A decentralized multi-agent algorithm for the set partitioning problem

Gerrit Anders, Florian Siefert, Jan-Philipp Steghöfer, Wolfgang Reif

Angaben zur Veröffentlichung / Publication details:

Anders, Gerrit, Florian Siefert, Jan-Philipp Steghöfer, and Wolfgang Reif. 2012. "A decentralized multi-agent algorithm for the set partitioning problem." In *Principles and Practice of Multi-Agent Systems: 15th International Conference, PRIMA 2012, Kuching, Sarawak, Malaysia, September 3-7, 2012, proceedings*, edited by Iyad Rahwan, Wayne Wobcke, Sandip Sen, and Toshiharu Sugawara, 107–21. Berlin: Springer.
https://doi.org/10.1007/978-3-642-32729-2_8.

A Decentralized Multi-agent Algorithm for the Set Partitioning Problem

Gerrit Anders, Florian Siefert, Jan-Philipp Steghöfer, and Wolfgang Reif

Institute for Software & Systems Engineering, Augsburg University, Germany
{anders,siefert,steghoefer,reif}@informatik.uni-augsburg.de

Abstract. A decentralized algorithm for solving set partitioning problems (SPP) has numerous applications in multi-agent systems. Apart from its relevance for problems of operations research, the SPP is equivalent to clustering of agents as well as the generation of coalition structures. SPADA, the algorithm proposed in this paper, does not use central metrics, relies exclusively on local agent knowledge, and respects agent autonomy. By trying to increase its own benefit, each subset of agents, representing a coalition or cluster, contributes to establishing a suitable partitioning. Agents are at liberty to decide if they want to join or leave a subset. All operations that change the composition of these subsets or the acquaintances of agents can be mapped onto a graph that represents these relations. The algorithm is evaluated in a scenario of decentralized energy management where it is used as a self-organization mechanism. The evaluations show that the quality of the partitioning is within 10% of the solutions found by a particle swarm optimizer with global knowledge.

Keywords: Self-Organization, Distributed Problem Solving, Set Partitioning Problem, Coalition Formation, Clustering.

1 Introduction

In the set partitioning problem (SPP), a set $A = \{a_1, \ldots, a_n\}$ is partitioned into $k \leq n$ subsets, called *partitions*, such that each partition is pairwise disjoint from all others. The resulting set of partitions is called a *partitioning*. If a SPP is complemented with a similarity metric and the set is partitioned in a way that similar or dissimilar elements are grouped in the same partition, the SPP is equivalent to *clustering* or *anticlustering* [14], respectively. If the a_i represent agents and the metric encodes how well different agents can work together to solve a problem, the SPP is equivalent to *coalition structure generation* [12,13].

Due to these interrelations, algorithms for the solution of the SPP in multi-agent systems (MAS) have a broad area of application. Some of these algorithms, e.g., those formulated and solved as a linear programming problem, require global system knowledge and are thus not applicable if the cost of gathering this knowledge is high or the system does not support obtaining such knowledge due to its openness and adaptivity. Many existing approaches try to circumvent this problem one way or the other, but they still suffer from some drawbacks

that limit their usefulness in highly adaptive systems. In many cases, a global metric is required or some form of global knowledge needs to be gathered before the algorithm can be run (e.g., [12]). A crucial step in some algorithms (e.g., those proposed in [13]) is to distribute this global knowledge among the agents. All possible coalitions and their utility are pre-calculated and the best one is picked after a global announcement. This ensures optimal solutions but introduces global synchronization points. This is not practical in many cases, especially when agents enter and leave the system arbitrarily.

The decentralized algorithm proposed in this paper – the **S**et **P**artitioning **A**lgorithm for **D**istributed **A**gents (SPADA) – does not suffer from these drawbacks. It solves SPPs in a completely decentralized fashion, relying only on local knowledge. Thus, no central metric is necessary and the quality of the partitioning is evaluated locally. With different metrics, SPADA can easily be applied to different clustering and coalition structure generation problems in MAS. For example, SPADA can be used to perform anticlustering [14] in which a set is partitioned in a way that the resulting clusters are similar but the elements within each cluster are dissimilar. A further possibility is to establish a clustering in which each cluster must have properties similar to the original set. In Sect. 4, we show an application in the field of decentralized energy management, where anticlusters form a system structure at runtime. The case study also shows that our algorithm is well-suited for reconfiguration in self-organizing systems.

The paper is structured as follows: Sect. 2 introduces a graph-based view of partitions. The decentralized algorithm, SPADA, is detailed in Sect. 3. Subsequently, Sect. 4 describes a scenario in which SPADA was evaluated and presents the evaluation results. In Sect. 5, SPADA is compared to related approaches. Finally, Sect. 6 summarizes the paper and gives an outlook on future work.

2 The Agent and System Model

Before we explain SPADA in detail, we state the assumptions we make about the MAS on which the algorithm is implemented and introduce the model, a graph-based view on the system, used to specify agent knowledge and partitions.

We assume a MAS in which all agents are able to communicate with each other. The knowledge of agents, however, is limited so that each agent is acquainted with a constant number of other agents. This property restricts the number of agents an interaction can be initiated with. Although acquaintances may change over time, the number of acquaintances per agent is constant to minimize the memory required and reduce the probability of working with outdated information. Acquaintances are represented by directed and unique *links*. A link (a, b) indicates that agent a is acquainted with agent b. Furthermore, the acquaintance relation is (1) irreflexive as an agent is not acquainted with itself, (2) not symmetric as links are directed, and (3) not transitive to limit the number of acquaintances per agent. As a result, a system of agents and links between them forms an overlay network that can be represented as a directed *acquaintances graph* (see Fig. 1(a)). We assume that the undirected counterpart

of the acquaintances graph is fully connected. This is important because partitions located in different subgraphs cannot exchange knowledge and agents. Thus, disconnected subgraphs would reduce the set of possible solutions.

