems as a set of autonomous agents that interact at runtime to make decisions. In the domain of adaptive manufacturing systems, however, many approaches following this methodology either assume (partially) centralized control [7], [8] or require implementing proprietary algorithms [8], [9]. In addition, the message and runtime overhead of some approaches depends on the current system configuration and is therefore hard to quantify in advance [10], [11].

Distributed constraint optimization promises to relax the problem of centralized control by distributing the decision variables of the problem to the agents [12]. The agents then cooperate to solve the problem, eliminating the need for a central controller. Several algorithms, such as the distributed pseudotree-optimization procedure (DPOP), claim to offer scalability [13] and are publicly available [14]. Hence, we investigate the use of distributed constraint optimization for task allocation and routing in adaptive production systems. We aim for a decentral control mechanism that combines the advantages of well-tested and freely available algorithms with predictable runtime and message overhead.

The remainder of this paper is structured as follows: Section II introduces the problem using a motivating example. In Section III, we summarize related work covering task allocation in adaptive manufacturing systems and Distributed Constraint Optimization. Section IV presents our Distributed Constraint Optimization approach towards solving the problem and explains our modeling in greater detail. We evaluate our approach theoretically and experimentally on several system configurations, also comparing our approach to a centralized Constraint Optimization approach in Section V. Section VI discusses the results and concludes the paper.

II. Problem Definition

A. Organic Design Pattern

Adaptive manufacturing systems in this paper are modeled using the Organic Design Pattern (ODP) [15], [16]: The active participants of the system are called *agents*, while *products* are processed by the agents[1]. The blueprint on how to manufacture a product is referred to as a *task*. A task consists of a sequence of *capabilities* in a specified order. The task's state specifies which capabilities were already applied and which capability has to be applied next. To meet the required capabilities of a product, each agent provides a set of capabilities. Further, a *connection matrix* defines which agents are connected. Only connected agents can hand over products. We now focus on finding adequate *roles*. A role comprises three aspects:

1) Where an agent gets products from.
2) Which capabilities the agent applies.

[1]In contrast to previous work, we refrain from using the term *resource* as it is ambiguous in the context of manufacturing systems [2].

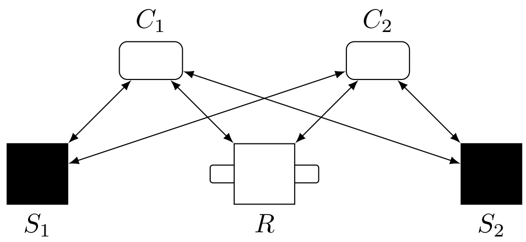


Fig. 1. Motivating example: A manufacturing system consisting of two storage agents S_1 and S_2, two carts C_1 and C_2, and a robot R. An arrow between two agents indicates that they are connected and can therefore exchange products.

3) Where it gives products to after applying the capabilities.

A valid system configuration associates each capability of a task with an agent in a corresponding role. It also ensures that successive agents are connected, either directly or via another agent.

B. Motivating Example

Consider the system configuration depicted in Fig. 1: A manufacturing system consisting of 5 agents is shown. Agents S_1 and S_2 are storages producing and consuming products, C_1 and C_2 are so-called carts, transporting products, while R is a robot processing products, e.g., by drilling a hole. The edges between the agents represent the connection matrix: C_1 and C_2 are connected to any storage or robot. The problem can now be defined as finding corresponding roles for a given task, e.g., $(Produce, Drill, Consume)$.

In language terms[2], the following roles represent an exemplary solution to our problem:

- Storage S_1 "produces" products and hands them over to C_1.
- Cart C_1 transports products from S_1 to R.
- Robot R receives products from C_1, applies the capability drill and hands the finished products over to C_2.
- Cart C_2 transports products from R to S_2.
- Storage S_2 receives and "consumes" products from C_2.

Note that storages producing products do not specify where products come from and storages consuming products do not specify where products are given to afterward.

III. Related Work

The problem described above is closely related to the Flexible Job Shop Scheduling Problem (FJSP) [17] with transportation constraints. The FSJP with transportation constraints further includes the sequencing of operations and transports. However, we focus on the assignment of operations to suitable machines and the routing of products. The sequencing of operations and transports emerges through the interaction of the agents at runtime.

