A particle swarm optimizer for solving the set partitioning problem in the presence of partitioning constraints

Gerrit Anders, Florian Siefert, Wolfgang Reif

Angaben zur Veröffentlichung / Publication details:

Anders, Gerrit, Florian Siefert, and Wolfgang Reif. 2015. "A particle swarm optimizer for solving the set partitioning problem in the presence of partitioning constraints." In Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART-2015), January 10-12, 2015, in Lisbon, Portugal, edited by Stephane Loiseau, Joaquim Filipe, Beatrice Duval, and Jaap van den Herik, 151–63. Setúbal: SciTePress. https://doi.org/10.5220/0005220501510163.

A Particle Swarm Optimizer for Solving the Set Partitioning Problem in the Presence of Partitioning Constraints

Gerrit Anders, Florian Siefert and Wolfgang Reif
Institute for Software & Systems Engineering, Augsburg University, Augsburg, Germany

Keywords: Set Partitioning Problem, Clustering, Particle Swarm Optimization, Evolutionary Computing.

Abstract: Solving the set partitioning problem (SPP) is at the heart of the formation of several organizational structures in multi-agent systems (MAS). In large-scale MAS, these structures can improve scalability and enable cooperation between agents with (different) limited resources and capabilities. In this paper, we present a discrete Particle Swarm Optimizer, i.e., a metaheuristic, that solves the NP-hard SPP in the context of partitioning constraints – which restrict the structure of valid partitionings in terms of acceptable ranges for the number and the size of partitions – in a general manner. It is applicable to a broad range of applications in which regional or global knowledge is available. For example, our algorithm can be used for coalition structure generation, strict partitioning clustering (with outliers), anticlustering, and, in combination with an additional control loop, even for the creation of hierarchical system structures. Our algorithm relies on basic set operations to come to a solution and, as our evaluation shows, finds high-quality solutions in different scenarios.

1 INTRODUCTION AND RELATED WORK

In numerous multi-agent systems (MAS), a crucial step is to establish an organizational structure that supports the agents' and system's objectives (Horling and Lesser, 2004). Among other things, these structures allow agents to benefit from the capabilities of others, thereby increasing their own value of participating in the system. In large-scale systems, organizations are also a way to deal with complexity and scalability issues, which is often accomplished by the formation of hierarchies (Steghöfer et al., 2013).

In many cases, these organizations are based on structures that can be described as a *partitioning*. In the *set partitioning problem* (SPP) (cf. (Chu and Beasley, 1998)), a set $\mathfrak{A} = \{a_1, \ldots, a_n\}$ of $n > 1$ agents a_i is partitioned into non-empty and pairwise disjoint subsets, called *partitions*, that together constitute a partitioning at minimal cost. Feasible, i.e., valid, partitions $\mathfrak{B} = \{b_1, \ldots, b_m\}$ are predefined and finding the optimal partitioning is NP-hard. In this paper, we assume that feasible partitions are only constrained in terms of a minimum s_{min} and maximum s_{max} size. This will often result in a very large number of feasible partitions. In the unbounded case, i.e., the *complete SPP*, there are $2^{|\mathfrak{A}|} - 1$ partitions and the size of the search space is given by the nth *Bell number* $\mathcal{B}_n$ (e.g., $\mathcal{B}_{50} \approx 1.86 \cdot 10^{47}$), which satisfies *Dobiński's formula* (Bender et al., 1999). Thus, the search space does not only grow exponentially with the number m of feasible partitions but also with the number n of agents. Even in a system in which the set of agents $\mathfrak{A}$ is not subject to change over time, it would not be suitable to pre-calculate all feasible partitions in advance (for $|\mathfrak{A}| = 50$, this needs more than one week on our Xeon machine). To differentiate this specific problem from the original SPP more clearly, we will refer to it as the *partitioning problem* (PP). In contrast to the SPP's original definition – in which the costs of having a partition b_j included are additive and predefined –, we allow a more flexible objective function in the PP: We only presume an application-specific metric that evaluates if a partitioning, i.e., a combination of partitions, is fit for purpose. If the metric specifies to group similar or dissimilar agents, the PP is equivalent to *strict partitioning clustering (with outliers*[1]*)* or *anticlustering*[2] (Valev, 1998), respectively. If the metric defines how well agents can work together on a common task, the PP is equivalent to *coalition structure generation* (cf. (Shehory and Kraus, 1998)).