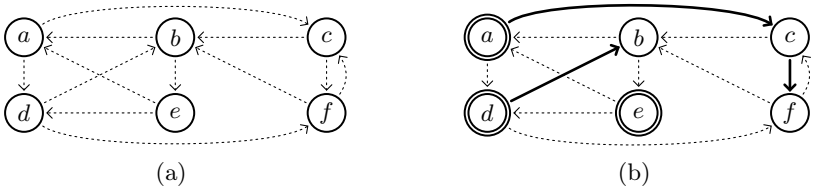


Fig. 1. Fig. 1(a) shows a system of six agents represented as a graph. Agents are nodes and acquaintances are depicted by directed edges, e.g., f is acquainted with b and c. Fig. 1(b) depicts a system consisting of three partitions $\{a, c, f\}, \{b, d\}, \{e\}$ with leaders a, d, e. Dashed edges represent unmarked links, whereas bold edges represent marked links.

A partitioning for such a MAS is a division of the given graph into several subgraphs with a pairwise disjoint set of nodes (i.e., agents). Partitions are represented by *marked links*, which are links with a partition-specific flag. Links without this flag are called *unmarked links*. A marked link (a, b) between two agents a, b states that a is acquainted with b, and that a, b are members of the same partition. Note that two agents c, d can be in the same partition without a link between them. As links are unique, a link is either marked or unmarked.

In terms of the acquaintances graph, a partition is defined by a tree T of marked links with a designated agent (the *leader* λ) at its root. This results in a directed forest for all partitions. With x an arbitrary agent and $\mathcal{R}^*$ the reflexive transitive closure of the relation $\mathcal{R}$ induced by marked links, $\{x \mid \lambda\, \mathcal{R}^*\, x\}$ describes the set of members of λ's partition. Fig. 1(b) shows three partitions $\{a, c, f\}$, $\{b, d\}$, and $\{e\}$ with leaders a, d, and e at the trees' roots.

Each leader λ steers the development of its partition, and is known to every member of its partition[1]. Initially, every agent is the leader of a partition with size one. Though every agent is basically capable of being a leader, there is only one leader per partition to avoid inconsistencies in the course of the formation process. A leader therefore unambiguously identifies its partition. It is responsible for managing the partition's composition and knowledge by modifying the marked and unmarked links of its members. For this purpose, a leader knows all marked links of its partition (the members) and all unmarked links of all members (further acquaintances of the members). With respect to Fig. 1(b), leader d, e.g., has knowledge of its own and b's marked links $\{(d, b)\} \cup \emptyset$ and unmarked links $\{(d, f)\} \cup \{(b, a), (b, e)\}$. A leader λ changes its partition Λ by requesting Λ's acquaintances (i.e., the acquaintances of Λ's members) to join or by asking agents within Λ to leave in order to increase the benefit of the individual partitions or the system. Because of the constant number of links per agent, the

[1] This does not imply that member x is acquainted with λ via an unmarked link (x, λ).

number of external agents a leader can contact to extend its partition is limited by the number of partition members multiplied by the number of acquaintances per agent, minus the number of partition members.

3 The Decentralized Set Partitioning Algorithm

As mentioned in the previous section, each leader tries to optimize the composition of its partition with regard to various application-specific criteria. Therefore, each leader periodically evaluates whether it is worthwhile 1) to integrate new agents into its partition and 2) to exclude current members from its partition. Leaders must thus be able to identify appropriate candidates for integration (see Sect. 3.1) and exclusion (see Sect. 3.5). Because agents that were requested to join another partition are at liberty to decline the invitation (see Sect. 3.2 and Sect. 3.3), they must be able to make an informed decision whether to accept or reject such an invitation. If an agent accepts an invitation, it is removed from its current partition (see Sect. 3.4). In general, leaders and requested agents make such a decision with the help of a local fitness function $f(\Lambda, \epsilon)$ on the basis of local knowledge. $f(\Lambda, \epsilon)$ assesses the quality of a partition Λ in its local environment ϵ that is defined as a set of agents (the higher the fitness, the better the partition). This set consists of the acquainted agents of Λ that are not members of Λ. A leader uses the fitness function as a basis to evaluate the reward functions $r_r(a, \Lambda, \epsilon)$ (r stands for *request*) and $r_e(a, \Lambda, \epsilon)$ (e stands for *exclude*) to determine appropriate candidates a for integration and exclusion. Further, a requested agent a makes its decision whether to change from partition Λ_{cur} to partition Λ_{req} based on the fitness function by applying a reward function $r_j(a, \Lambda_{req}, \Lambda_{cur}, \epsilon)$ (j stands for *join*). While ϵ depends on the situation in which such a reward function is evaluated, each agent must use the same fitness and reward functions in order that the agents pursue a common objective to solve the SPP. For example, if agents with similar computational power should be grouped into a partition, each agent should use $f(\Lambda, \epsilon)$ to compare the similarity of the computational power in a specific partition Λ to the corresponding value in partitions defined by the agents contained in the local environment ϵ.

Since agents use local knowledge to evaluate the fitness, they make assumptions about their environment based on their own knowledge and application-specific information retrieved from acquainted agents. To make better use of this information, leaders reorganize the acquaintances of agents being part of their partitions so that agents and partitions become acquainted with several different agents over time (see Sect. 3.6). As a result, local knowledge spreads in the system, resulting in a larger variety of possible partitions. Moreover, as integrated agents introduce access to new acquaintances in the form of unmarked links, the partition's knowledge and flexibility is increased with each new member.