[2]For a more formal description, refer to the MiniZinc model in our replication package: https://github.com/isse-augsburg/ecas2021-DisCSP

A. Dynamic Control of Adaptive Manufacturing Systems

Nafz et al. present a universal reconfiguration mechanism to control adaptive manufacturing systems in [7]. The authors propose to model the agents with the ODP and state the properties for a valid task allocation as Object Constraint Language (OCL) constraints. The agents then monitor these properties and whenever the agents notice a constraint violation, they gather a global view of the system. This view is then transformed into a Constraint Satisfaction Problem using the OCL constraints. A constraint solver can then calculate a new task allocation centrally and distribute it to the agents. This approach is easy to implement as an off-the-shelf constraint solver can be used. However, it suffers from the disadvantages of central control.

To alleviate the problem of central control, in [8] and [10], the authors present a reconfiguration mechanism based on coalition formation. If an agent notices a constraint violation, it creates a new coalition and becomes its leader. The leader recruits neighboring agents until it can calculate a feasible task allocation. The leader then distributes the result of the task allocation and dismisses the coalition. Thus, information and control are only partially centralized.

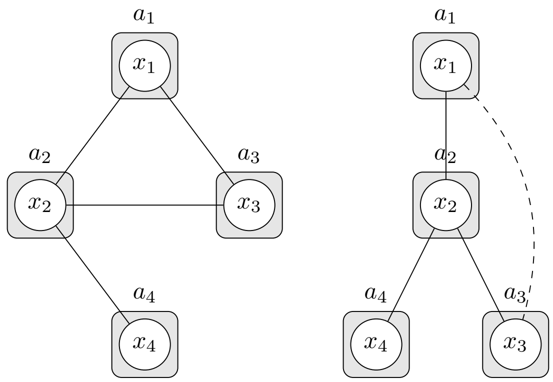
A wave-like reconfiguration mechanism for systems based on the ODP is presented in [9]. If an agent loses a capability, it requests assistance from the neighboring agents. A neighboring agent capable of replacing the broken capability might answer the request and both agents swap roles. If a single swap is not sufficient to restore a valid system configuration, a wave of swaps may run through the system. So instead of forming coalitions, this approach is entirely decentralized. However, a complete breakdown of an agent may go unnoticed as agents have to requests assistance themselves [10]. The wave-like approach is similar to a decentralized swapping-based min-conflicts local search heuristic which is used to schedule workforce in [18]. Both the coalition formation and the wave-like approach require implementing a proprietary, distributed algorithm. Further, in both approaches, the reconfiguration overhead is hard to quantify as it is dependent on the available redundancy and the system configuration [10], [11].

In [19], the authors compare centralized and decentralized approaches towards allocating machines and operators in a slightly different manufacturing context. In their simulation, operators run machines under qualification constraints to process products. Further, perturbations such as machine failure or operator unavailability occur. The authors compare distributed constraint optimization approaches and their own heuristic extensions for this problem. They conclude that distributed constraint optimization approaches are suited for this use case.

B. Distributed Constraint Optimization with DPOP

Following [12], a Distributed Constraint Satisfaction Problem (DisCSP) is a tuple $\langle A, X, D, C, \alpha \rangle$, where

- $A = \{a_1, \dots, a_m\}$ is a finite set of agents.
- $X = \{x_1, \dots, x_n\}$ is a finite set of variables.
- $D = \{D_1, \dots, D_n\}$ denotes finite sets of domains for the variables in X. D_i corresponds to possible values of x_i.



(a) Constraint Graph (b) Pseudo-Tree

Fig. 2. Typical representations of an example DCOP with four agents a_1 to a_4, each controlling one variable x_1 to x_4. Edges indicate constraints between the connected variables. Figure adapted from [12].

- $C = \{c_1, \ldots, c_k\}$ is a finite set of constraints over a subset of X. A constraint specifies how variable values should be related to each other by disallowing certain combinations of values.
- $\alpha : X \rightarrow A$ is a function that assigns each variable in X to an agent $\alpha(x)$.

In a Distributed Constraint Optimization Problem (DCOP), the set of constraints C is replaced by a set of weighted constraints $F = \{f_1, \ldots, f_k\}$ that indicate a degree of preference about their violation by assigning costs or utilities to combinations of values or disallowing certain combinations [12].

DCOPs and DisCSPs are often represented as a Constraint Graph, where agents controlling their respective variables are shown as nodes, while constraints are shown as edges. An example Constraint Graph is shown in Fig. 2a.