If an algorithm solves the PP by representing each $a_i \in \mathfrak{A}$ by a vector $\boldsymbol{g}_i$ of those attributes of a_i that are relevant to solve the PP, it actually has to solve a *mul-*

[1] Supported by a separate partition that holds all outliers.
[2] In anticlustering, the resulting partitions are similar.

Anders G., Siefert F. and Reif W..
A Particle Swarm Optimizer for Solving the Set Partitioning Problem in the Presence of Partitioning Constraints.
DOI: 10.5220/0005220501510163
In *Proceedings of the International Conference on Agents and Artificial Intelligence* (ICAART-2015), pages 151-163
ISBN: 978-989-758-074-1

tiset partitioning problem (MPP) for the multiset $\mathfrak{G} = \wr \boldsymbol{g}_1, \ldots, \boldsymbol{g}_n \int$. That is because we might have $\boldsymbol{g}_i = \boldsymbol{g}_j$ for two agents $a_i \neq a_j$. In the MPP, the multiset sum $\uplus_{K \in \mathcal{P}} K$ of all partitions K (here, non-empty multisets) in the partitioning $\mathcal{P}$ must equal $\mathfrak{G}$. In this paper, we assume heterogeneous agents so that all vectors $\boldsymbol{g}_i$ are different (i.e., $\forall a_i, a_j \in \mathfrak{A} : a_i \neq a_j \rightarrow \boldsymbol{g}_i \neq \boldsymbol{g}_j$). Hence, $\mathfrak{G}$ is a set and the problem is reduced to a PP.

Algorithms for the solution of the PP in MAS have a broad area of application, e.g., in sensor networks, energy management systems, manufacturing systems, communication systems, or e-commerce: In (Younis and Fahmy, 2004), a highly decentralized algorithm is used to assign each sensor node a cluster head within its communication radius and to allow all cluster heads to communicate with each other. (Anders et al., 2012) present a decentralized graph-based algorithm, called SPADA, that allows power plants to self-organize into virtual power plants in order to lower the time needed to create power plant schedules. In (Anders et al., 2011), existing organizational structures are exploited in form of input/output relations between agents to guide a decentralized coalition formation that reconfigures a production cell. (Al Faruque et al., 2008) show an agent-based clustering approach for networks on chip that is used to map tasks to processing elements. In (Buccafurri et al., 2002), the costumers of e-commerce websites are categorized into different profiles on the basis of global knowledge.

Some algorithms that solve instances of the PP, e.g., those formulated and solved as a linear programming problem, require global system knowledge but yield optimal solutions (e.g., (Rahwan et al., 2009)). Because of the PP's complexity they are often designed as anytime algorithms or distribute this global knowledge, i.e., the search space, among the agents, which allows to calculate the utility of all possible partitions and pick the best one after a global announcement (e.g., (Shehory and Kraus, 1998)). Other approaches, such as (Anders et al., 2012) or (Ogston et al., 2003), rely on local knowledge and solve the PP in a completely decentralized fashion. While such strong self-organization approaches can deal with very large systems (Di Marzo Serugendo et al., 2005), the lack of regional or global knowledge is sometimes reflected in the solutions' quality. Especially in self-organizing hierarchical systems (Steghöfer et al., 2013), we can often assume that *regional* knowledge is available: In such systems, the overall system is decomposed in a system of systems in which each subsystem is represented by an intermediary. Since intermediaries encapsulate the essence of the agents they control, we can often suppose they have regional knowledge (i.e., global knowledge with regard to their subsystem) about their subordinates (Abdallah and Lesser, 2004). Intermediaries can use this information to create a suitable partitioning of their subordinates.