In the following sections, we describe the procedure for integrating and removing agents in detail, and show how SPADA deals with agents entering and leaving the system (see Sect. 3.7). We further explain that leaders stop modifying their partition as soon as it satisfies given termination criteria (see Sect. 3.8).

3.1 Identifying Candidates for Integration

Each leader λ identifies candidates for integration into its partition Λ and determines the order in which they are invited to join Λ as shown in the following.

Let $\mathcal{A}_r$ be the set of all acquainted agents of Λ's members that, in principle, can be integrated into Λ (i.e., $\mathcal{A}_r$ does not contain members of Λ). To identify suitable candidates, λ assesses the benefit of their participation in Λ for at most $m_r \in \mathbb{N}$ agents. The number of agents to be assessed is limited for efficiency reasons, but the quality of the result increases with m_r. λ defines which agents are to be rated by determining a largest arbitrary subset $\mathcal{A}_r^* \subseteq \mathcal{A}_r$ with $|\mathcal{A}_r^*| \leq m_r$ elements. For each potential member $a \in \mathcal{A}_r^*$, λ rates the reward $r_r^a \in \mathbb{R}$ if agent a changes into Λ by using a reward function $r_r(a, \Lambda, \epsilon)$ that relies on the fitness function. The reward function compares the fitness $f(\Lambda, \epsilon)$ of Λ's current composition to the fitness $f(\Lambda \cup \{a\}, \epsilon \setminus \{a\})$ when a is a member of Λ:

$$r_r^a = r_r(a, \Lambda, \epsilon) = f(\Lambda \cup \{a\}, \epsilon \setminus \{a\}) - f(\Lambda, \epsilon)$$

For this evaluation, λ requests local knowledge about Λ and its environment ϵ from its partition members and from agents its members are acquainted with. A reward $r_r^a < 0$ states that it is not beneficial to integrate a into the partition since the fitness would decrease. If r_r^a is greater than a given threshold $\tau_r \in \mathbb{R}$, a is a candidate for integration. In such a case, λ adds (a, r_r^a) to a list $\mathcal{L}_r$ that holds rated candidates. Afterwards, λ sorts $\mathcal{L}_r$ by reward in descending order.

Since τ_r may be less than 0, λ's partition can be extended by an agent a (and thus become acquainted with other agents), although a is not considered beneficial from the partition's perspective. This can be of use in some situations, e.g., if the partition's fitness is low and there are no alternatives, or if the partition has too little knowledge for appropriate evaluations of the local fitness function.

Next, we show how λ integrates some of the candidates in $\mathcal{L}_r$ into Λ.

3.2 Trying to Integrate Agents into a Partition

In case $\mathcal{L}_r$ is not empty, a leader λ assumes that it is beneficial to extend its partition Λ by at most $k_r \in \mathbb{N}$ candidates given in the list $\mathcal{L}_r$. Until Λ has been extended by k_r agents or $\mathcal{L}_r$ is empty, λ removes the first entry (a, r_r^a) from $\mathcal{L}_r$ and sends an invitation to agent a. This invitation includes application-specific information about Λ and its acquainted agents. Based on this information and the information about a's current partition, a decides whether or not it accepts the request (see Sect. 3.3). If a accepts the invitation, it sends an acknowledgment to λ, whereupon λ asks a to leave its current partition. After a was removed from its current partition (see Sect. 3.4), it provides λ with all its acquaintances (unmarked links) and adopts λ as its leader.

λ integrates a into Λ by converting an arbitrary unmarked link (b, a) pointing from member b to a into a marked link by adding a flag. Hence, a is a leaf with parent b in the tree representing Λ. The number of links per agent remains unchanged as the unmarked link (b, a) is converted into a marked link and removing a from its prior partition does not change the number of links per agent.

3.3 Handling Join Partition Requests

Whenever an agent a receives a request to join the partition Λ_{req} of a requesting leader λ_{req}, a decides whether to accept or refuse the invitation based on local knowledge. To come to a decision, a uses the fitness function to compute the reward $r_j^{\Lambda_{req}}$ of changing from its current partition Λ_{cur} to partition Λ_{req}:

$$r_j^{\Lambda_{req}} = r_j(a, \Lambda_{req}, \Lambda_{cur}, \epsilon) = f(\Lambda_{req} \cup \{a\}, \epsilon \setminus \{a\}) - f(\Lambda_{req}, \epsilon) + f(\Lambda_{cur} \setminus \{a\}, \epsilon) - f(\Lambda_{cur}, \epsilon \setminus \{a\})$$

The knowledge of a used to evaluate the reward is information sent by λ_{req} about Λ_{req} and Λ_{req}'s acquaintances as well as knowledge requested from its current leader λ_{cur} about Λ_{cur} and Λ_{cur}'s acquaintances. This additional estimation of the reward by a is advantageous because λ_{req} assessed the use of a based on its own local knowledge which differs from the knowledge available to a. In particular, since a's view on the local environment ϵ is extended by Λ_{req}'s knowledge, a can come to a different decision and refuse the invitation.

In case a accepts the invitation, λ_{req} asks a to leave Λ_{cur}. If a is not the leader λ_{cur}, it informs λ_{cur} of its intention to leave the partition. Thereupon, λ_{cur} removes a from Λ_{cur} (see Sect. 3.4). Otherwise, if a is the leader of Λ_{cur}, two cases have to be distinguished: 1) In case a is not the only member of Λ_{cur}, a removes itself from the partition (see Sect. 3.4). 2) In case a is the only member of Λ_{cur}, Λ_{cur} dissolves after a has been integrated into Λ_{req}.

Having been removed from Λ_{cur}, a sends its acquaintances in the form of unmarked links to λ_{cur}, thus completing the integration process.