One well-known algorithm to solve DCOPs and DisCSPs is the distributed pseudo-tree optimization procedure (DPOP) [13]. As the name suggests, DPOP relies on the agents forming a so-called pseudo-tree and communicating via messages. DPOP consists of three phases:

1) Pseudo-tree formation: The agents form a pseudo-tree, e.g., by performing a depth-first search [13]. An exemplary pseudo-tree is shown in Fig. 2b: Agents that are connected in the constraint graph are either connected via tree edges (solid lines) or backedges (dotted lines).
2) UTIL propagation: Starting from the tree leaves, the agents compute their optimal utilities considering their adjacent tree edges and backedges. E.g., agent a_3 in Fig. 2b must consider x_2 and x_1, while a_4 must only consider x_2. Finding the optimal value combination is done by dynamic programming. The agents then propagate the optimal value combination to their parent in a UTIL message.
3) VALUE propagation: The root node collects all the UTIL messages of its children and then computes the optimal overall utility and sends a VALUE message to its children, informing them about its decision. The children adapt the root node's decision and pass the UTIL message on to their children until they reached every leave node.

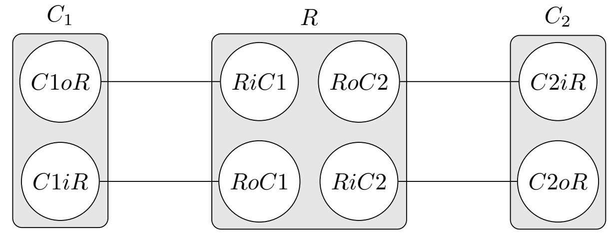


Fig. 3. Excerpt of the constraint graph of our motivating example with the connected agents C_1, C_2 and R.

For a more detailed description of DPOP and its phases, refer to [13]. In our implementation, we use the DPOP implementation of the FRODO framework [14], which is an open-source framework for distributed combinatorial optimization written in Java [20].

IV. Implementation

In the following section, we describe our implementation of the reconfiguration as a DisCSP. We provide a model description in plain words, present a formal model, and elaborate on the assumptions and limitations of our model.

A. Model Description

The main difficulty in modeling is to use local agent knowledge only. Each agent knows about its capabilities and its direct neighbors, i.e., other agents it can transfer products to and receive products from directly. To model this neighborhood, each agent has two variables for each of its neighbors: One variable represents an input from this neighbor, the other variable represents an output to this neighbor. The variables are named as follows: `<agent><direction><neighbor>`, where `<direction>` is either i for input or o for output. $C1oR1$, for example, is a variable located at agent $C1$ and represents an output to agent $R1$. To illustrate this variable naming, Fig. 3 revisits our motivating example and presents an excerpt of the corresponding constraint graph.

Transports and the state of the task are represented by the values of the agents' variables. Therefore, the variables' domains are the integer numbers that represent the state of the task. Variable values default at -1 which means that no products are transferred along the respective connection, e.g., $C1oR1 = -1$ means no products are output from $C1$ to $R1$. If a product's state does not change while at the same agent, e.g., $C1iS1 = 1$ and $C1oR = 1$, the agent does not apply any capability but instead transports the product. However, if an agent receives a product with an initial task state and outputs it with a higher task state, it has applied the corresponding capabilities. E.g., agent R might receive a product from agent C_1 with task state 1 ($RiC1 = 1$), apply the drill capability, and output the product to agent C_2 with task state 2 ($RoC2 = 2$).

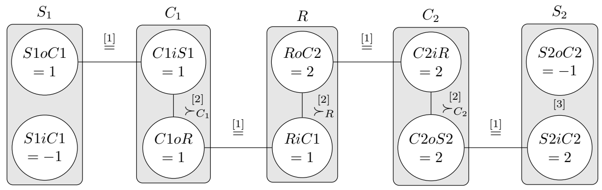


Fig. 4. Example solution for the Task Allocation of PRODUCE DRILL CONSUME: $S1$ produces and outputs products to C_1, C_1 transports products to R without applying any capability. At R products are drilled, i.e., the state changes from 1 to 2, and given to cart C_2, which transports products to S_2 where they are consumed. Constraints are referenced in [ ]. $\succ_a$ is a relation that formalizes correct application, i.e., the agent does only apply capabilities it has, which is further described in Section IV-B. Not all variables with the value -1 are shown.