Usually, algorithms that solve the PP are either **1)** specialized to a specific problem in a specific domain or **2)** very restrictive with regard to the possibility to specify mandatory characteristics of the resulting partitioning's structure in the form of the number and the size of partitions. These properties limit the algorithms' applicability. With respect to point 2), most algorithms either do not allow to characterize valid partitionings at all (e.g., (Ogston et al., 2003)) or the user or the agents have to be very specific. Using the well-known k-means clustering algorithm (MacQueen, 1967), for instance, the user has to specify the number of partitions k exactly. Because a suitable exact number of partitions is often not known (further drawbacks of k-means, e.g., the formation of partitions of similar size, are discussed in (Äyrämö and Kärkkäinen, 2006)), there are different approaches that extend the k-means algorithm by the possibility to automatically find a suitable number of partitions for a given data set, such as the x-means algorithm (Ishioka, 2005). In contrast to these approaches, we want to allow the user or the system itself to specify suitable *ranges* for the number and the size of partitions, i.e., the minimum n_{min} and the maximum n_{max} number of partitions *as well as* their minimum s_{min} and maximum s_{max} size. These partitioning constraints allow, e.g., to specify appropriate sizes of subsystems in the context of compartmentalization in MAS. As mentioned at the beginning of this section, compartmentalization is a possibility to decompose the complexity of a system's task. In (Steghöfer et al., 2013), e.g., the clustering of power plants into virtual power plants decreases the time needed to calculate schedules for them, a task whose complexity depends on the number of power plants involved. In this example, it is required that the size of each virtual power plant is not less than two and below a certain threshold restricting the maximum time needed for schedule creation.

In this paper, we present *PSOPP*, a ***P**article **S**warm **O**ptimizer for the **P**artitioning **P**roblem*. PSOPP is based on *Particle Swarm Optimization* (PSO) (Kennedy and Eberhart, 1995), a biologically-inspired computational method and metaheuristic for optimization in large search spaces. The application of a metaheuristic is suitable because of the PP's complexity. For this reason, a plethora of metaheuristics solving related problems can be found in the body of literature: In (Chu and Beasley, 1998), a genetic algorithm (GA) is used to solve the original SPP, meaning that the GA needs a pre-calculated set of feasible partitions. As discussed before, we

want to avoid this in our approach. In theory, their GA could also be extended to respect prescribed ranges for the number of partitions by so-called *base constraints*. However, since their GA allows the generation of invalid solutions, it would not benefit from a reduced search space and require additional heuristics for the correction of solution candidates. Using PSO for data clustering has been proposed in (Van der Merwe and Engelbrecht, 2003), where each particle represents a complete solution of the clustering. In (Alam et al., 2008), the authors present an evolutionary PSO algorithm in which a new generation of particles can replace those contributing to a bad solution to be able to leave local optima. Importantly, their particles represent partial solutions, i.e., a single centroid, instead of a complete solution.

In contrast to these and the other afore-mentioned approaches, PSOPP **1)** solves the PP in a general manner and **2)** allows to specify and efficiently deal with suitable ranges for the number as well as the size of partitions. Our central idea – which could also be applied to other metaheuristics – is to use basic set operations to come to a solution. Because we define these operations in a way that their application always maintains solution correctness, PSOPP combs through a search space that only contains correct solutions, which is advantageous with regard to its performance. Because PSOPP is initialized with a correct solution candidate, it is an anytime algorithm. Moreover, PSOPP can be customized to a specific application by devising an appropriate fitness function that assesses the quality of solutions and thus steers the search for them. Due to these characteristics, PSOPP can be applied to many different applications in which solving the PP is relevant and global knowledge is available. In conjunction with the control loop presented in (Steghöfer et al., 2013), PSOPP can be used to establish self-organizing hierarchical system structures that overcome the drawbacks of strictly weak self-organization (Di Marzo Serugendo et al., 2005).