3.4 Removing an Agent from a Partition

Removing an agent a from a partition Λ with leader λ modifies Λ's acquaintances graph. Because λ separates a from the tree T that represents Λ, the number of modifications needed depends on the structure of the acquaintances graph and the position of a in T. a is removed from T by converting all incoming and outgoing marked links of a into unmarked links by deleting the flag. Afterwards, λ re-establishes the tree property. The number of links n per agent is not changed. As any agent can be removed from a partition, a can be a common node, a leaf, or the root of T. The latter is the case if $\lambda = a$. In such a situation, a removes itself from Λ, determines a new leader, and sends all knowledge about Λ to the new leader. We describe the necessary operations for all cases in the following.

With the tree T in mind, let C_a be the set of children of a, p_a the parent of a if a is not the root, and λ^* the leader of Λ after a was removed. Furthermore, let $\mathcal{L}_{rem}$ and $\mathcal{L}_{new}$ be two preliminary empty sets of links. $\mathcal{L}_{rem}$ gathers all marked links to be removed from T, and $\mathcal{L}_{new}$ all marked links to be added to T.

First, λ adds the marked link (p_a, a) to $\mathcal{L}_{rem}$ if a is not T's root (see Fig. 2(a)). In such a case, λ remains the root of T ($\lambda^* = \lambda$). Then, if a has been a leaf, no further marked links have to be identified for removal and λ updates the sets of links. Otherwise, if a has not been a leaf ($C_a \neq \emptyset$), λ converts all marked

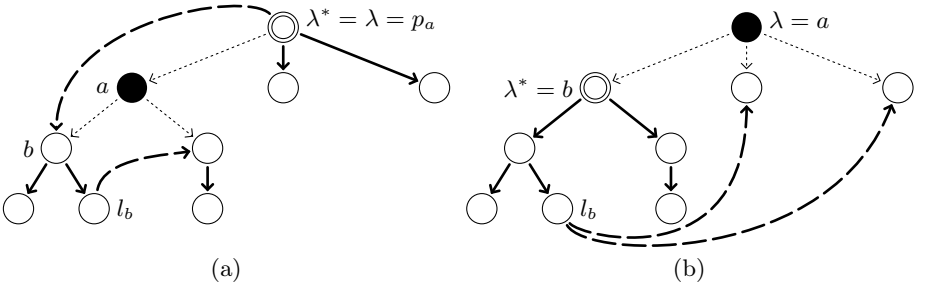


Fig. 2. Fig. 2(a) depicts the removal of agent a from a partition with leader λ (bold edges represent marked links): existing marked links from/to a become unmarked links (dashed non-bold edges) and new marked links (dashed bold edges) to a's children are created. Fig. 2(b) shows the removal of an agent a that is the leader λ of a partition: a's child b becomes the new leader λ^*, existing marked links starting at a become unmarked links, and new marked links from l_b to all of a's children except b are created.

links (a, h) with $h \in C_a$ to unmarked links and updates a's knowledge accordingly. Afterwards, λ selects an arbitrary agent $b \in C_a$. If a is not the root of T, λ creates a new marked link (p_a, b) to keep the tree connected and adds this link to $\mathcal{L}_{new}$. Otherwise, b becomes the new root, and thus the new leader $\lambda^* = b$ (see Fig. 2(b)). Subsequently, λ selects a leaf l_b of the subtree with root b (note that this could be b itself and that all n outgoing links of a leaf are unmarked) and connects l_b with each child $i \in C_a \setminus \{b\}$ ($|C_a \setminus \{b\}| < n$) so that i remains in the partition. Therefor, λ establishes a new marked link (l_b, i), and adds (l_b, i) to $\mathcal{L}_{new}$. l_b thus becomes the new parent of each such agent i.

Next, λ updates the set of marked links $\mathcal{M}$ of Λ's members by adding all links contained in $\mathcal{L}_{new}$ and removing all links contained in $\mathcal{L}_{rem}$. Having updated $\mathcal{M}$, λ adds all links contained in $\mathcal{L}_{rem}$ to the set of unmarked links $\mathcal{U}$ of Λ's members and removes all links in $\mathcal{L}_{new}$ from $\mathcal{U}$. For p_a and l_b, these modifications can lead to a situation in which the number of outgoing links exceeds n by k. In such a case, λ removes k unmarked links for the corresponding agent. This is possible as (1) k is at most 1 for p_a and p_a has at least one unmarked link (p_a, a), and (2) l_b had n unmarked links before a has been removed. However, this procedure can break the acquaintances graph into several disconnected subgraphs (see Sect. 2 and 4). Since a is removed from Λ, λ's knowledge of links starting at a is obsolete.

Having updated the links, λ informs a that it was removed from Λ. In case the leader removed itself ($\lambda^* \neq \lambda$), λ transfers its knowledge to λ^*. Afterwards, it informs each member to adopt λ^* as leader.

3.5 Excluding Agents from a Partition

Excluding agents is another way to increase the fitness of a partition with two or more members. This can be useful in situations in which a partition or a partition member changed over time, resulting in a structure where some of its members, initially beneficial, decrease the partition's fitness.

A leader λ identifies members to be excluded from its partition Λ similar to identifying candidates for integration (see Sect. 3.1) by determining a largest arbitrary subset $\mathcal{A}_e$ of Λ's members ($|\mathcal{A}_e| \leq m_e$) that should be analyzed with respect to the benefit of their exclusion. λ evaluates the reward r_e^a of excluding agent $a \in \mathcal{A}_e$ by subtracting the fitness $f(\Lambda \setminus \{a\}, \epsilon \cup \{a\})$ in a situation in which a is not a member of Λ from Λ's current fitness $f(\Lambda, \epsilon)$. Again, λ evaluates the fitness by making use of local knowledge requested from Λ's members and their acquaintances. A reward $r_e^a \leq 0$ means that it is not beneficial that a is a member of Λ. If r_e^a is below a threshold τ_e, λ adds the tuple (a, r_e^a) to a list $\mathcal{L}_e$ that holds rated candidates for exclusion sorted by reward in ascending order.