Fig. 4 depicts the exemplary solution to our motivating example as lined out in Section II-B. Any solution has to ensure that the following three types of constraints hold:

1) The values of matching output and input variables are the same. In the example of Fig. 4:
$$C1oR = RiC1 \wedge \ldots \wedge RiC2 = C2oR.$$
2) The agents apply capabilities correctly, i.e., if an input variable is $x \neq -1$, exactly one output variable has to be $y \geq x$, and the agent has to have all capabilities of the task between the states x and y.
3) Under the previous constraints, a variable assignment where all variables are -1 is valid. To ensure production, we add an auxiliary constraint: A storage has to ensure that one of its input variables is not -1 but the last state of the task before CONSUME.

B. Formal Model

The DisCSP can be formalized as follows: Given

- A: the agents of the system,
- X: the variables of the DisCSP,
- $|t|$: the number of capabilities in the task,
- $\succ_a$: a relation of correct capability applications for agent a. $\succ_a$ contains pairs of states (s_1, s_2): If $(s_1, s_2) \in \succ_a$, a can change the state of a product from s_1 to s_2 by applying its capabilities. $\succ_a$ prevents that an agent is instructed to apply a capability it does not have.

Satisfy following constraints:

$$\forall a, a' \in A: \; aIa' \in X \rightarrow a'Oa \in X \wedge aIa' = a'Oa \quad (1)$$

$$\begin{aligned} &\forall a, a' \in A: \; aOa' \in X \wedge aOa' \neq -1 \rightarrow && \text{(2a)}\\ &\quad \exists aIa'' \in X : (aIa'', aOa') \in \succ_a && \text{(2b)}\\ &\quad \wedge (aIa'' = -1 \vee \neg \exists aIa''' \in X : && \text{(2c)}\\ &\quad\quad a'' \neq a''' \wedge aIa'' = aIa''') && \text{(2d)} \end{aligned}$$

$$\exists a' \in A: \; aIa' \in X \wedge aIa' = |t| - 1 \quad (3)$$

(1) The inputs of agent a match the outputs of agent a'.

(2a) For all outputs that are not -1, (2b) exists a correct corresponding input, (2c) and the input is the only input with this state, (2d) or is -1.

(3) An agent exists with an input of the products at the last state before consumption. This is a necessary auxiliary constraint to make sure that not all variables are set to -1.

C. Assumptions and Limitations

Our model does not consider situations where several agents need to cooperate to complete one transport or capability. We further assume that the last agent of a task is known, i.e., we know where the finished products are consumed or stored. Consequently, we also assume that each task ends with the capability CONSUME. One limitation of our model is that it cannot produce assignments where an agent receives products it has already received before. While this modeling prevents deadlocks, it also restricts the feasible system configurations severely as it rules out cyclic configurations [21]. In our following evaluation, we additionally assume that all stationary agents like robots and storages are connected to all carts but not to other stationary agents. Carts are connected to all stationary agents respectively.

V. Evaluation

A. Complexity Analysis

Solving CSPs is NP-hard [22]. All known complete CSP algorithms, therefore, have an exponential complexity of $O\left(d^{|X|}\right)$ in the worst case, where $|X|$ is the number of variables and d is the size of the domain. In our case, d is the length of the task, and $|X|$ can be calculated using the following formula: $|X| = 4 \cdot |A_{Carts}| \cdot (|A| - |A_{Carts}|)$, where A is the set of agents in the system and A_{Carts} the set of carts. We assume that every cart is connected to every stationary agent. More generally spoken, the number of variables depends on the number of neighbor-relations in the system since each neighbor-relation requires 4 variables (2 variables for each direction). We refer to the number of neighbor-relations as v in the following. In a system with n agents where the share of carts is c, v is given by $v = (nc) \cdot (n(1-c))$ which simplifies to $n^2(c - c^2)$. While the share of carts is application-specific, the maximum for v with a given n is at $c = 0.5$, i.e., half of the agents are carts. Under the assumptions stated in Section IV, the complexity of the reconfiguration is therefore given by $O(d^{4v})$. Treating each pair of matching output and input variables as one variable would reduce the complexity to $O(d^{2v})$.

B. Experimental Evaluation

To evaluate the model presented in Section IV, we implement it using the DPOP algorithm from the FRODO framework. We then run several experiments with different system configurations, measuring the runtime needed to come up with a solution. Further, we compare the runtime against a central model that does not distribute the problem among the agents but instead assumes central knowledge. The implementation of the central model follows [8]. However, compared to the authors in [8], our model is written in MiniZinc [23] and solved using the Gecode solver.