The remainder of this paper is structured as follows: In Sect. 2, we give an introduction to the principle of PSO and some of its variants for combinatorial optimization. Afterwards, we present our algorithm, PSOPP, in Sect. 3. Sect. 4 outlines evaluation results showing that PSOPP efficiently solves the PP in various scenarios. Finally, we conclude the paper and give an outlook on future work in Sect. 5.

2 PARTICLE SWARM OPTIMIZATION

PSO is a search heuristic for optimization problems. Its principle is based on the flocking behavior of birds or schools of fish. Before we present a special form of PSO that is applicable to discrete optimization problems, such as the PP, in Sect. 2.2, we explain the basic idea of PSO in Sect. 2.1.

2.1 General Definition

In the original definition of PSO (Kennedy and Eberhart, 1995), a swarm of *particles* moves around in an n-dimensional continuous search space in order to find nearly optimal solutions by iteratively improving *candidate solutions* of the optimization problem. Such a candidate solution is represented by a particle's position in the search space. Its quality is rated by a fitness function: the better the fitness, the better the solution. To be able to improve the quality of its solution over time in a target-oriented manner, each particle Π_i is aware of *its best found solution* $\mathcal{B}_i$ and the *best found solution* $\mathcal{B}_{\mathcal{N}_i}$ *in its neighborhood* $\mathcal{N}_i$. If a particle's neighborhood consists of all particles, $\mathcal{B}_{\mathcal{N}_i}$ corresponds to the *global best found solution* $\mathcal{B}$.

Initially, particles usually start at random positions. In each iteration, the particles update their positions and best found solutions. The algorithm terminates, e.g., after a certain amount of iterations or if the particles converge to a (local) optimum. Its outcome is the global best found solution $\mathcal{B}$. In detail, a particle Π_i determines its *position* $\boldsymbol{x}_i(t+1)$ for the next iteration $t+1$ on the basis of its current position $\boldsymbol{x}_i(t)$ and its updated *velocity* $\boldsymbol{v}_i(t+1)$:

$$\boldsymbol{x}_i(t+1) = \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t+1) \tag{1}$$

$$\begin{aligned}\boldsymbol{v}_i(t+1) = {} & \omega \cdot \boldsymbol{v}_i(t) + c_1 \cdot r_1 \cdot (\mathcal{B}_i - \boldsymbol{x}_i(t)) \\ & + c_2 \cdot r_2 \cdot (\mathcal{B}_{\mathcal{N}_i} - \boldsymbol{x}_i(t))\end{aligned} \tag{2}$$

$$\text{with } \omega, c_1, c_2 \in \mathbb{R}_0^+,\ r_1, r_2 \in [0,1],$$

$$\text{and } \forall t : \boldsymbol{x}_i(t), \boldsymbol{v}_i(t), \mathcal{B}_i, \mathcal{B}_{\mathcal{N}_i} \in \mathbb{R}^n$$

Because $\boldsymbol{v}_i(t+1)$ depends on the current velocity $\boldsymbol{v}_i(t)$, it embodies a certain *inertia*. To establish the right balance between exploitation and exploration, a particle's motion in the search space is further influenced by its best found solution $\mathcal{B}_i$ and the best found solution $\mathcal{B}_{\mathcal{N}_i}$ in its neighborhood $\mathcal{N}_i$. The particle's inertia and its attraction towards $\mathcal{B}_i$ and $\mathcal{B}_{\mathcal{N}_i}$, i.e., the trade-off between exploration and exploitation, can be adjusted by the constants ω, c_1, and c_2. The random numbers r_1 and r_2 are regenerated in every iteration.