Next, those candidates whose exclusion is most beneficial are removed. Until Λ has been reduced by $k_e < |\Lambda|$ agents or $\mathcal{L}_e$ is empty, λ removes the first entry (a, r_e^a) from $\mathcal{L}_e$ and removes a from Λ as described in Sect. 3.4. Before removing another agent, λ informs a about its exclusion which then forms a new partition of size 1 with itself as leader. If a was Λ's leader, no further agents are excluded.

3.6 Mixing Acquaintances

Having enlarged their partition, leaders mix the acquaintances of their partition members to enable variety in partition structures. Thus, each agent becomes acquainted with various different agents over time. Since agents keep their acquaintances whenever they change partitions, a broad spectrum of agents becomes available as potential members. Mixing acquaintances only modifies the heads of unmarked links. This entails that the direction of links does not change, and that marked links and thus the tree structure are not altered. For example, two unmarked links (a, c) and (b, d) can therefore be replaced by (a, d) and (b, c).

SPADA's implementation makes sure that the number of acquaintances of the partition as well as of each agent is kept constant and that the changes adhere to the properties of links named in Sect. 2. Consequently, mixing acquaintances does not break the acquaintances graph into disconnected subgraphs (see Sect. 2).

3.7 Dealing with the Dynamic Nature of MAS

To accommodate the dynamics of MAS, SPADA deals with agents joining or leaving the system so that the constant number of links per agent as well as the tree property is maintained. We currently assume that messages are processed correctly and malfunctioning agents become unavailable. If a new agent a joins the system, it broadcasts a message and creates unmarked links to a random subset of agents that reply to this message. Further, some agents replace one of their unmarked links by a new unmarked link to a. To identify unavailable agents, an agent periodically pings its acquaintances and its leader. If an agent detects that the head of an unmarked link is unavailable, it broadcasts a message and creates a new unmarked link to a responding agent. Moreover, if an agent notices that a head of its marked links is unavailable, it informs its leader which removes the unavailable agent from its partition. In case a leader left the system, each member of the corresponding partition assumes that it is the leader of a

new partition and sends a message to the heads of its marked links. Agents that receive such a message adopt the sender as leader and forward the message to the heads of their marked links, thus recreating a tree structure.

3.8 Termination and Completeness

Initially, every agent is the leader of its own partition (see Sect. 2)[2]. This initial structure changes over time until an adequate solution is found. To decide whether its partition contributes to an adequate solution, each leader evaluates application-specific termination criteria formulated as predicates. Since we regard a decentralized algorithm for MAS, leaders perform this evaluation on the basis of local knowledge provided by partition members and acquainted agents. If the termination criteria are met for a specific partition, its leader marks it as terminated. Consequently, there may be partitions marked as terminated while others keep on changing their structure. As the shape of partitions influence the satisfaction of termination criteria, the local fitness function used by agents and leaders should be devised in such a way that their decisions target termination.

If a member of a terminated partition Λ_{cur} receives and accepts a join partition request from an active partition Λ_{req}, the algorithm is reactivated for Λ_{cur}. Thus, partitions are reactivated individually which enables selective changes with respect to their composition. This is particularly beneficial when using the algorithm for reconfiguration in self-organizing systems (see Sect. 4).

Because of these properties, it is not certain that all partitions reach a consensus and thus termination cannot be guaranteed. Furthermore, just like every algorithm using only local knowledge, SPADA is not complete since there can be a global solution satisfying the termination criteria that cannot be found locally.

4 Analysis and Evaluation

For evaluation, we regarded an application from the domain of decentralized energy management, which benefits from decentralized, locally operating mechanisms like SPADA. In the power grid, the main task is to hold the balance between energy production and consumption. However, this task gets more and more complicated because the number of power plants, especially distributed energy resources like solar panels, is steadily increasing. An approach to deal with this rising complexity is to partition all power plants into self-organizing groups of power plants, called *Autonomous Virtual Power Plants (AVPPs)* [5]. Each AVPP autonomously plans the energy supply based on predictions made by producers and consumers, reacts to load or supply changes, and adapts its structure when required. Additionally, uncertainty – introduced by stochastic energy sources like wind and sun – is a great challenge since weather-dependent power plants suffer from limited controllability and hard to predict power output. As a way to handle uncertainty, AVPPs utilize the trustworthiness of power plants.

[2] Any structure adhering to the SPP's definition can be used as an initial partition.

A power plant's trust value increases with the accuracy of its power predictions and its availability. Besides other factors, these trust values play a key role in the formation of AVPPs. The aim is to achieve a similar mix of trustworthy and untrustworthy power plants in each AVPP by performing an anticlustering.

To assess the fitness of a partitioning, for each AVPP, we calculate the mean trust value of power plants contained in this AVPP. Since the goal is to equalize these mean trust values, their standard deviation σ is to be minimized. Because our trust values are from the interval $[0, 1]$, σ is always between 0 and $\sqrt{0.5}$. We therefore define a partitioning's fitness $\mathcal{F}(\sigma)$ as a function of σ as follows:

$$\mathcal{F}(\sigma) = \left(\frac{1}{\sigma + 1} - a\right) \cdot \frac{1}{1 - a}, \text{ with } a = \frac{1}{\sqrt{0.5} + 1}$$

$\mathcal{F}(\sigma)$ monotonically decreases on the interval $\left[0, \sqrt{0.5}\right]$ so that we have $\mathcal{F}(0) = 1$ for optimal partitionings (each partition has the same mean trust value) and $\mathcal{F}\left(\sqrt{0.5}\right) = 0$ for a maximum standard deviation. Expecting SPADA to perform well, $\mathcal{F}(\sigma)$ is particularly sensitive to changes when σ is small.