We investigate the following questions in our evaluation:

1) How does the runtime of the distributed model compare to the centralized reconfiguration model?
2) How does the runtime of the two approaches scale with an increasing number of agents and neighbor-relations?

C. Experimental Setup

As seen in the complexity analysis, the complexity of the problem rises with the length of the task and the number of neighbor-relations. Therefore, we evaluate a set of configurations (see Table I) covering different task lengths and a varying number of agents and neighbor-relations. Each configuration is run 10 times and results are averaged.

For every configuration, we follow the given procedure:

1) We initialize our system with a given valid configuration where each agent performs one capability.
2) To start the reconfiguration process, we then simulate the failure of a predefined capability in one of the agents.
3) We measure the time for solving the resulting constraint problem.

In this setup, we ignore the runtime needed for the organization and synchronization of the agents before and after the calculation. As we simulate the execution on a single machine, inter-machine communication delays are also ignored.

D. Experimental Results

The results of 90 runs are shown in Table I. Fig. 5 shows the runtime of every configuration in a boxplot. Fig. 6 displays the runtime in dependency of the number of neighbor-relations v and compares central and distributed solving. To do so, the runtime results of configurations with the same value of v were averaged. The results show some mentionable features:

a) Runtime: For systems with up to 4 neighbor-relations, the distributed solving is faster than the centralized. However, both approaches have a runtime of less than 0.5s, which we consider suitable for a practical application. For larger systems with 6 or more neighbor-relations, the runtime of the distributed solving underlies an exponential growth. In systems with 8 relations, the runtime already averages around 6s, and in even more complex systems, the runtime reaches a magnitude of several minutes. Therefore, we argue that the distributed approach might be infeasible in practical applications with dozens or hundreds of machines. The centralized approach has a constant time complexity in our experiments: All reconfiguration problems were solved in less than 0.5s, even configurations with up to 10 neighbor-relations.

b) Scalability: The distributed solving of the reconfiguration problem underlies exponential growth, while the centralized solving seems constant in time. For large systems, the runtime of the distributed calculation is scattered over several orders of magnitude. The distributed solving of the reconfiguration problem, therefore, clearly does not scale well on larger systems.

TABLE I
AVERAGE (avg) AND MINIMAL (min) RUNTIME OF DIFFERENT SYSTEM CONFIGURATIONS IN MILLISECONDS. d IS THE LENGTH OF THE TASK, v IS THE NUMBER OF NEIGHBOR-RELATIONS.

Conf.	d	Carts	Robots & Storages	v	$avg(T)[ms]$	$min(T)[ms]$
a	3	1	2	2	70	63
b	3	1	3	3	81	67
c	3	2	2	4	170	125
d	3	1	4	4	94	81
e	3	2	3	6	245	213
f	4	2	3	6	589	427
g	4	2	4	8	4114	3519
h	4	3	3	9	53198	20048
i	4	2	5	10	79717	68616

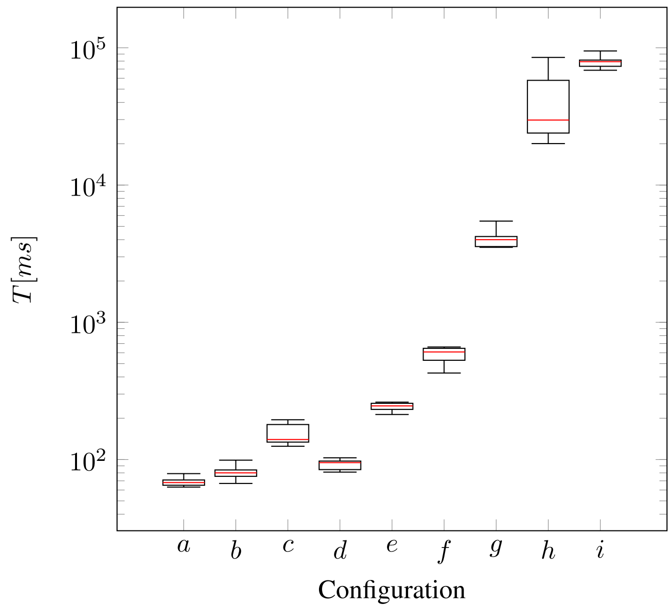


Fig. 5. Runtime of distributed solving different system configurations (see Table I) in milliseconds. The time axis is logarithmically scaled.