2.2 Discrete Particle Swarm Optimization

PSO as defined in Sect. 2.1 is not applicable to discrete, e.g., combinatorial, optimization problems,

such as the PP. (Kennedy and Eberhart, 1997) solve this dilemma for n-dimensional binary search spaces by introducing *Discrete PSO* (DPSO) in which the positions $\boldsymbol{x}_i(t)$, $\mathcal{B}_i$, and $\mathcal{B}_{\mathcal{N}_i}$ are values of the domain $\{0,1\}^n$. While the domain and the definition of the velocity $\boldsymbol{v}_i(t+1) \in \mathbb{R}^n$ are not modified (see Eq. 2), the semantics of the velocity changes. In contrast to the original definition, each component $(v_i(t+1))_j \in \mathbb{R}$ of the vector $\boldsymbol{v}_i(t+1)$ represents a probability that the jth component of the particle's position $\boldsymbol{x}_i(t+1)$ is either 0 or 1. Eq. 1 therefore becomes invalid.

Another DPSO approach, which is called *Jumping PSO* (JPSO) (Garcia and Perez, 2008), omits the concept of the velocity as defined in (Kennedy and Eberhart, 1995). In simplified terms, JPSO redefines the motion of particles by replacing the linear combinations in Eq. 1 and Eq. 2 by an "either-or" operation that makes them "jump" through the search space:

$$\boldsymbol{x}_i(t+1) = \begin{cases} rdm(\boldsymbol{x}_i(t)) & \text{if } r_i \leq c_{rdm} \\ appr(\boldsymbol{x}_i(t), \mathcal{B}_i) & \text{if } c_{rdm} < r_i \leq c^*_{\mathcal{B}_i} \\ appr(\boldsymbol{x}_i(t), \mathcal{B}_{\mathcal{N}_i}) & \text{if } c^*_{\mathcal{B}_i} < r_i \leq c^*_{\mathcal{B}_{\mathcal{N}_i}} \\ appr(\boldsymbol{x}_i(t), \mathcal{B}) & \text{otherwise} \end{cases} \quad (3)$$

$$c_{rdm}, c_{\mathcal{B}_i}, c_{\mathcal{B}_{\mathcal{N}_i}}, c_{\mathcal{B}} \in [0,1],\ c_{rdm} + c_{\mathcal{B}_i} + c_{\mathcal{B}_{\mathcal{N}_i}} + c_{\mathcal{B}} = 1,$$

$$r_i \in [0,1],\ c^*_{\mathcal{B}_i} = c_{rdm} + c_{\mathcal{B}_i},\ c^*_{\mathcal{B}_{\mathcal{N}_i}} = c_{rdm} + c_{\mathcal{B}_i} + c_{\mathcal{B}_{\mathcal{N}_i}}$$

Eq. 3 states that a particle Π_i either makes a random move $rdm(\boldsymbol{x}_i(t))$ with a probability of c_{rdm} or approaches $appr(\boldsymbol{x}_i(t), \beta)$ a specific solution candidate $\beta \in \{\mathcal{B}_i, \mathcal{B}_{\mathcal{N}_i}, \mathcal{B}\}$ with a probability of $c_{\mathcal{B}_i}$, $c_{\mathcal{B}_{\mathcal{N}_i}}$, or $c_{\mathcal{B}}$, respectively. In each iteration, this direction is determined by a random number r_i that is generated individually for each particle. Similarly to Eq. 2, the constants $c_{rdm}, c_{\mathcal{B}_i}, c_{\mathcal{B}_{\mathcal{N}_i}}$, and $c_{\mathcal{B}}$ stipulate the particles' attitude towards exploration and exploitation. The idea of JPSO has been successfully applied to a number of high dimensional combinatorial problems (see, e.g., (Consoli et al., 2010; Seren, 2011)).