We compare the results of SPADA to a central approach that also solves the SPP in general, but by making use of global knowledge. More precisely, we use a particle swarm optimizer (PSO) [9], a metaheuristic for finding solutions in optimization problems, based on the flocking behavior of birds. Basically, the PSO uses a swarm of particles that roams the search space by modifying candidate solutions by basic set operations ("split" and "merge") in order to find optimal solutions with respect to $\mathcal{F}(\sigma)$. In contrast to other existing mechanisms like k-means clustering or the k-nearest neighbor algorithm, SPADA and the PSO do not specify a concrete number of partitions or clusters in advance.

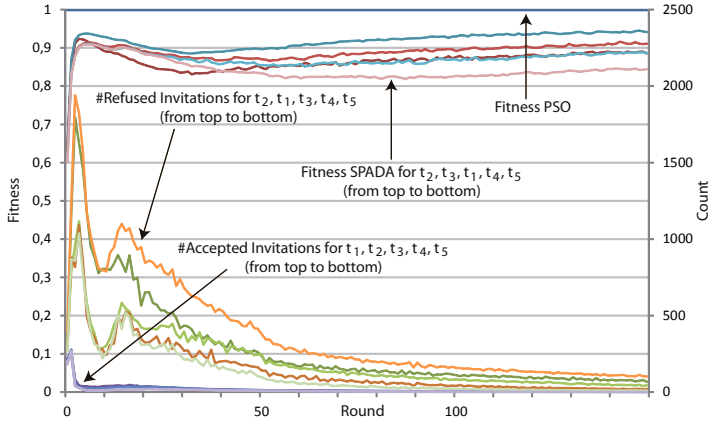
For evaluation, we implemented a system consisting of 435 agents in Repast Simphony [1], which uses a sequential, round-based execution model. Since we avoided the complexity of an asynchronous execution model, SPADA could not benefit from parallelism. In each round, every leader could modify the composition of its partition, mix acquaintances, and evaluate termination criteria, which could result in multiple changes to the partitioning (see Sect. 3). We assume that all messages are processed correctly and all agents work properly. SPADA's local fitness function $f(\Lambda, \epsilon)$ compared the mean trust value of a partition to the mean trust value of the partition's local environment. If $f(\Lambda, \epsilon) \leq 0.4$, a leader marked its partition as terminated. The higher the deviation between the mean trust value of the regarded partition and the local environment, the lower the partition's fitness. As mentioned in Sect. 3, a partition's local environment consists of the partition's acquaintances that are not members of the partition. An agent's decisions thus aimed at minimizing this deviation, thereby improving $\mathcal{F}(\sigma)$. To avoid fluctuation of agents between partitions, the reward function $r_j^{\Lambda_{req}}$ of a requested agent additionally implemented a mechanism that allows the agent to prefer partitions in which it has been a long time if the reward is small. Note that it is important to distinguish between an agent's local assessment of a partition's fitness by applying $f(\Lambda, \epsilon)$, and the global fitness of the partitioning gauged by $\mathcal{F}(\sigma)$ appraising the result of solving the SPP. The PSO used five particles to

find a nearly optimal partitioning within 30 seconds. We recorded 300 simulation runs for each test (see Fig. 3 for parameters). For each run, we generated the initial acquaintances graph at random. Further, each agent had a fix uniformly distributed random trust value $\in [0, 1]$. By parametrization, we prevented both algorithms from forming the trivial optimal partitioning consisting of one big AVPP ($\sigma = 0$). The following results are arithmetic means of recorded data (similar results are obtained with a beta distribution with $\alpha = \beta = 0.1$).

In the first simulation runs for t_1, ..., t_5, both algorithms modified an initial partitioning in which each agent formed its own partition to identify suitable parameters for SPADA. The results are depicted in Fig. 3. The PSO achieves a nearly optimal mean fitness of 0.999, and SPADA also performs very well: the mean fitness increases rapidly to a value beyond 0.9 (however, please note that the partitioning is changed several hundred times in each round), and slowly converges to mean values between 0.856 and 0.959. For each test, the graphs show similar characteristics: the number of accepted and refused invitations drops rapidly because partitions terminate over time[3]. Some partitions are reactivated by active partitions, leading to a temporal decline in fitness. However, since memberships are rearranged, SPADA achieves a higher fitness value for suitable parameter settings afterwards (t_2 and t_3). Looking at the results in detail, for t_1, the fitness is rather low and the number of refused invitations high, which is not desired as this means unnecessary processing and message delivery in the system. To increase local knowledge and thus the quality of the result, we increased the number of links n per agent in t_2. However, despite better fitness, the number of refused invitations even increases slightly. We thereupon used only half the number of links to be rated by a leader to limit the number of refused invitations per leader and round to $m_r = 10$ instead of $m_r = 20$; we used $k_r = 5$ for all simulation runs. Compared to t_2, the fitness is still high but refused invitations are reduced by nearly 50%. Next, we varied the minimum reward τ_r necessary to become a candidate for integration. In t_4, we increased τ_r from -2 to 0 so that leaders do not send invitations to potentially non-beneficial agents. Compared to t_3, the number of refused invitations is reduced but, as $\tau_r = 0$ reduces the spread of local knowledge, fitness decreases. Regarding t_5, where τ_r is 1, the number of refused invitations is further decreased. The fitness for t_5 is low because, with respect to the composition of partitions, variety is too limited: sometimes, leaders could not find candidates for integration.