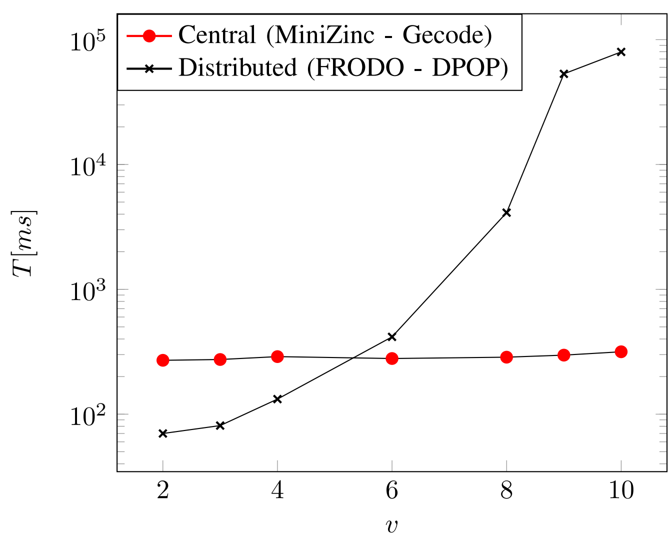


Fig. 6. Average runtime of the central and distributed solving in milliseconds depending on the number of neighbor-relations. Configurations with the same v were averaged. The time axis is logarithmically scaled.

VI. Conclusion and Discussion

In this paper, we addressed the task allocation problem in self-organizing manufacturing systems. We investigated the use of distributed constraint optimization to calculate a valid agent-role mapping. A possible DisCSP model of the task allocation problem has been introduced. This enabled a decentralized solution calculation that does not require a central specialized agent and therefore relaxes the problem of a single point of failure. We evaluated the runtime of the system to find out if it is suitable for a practical application. For the execution of the DPOP algorithm, we used the publicly available FRODO framework. Our evaluation results indicate that the calculation is fast (less than 0.5 seconds) in small systems. However, the runtime of the DisCSP model grows exponentially with the number of neighbor relationships. In our experiments, the DisCSP model reaches an average runtime of over a minute, even in configurations with less than 10 agents, while the MiniZinc model maintains a constant runtime. Therefore, we argue that the DisCSP model is unsuitable for practical application with dozens or even hundreds of agents. While this is clearly a negative result, we want to contribute to reducing the positive result publication bias in the field of adaptive systems [24] and foster debate on when to prefer distributed over centralized control.

Further research is needed to explain why the scalability of the two approaches differs so greatly. In particular, it would be interesting to investigate the influence of the different frameworks on the runtime. In order to use distributed constraint optimization for task allocation in self-organizing manufacturing systems, it also seems necessary to reduce the complexity of the problem, which is left as future work. A way to reduce the complexity of the system is to reduce the neighbor-relations. We assumed a single manufacturing facility where carts can reach every other agent and are therefore fully connected. The number of neighbor-relations can be decreased if not all of the connections of a cart are considered in the DisCSP model. If no solution can be found, these left-out connections would have to be added again. Another way to reduce complexity would be to do a neighbor detection of the robots and storages to see which cart connects the agent to which other agents and then reduce the problem to the processing agents without considering the carts in between

References

[1] D. Sanderson, N. Antzoulatos, J. C. Chaplin, D. Busquets, J. Pitt, C. German, A. Norbury, E. Kelly, and S. Ratchev, "Advanced manufacturing: An industrial application for collective adaptive systems," in *2015 IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, 2015, pp. 61–67.

[2] Y. Chevaleyre, U. Endriss, J. Lang, P. Dunne, M. Lemaitre, N. Maudet, J. Padget, S. Phelps, J. Rodriguez-Aguilar, and P. Sousa, "Issues in multiagent resource allocation," *Informatica*, vol. 30, pp. 3–31, 2006.

[3] P. Leito, "Agent-based distributed manufacturing control: A state-of-the-art survey," *Engineering Applications of Artificial Intelligence*, vol. 22, no. 7, pp. 979 – 991, 2009.

[4] W. Shen and D. H. Norrie, "Agent-based systems for intelligent manufacturing: a state-of-the-art survey," *Knowledge and information systems*, vol. 1, no. 2, pp. 129–156, 1999.