In addition to the tests for initial configuration, we evaluated the behavior of SPADA for parameter sets t_3, t_4, and t_5 if it is used for reconfiguration, meaning that SPADA and the PSO were initialized with a randomly generated partitioning consisting of 10 to 20 partitions of a size between 5 and 50, and a rather high mean fitness of 0.875 as it might be the case in reconfiguration scenarios. Further, one random partition triggered reconfiguration while others were terminated. The results for t_3, t_4, and t_5 are similar to initial configuration

[3] Regarding a partition Λ, the number of messages necessary to identify suitable candidates for integration is $O(n \cdot |\Lambda| + m_r)$ (see Sect. 3.1), for exclusion $O(n \cdot |\Lambda| + m_e)$ (see Sect. 3.5), and to handle join partition requests $O(n \cdot |\Lambda|)$ (see Sect. 3.3).



Test	n	m_r	τ_r
t_1	10	20	−2
t_2	20	20	−2
t_3	20	10	−2
t_4	20	10	0
t_5	20	10	1

Fig. 3. For evaluation, we varied the number of links n per agent, the number of links m_r rated by leaders to identify candidates for integration, and the minimum reward τ_r necessary to become a candidate. The graph shows SPADA's average performance for tests t_1, t_2, t_3, t_4, and t_5 compared to the PSO when modifying an initial partitioning.

(see Fig. 3). Again, SPADA and the PSO achieve a high fitness of 0.941 and 0.997, respectively. The number of accepted and refused invitations, however, are less than a tenth of the number of accepted and refused invitations when regarding initial configuration. However, we observed that SPADA sometimes could not complete reconfiguration for t_5 because of the high value of τ_r. In such a case, the leader could not identify a candidate for integration because its partition was situated in a local optimum from its perspective. Consequently, we identified t_3 as the best set of parameters. Although nearly all partitions were re-triggered over time, SPADA needed 119 changes on average for t_3 to reconfigure the system as most of the re-triggered partitions terminated after a few rounds, whereas it made 1858 modifications to the initial partitioning. Hence, SPADA is very well-suited for reconfiguration where selective changes are often desired.

In all simulation runs, the undirected analog of the acquaintances graph remained connected although its connectivity is not yet guaranteed (see Sect. 3.4). The reason for this is that it is rather unlikely to break the acquaintances graph into disconnected subgraphs if the number of links per agent is high enough. Summarizing, despite using local knowledge only, the quality of partitionings found by SPADA is within 10% of the solutions found by the centralized PSO.

5 Related Work and Discussion

As outlined in the introduction, clustering is equivalent to solving the SPP. MAS clustering algorithms can be used in a variety of circumstances. However, truly decentralized ones, like SPADA, are relatively rare.

Agents are often used to represent complex data that needs to be categorized. In [6], e.g., agents are used to model the behavior of costumers on e-commerce

websites. Clustering the agents allows to define a hierarchical categorization of these websites and to categorize customers into different profiles. If a user can be assigned to one of the clusters, the type of website the user is most comfortable with can be displayed. The clustering is performed with global knowledge.

In networks on chip, a domain-specific agent-based clustering approach can be used to map tasks to processing elements (PEs) [3]. Clustered PEs fulfill tasks together. If a centralized task scheduler cannot map a task to a cluster, a re-clustering takes place. During the process, PEs are requested by clusters and switch if they are idle or can be made available until the task can be mapped.

There are several approaches for decentralized clustering in sensor networks. In this domain, it is easier to design efficient communication protocols on high-level structures represented by clusters than on individual nodes [7]. Additionally, as the network's structure can change arbitrarily at runtime and robustness is of great concern, centralized approaches are not employable. Instead, highly decentralized algorithms such as ACE [7] or HEED [15] are used. However, there are simplifying factors when clustering in sensor networks. First, the communication range of each node defines the set of neighbors. It is thus not necessary to define neighborhood relationships. Second, the goal of the algorithms is usually to find a clustering that assigns each sensor node a cluster head within its communication radius and to allow all cluster heads to communicate with each other. In contrast, the goal of our approach, SPADA, is not predefined and can differ greatly depending on the definition of the reward functions.

Coalition formation algorithms or task allocation algorithms [8] solve the SPP (and thus the clustering problem) during the process of *coalition structure generation*. Specialized algorithms for this task are often not fully distributed as, e.g., global knowledge is required to calculate the search space [12]. Even if the search space (i.e., the set of all possible coalitions) is distributed [13], such an approach still requires a lot of communication, scales badly, and is not applicable in systems with a fluctuating agent population. In other cases, pre-defined organizational structures are exploited to guide the search for coalitions. This enables the use of local knowledge and neighborhood relations. While [2] is based on a hierarchical system structure, [4] requires a graph structure that defines input/output relations between agents.

Our approach does not suffer from many of these drawbacks. It is based on an adaptive neighborhood relation, works with local knowledge only and is fully distributed. This also distinguishes the algorithm from other approaches to partitioning where global knowledge is a prerequisite, e.g., from control theory [10]. Apart from a fully connected acquaintances graph, no organizational structure is required. While other approaches are often optimized to specific problems in specific domains (e.g., [15]) and thus might make better use of domain-specific knowledge, SPADA deals with the properties of MAS in a generic way and allows the solution of different problems such as clustering, anticlustering, and coalition formation by applying different metrics (i.e., reward functions).

SPADA has been inspired by a clustering algorithm proposed by Ogston et al. [10], but has been substantially extended and improved. The differences

between this algorithm and SPADA primarily arise form our design goals: (1) to strictly adhere to the principle of agent autonomy, (2) to use only local knowledge, (3) to clearly separate different concerns, and (4) to achieve low complexity with (5) great flexibility by solving the SPP in general. [11] uses the notion of attributes and objectives to describe the properties and goals of an agent, as well as their euclidean distance to encode the distance between agents and clusters. SPADA encapsulated this information in the reward functions used by partition leaders. While this might at first seem to contradict design goals (1) and (3), it is beneficial since it allows altering the behavior of SPADA by only changing the reward functions. As this increases flexibility tremendously, in contrast to [11], SPADA can be used for clustering, anticlustering, or coalition structure generation. Additionally, SPADA respects the agent's autonomy as a leader has to ask agents whether or not they will join its partition. This allows for the efficient use of local knowledge, as the requested agent might have information that is not available to the requesting leader. Finally, unlike [11], SPADA represents partitions as trees and leaders try to change their partition's structure in a guided instead of a random process. This can lead to better results because of a lower probability of breaking a good partitioning and to fewer changes necessary in order to come to a result.

Unfortunately, it is not easily possible to quantify the differences between our approach and the one proposed in [11]. The algorithm of Ogston et al. does not solve the SPP in general and their paper leaves enough room for interpretation to allow different actual implementations of the basic idea, making it hard to give a fair evaluation. Therefore, we focused on a comparison to the particle swarm optimizer. The results obtained are even more significant as they demonstrate the quality of SPADA compared to nearly optimal solutions of a centralized optimization approach that also solves the SPP in general.

6 Conclusion and Future Work

In this paper, we presented a fully decentralized algorithm, called SPADA, that solves the set partitioning problem in multi-agent systems, a problem equivalent to clustering or coalition structure generation. In contrast to other approaches (e.g., [12,2]), our algorithm neither relies on global knowledge nor on a central metric nor does it require a predefined organizational structure to form appropriate partitions. It compensates for the drawback of having less knowledge available by honoring the autonomy of agents and the different perspectives they have on the system. As a result, despite its restriction to local knowledge, the evaluations show that the algorithm performs very well compared to a central approach using global knowledge. Furthermore, the above-mentioned properties widen its range of application and are advantageous in systems consisting of a great number of adaptive agents located in a changing environment.

Future work includes identifying possibilities to ensure the connectivity of the acquaintances graph whenever agents change partitions. We will further investigate how SPADA can deal with synergy effects when modifying partitions, and

how it can be applied to form partially predefined hierarchical system structures. Forthcoming papers will introduce the particle swarm optimizer used for evaluation, and show a modularized approach for creating local fitness functions.

Acknowledgments. This research is partly sponsored by the research unit *OC-Trust* (FOR 1085) of the German Research Foundation.

References

1. Repast Simphony, `http://repast.sourceforge.net` (accessed June 1, 2012)
2. Abdallah, S., Lesser, V.: Organization-Based Cooperative Coalition Formation. In: Int. Conference on Intelligent Agent Technology, pp. 162–168 (2004)
3. Al Faruque, M.A., Krist, R., Henkel, J.: ADAM: run-time agent-based distributed application mapping for on-chip communication. In: Proc. of the 45th Annual Design Automation Conference, pp. 760–765. ACM (2008)
4. Anders, G., Seebach, H., Nafz, F., Steghöfer, J.-P., Reif, W.: Decentralized Reconfiguration for Self-Organizing Resource-Flow Systems Based on Local Knowledge. In: 8th IEEE Int. Conference and Workshops on Engineering of Autonomic and Autonomous Systems (EASe), pp. 20–31 (2011)
5. Anders, G., Siefert, F., Steghöfer, J.-P., Seebach, H., Nafz, F., Reif, W.: Structuring and Controlling Distributed Power Sources by Autonomous Virtual Power Plants. In: Proc. of the Power & Energy Student Summit, pp. 40–42 (October 2010)
6. Buccafurri, F., Rosaci, D., Sarnè, G.M.L., Ursino, D.: An Agent-Based Hierarchical Clustering Approach for E-commerce Environments. In: Bauknecht, K., Tjoa, A.M., Quirchmayr, G. (eds.) EC-Web 2002. LNCS, vol. 2455, pp. 109–118. Springer, Heidelberg (2002)
7. Chan, H., Perrig, A.: ACE: An Emergent Algorithm for Highly Uniform Cluster Formation. In: Karl, H., Wolisz, A., Willig, A. (eds.) EWSN 2004. LNCS, vol. 2920, pp. 154–171. Springer, Heidelberg (2004)
8. Gerkey, B., Matarić, M.: A formal analysis and taxonomy of task allocation in multi-robot systems. Int. Journal of Robotics Research 23(9), 939 (2004)
9. Kennedy, J., Eberhart, R.: Particle Swarm Optimization. In: Proc. of the IEEE Int. Conference on Neural Networks, vol. 4, pp. 1942–1948 (1995)
10. Motee, N., Sayyar-Rodsari, B.: Optimal partitioning in distributed model predictive control. In: Proc. of the 2003 American Control Conference, vol. 6, pp. 5300–5305 (June 2003)
11. Ogston, E., Overeinder, B., Steen, M.V., Brazier, F.: A Method for Decentralized Clustering in Large Multi-Agent Systems. In: Proc. of the 2nd Int. Joint Conference on Autonomous Agents and Multiagent Systems, pp. 789–796 (2003)
12. Rahwan, T., Ramchurn, S.D., Jennings, N.R., Giovannucci, A.: An Anytime Algorithm for Optimal Coalition Structure Generation. Journal of Artificial Intelligence Research 34, 521–567 (2009)
13. Shehory, O., Kraus, S.: Methods for Task Allocation via Agent Coalition Formation. Artificial Intelligence 101(1-2), 165–200 (1998)
14. Valev, V.: Set Partition Principles Revisited. In: Amin, A., Pudil, P., Dori, D. (eds.) SPR 1998 and SSPR 1998. LNCS, vol. 1451, pp. 875–881. Springer, Heidelberg (1998)
15. Younis, O., Fahmy, S.: HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks. IEEE Transactions on Mobile Computing 3, 366–379 (2004)