[5] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *Journal of scheduling*, vol. 12, no. 4, pp. 417–431, 2009.

[6] W. Yeoh and M. Yokoo, "Distributed problem solving," *AI Magazine*, vol. 33, no. 3, pp. 53–65, Sep. 2012.

[7] F. Nafz, F. Ortmeier, H. Seebach, J.-P. Steghöfer, and W. Reif, "A universal self-organization mechanism for role-based organic computing systems," in *Autonomic and Trusted Computing*, J. González Nieto, W. Reif, G. Wang, and J. Indulska, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 17–31.

[8] F. Nafz, H. Seebach, J.-P. Steghöfer, G. Anders, and W. Reif, *Constraining Self-organisation Through Corridors of Correct Behaviour: The Restore Invariant Approach*. Basel: Springer Basel, 2011, pp. 79–93.

[9] J. Sudeikat, J. Steghöfer, H. Seebach, W. Reif, W. Renz, T. Preisler, and P. Salchow, "Design and simulation of a wave-like self-organization strategy for resource-flow systems," in *Proceedings of The Multi-Agent Logics, Languages, and Organisations Federated Workshops (MALLOW 2010)*, vol. 627, 2010.

[10] G. Anders, H. Seebach, F. Nafz, J. Steghöfer, and W. Reif, "Decentralized reconfiguration for self-organizing resource-flow systems based on local knowledge," in *2011 Eighth IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems*, 2011, pp. 20–31.

[11] J. Sudeikat, J.-P. Steghöfer, H. Seebach, W. Reif, W. Renz, T. Preisler, and P. Salchow, "On the combination of top-down and bottom-up methodologies for the design of coordination mechanisms in self-organising systems," *Information and Software Technology*, vol. 54, no. 6, pp. 593 – 607, 2012.

[12] F. Fioretto, E. Pontelli, and W. Yeoh, "Distributed constraint optimization problems and applications: A survey," *Journal of Artificial Intelligence Research*, vol. 61, pp. 623–698, 2018.

[13] A. Petcu and B. Faltings, "A scalable method for multiagent constraint optimization," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, ser. IJCAI'05, 2005, pp. 266–271.

[14] T. Laut, B. Ottens, and R. Szymanek, "Frodo 2.0: An open-source framework for distributed constraint optimization," *Proceedings of the IJCAI'09 Distributed Constraint Reasoning Workshop (DCR'09)*, pp. 160–164, 2009.

[15] H. Seebach, F. Ortmeier, and W. Reif, "Design and construction of organic computing systems," in *2007 IEEE Congress on Evolutionary Computation*, Sep. 2007, pp. 4215–4221.

[16] H. Seebach, F. Nafz, J.-P. Steghöfer, and W. Reif, "How to design and implement self-organising resource-flow systems," in *Organic ComputingA Paradigm Shift for Complex Systems*. Springer, 2011, pp. 145–161.

[17] I. A. Chaudhry and A. A. Khan, "A research survey: review of flexible job shop scheduling techniques," *International Transactions in Operational Research*, vol. 23, no. 3, pp. 551–591, 2016.

[18] N. Musliu, "Min conflicts based heuristics for rotating workforce scheduling problem," in *The Sixth Metaheuristics International Conference*, 2005.

[19] G. Clair, E. Kaddoum, M. Gleizes, and G. Picard, "Self-regulation in self-organising multi-agent systems for adaptive and intelligent manufacturing control," in *2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, 2008, pp. 107–116.

[20] Thomas Leaute, Brammert Ottens, Radoslaw Szymanek, "Frodo: a framework for open/distributed optimization version 2.17.1 user manual," 2019.

[21] J. Hirsch, M. Neumayer, H. Ponsar, O. Kosak, and W. Reif, "Deadlock avoidance for multiple tasks in a self-organizing production cell," in *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, 2020, pp. 178–187.

[22] T. Schiex, H. Fargier, G. Verfaillie *et al.*, "Valued constraint satisfaction problems: Hard and easy problems," *IJCAI (1)*, vol. 95, pp. 631–639, 1995.

[23] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, "Minizinc: Towards a standard cp modelling language," in *Principles and Practice of Constraint Programming – CP 2007*, C. Bessière, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 529–543.

[24] B. Porter, R. R. Filho, and P. Dean, "A survey of methodology in self-adaptive systems research," in *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, 2020, pp. 168–177.