3 THE PARTICLE SWARM OPTIMIZER FOR SOLVING SET PARTITIONING PROBLEMS

With regard to our algorithm, *PSOPP*, each particle embodies a solution of the PP, i.e., a partitioning of the set of elements $\mathfrak{G}$. PSOPP is inspired by DPSO's derivative JPSO (see Sect. 2.2). The motion of particles is thus not subject to inertia, i.e., $\boldsymbol{x}_i(t+1)$ does not depend on the modifications made to move from $\boldsymbol{x}_i(t-1)$ to $\boldsymbol{x}_i(t)$. In PSOPP, a particle's motion is influenced by its best found solution $\mathcal{B}_i$ and the best found solution $\mathcal{B}_{\mathcal{N}_i}$ in its neighborhood $\mathcal{N}_i$. This complies with the general definition of PSO outlined in Sect. 2.1. While we could easily extend PSOPP such that its particles' motion is additionally influenced by the global best solution $\mathcal{B}$ – as is the case with the definition of JPSO (see Sect. 2.2) –, we deliberately omit this feature for the sake of simplicity. With respect to the definition of JPSO's behavior in Eq. 3, this corresponds to a probability of $c_{\mathcal{B}} = 0$.

3.1 Constraining Valid Solutions

As stated in Sect. 1, PSOPP allows to specify mandatory characteristics of a solution, i.e., partitioning, in terms of the minimum n_{min} and the maximum n_{max} number of partitions ($1 \leq n_{min} < n_{max} \leq |\mathfrak{G}|$) as well as their minimum s_{min} and maximum s_{max} size ($1 \leq s_{min} < s_{max} \leq |\mathfrak{G}|$). These boundaries represent hard constraints. Obviously, as the possible number and size of partitions are interconnected, one has to make sure that the problem is not overconstrained. In case of $\mathfrak{G} = \{\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3\}$, e.g., there is no valid solution if we set n_{min} and s_{min} to 2. Either n_{min} or s_{min} would have to be relaxed, i.e., set to 1. Because we define PSOPP's operations for the particles' motion in a way that always preserves the correctness of solution candidates with respect to the constraints, partitionings that do not meet them are not represented in the search space. As we show in our evaluation in Sect. 4, suitable boundaries can thus lower the time needed to find high-quality solutions. In the following, we call these boundaries *partitioning constraints*.

3.2 The Algorithm's Basic Procedure

Having defined valid partitionings by means of $n_{min}, n_{max}, s_{min}, s_{max}$ as well as the particles' attitude towards exploration and exploitation by fixing the constants $c_{rdm}, c_{\mathcal{B}_i}, c_{\mathcal{B}_{\mathcal{N}_i}}$, PSOPP creates a predefined number of particles at random or predetermined positions (the set of particles does not change at runtime). The latter is especially suitable when a reorganization of an existing system structure has to take place: If the current structure does not contradict the partitioning constraints, it can be used as a starting point for the self-organization process. Mixing predefined and randomly generated initial partitionings allows to hold up diversity. When searching for an initial system structure, particles are created at random positions.

The position $\boldsymbol{x}_i(t)$ of each particle Π_i represents a partitioning $\mathcal{P}$ (in the following, we use $\mathcal{P}$ syn-

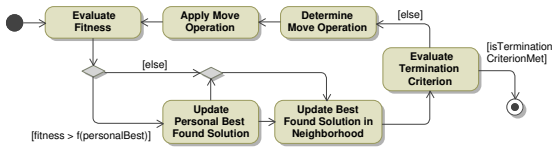

Figure 1: Actions performed by particles in each iteration.

onymous for $\boldsymbol{x}_i(t)$) that consists of $|\mathcal{P}|$ partitions ($n_{min} \leq |\mathcal{P}| \leq n_{max}$). Every partition $K \in \mathcal{P}$ consists of $s_{min} \leq |K| \leq s_{max}$ elements. All particles concurrently explore the search space in search of better solutions by modifying their current positions (at random or by approaching other solutions) as long as a specific termination criterion is not met. For this purpose, in each iteration, a particle Π_i performs the following actions that are also depicted in Fig. 1:

1. Evaluate the fitness $f(\mathcal{P})$ of the represented partitioning $\mathcal{P}$. The fitness function corresponds to the "metric" used in the PP's definition in Sect. 1.
2. If the particle's fitness $f(\mathcal{P})$ is higher than the fitness $f(\mathcal{B}_i)$ of its best found solution $\mathcal{B}_i$, set $\mathcal{B}_i$ to $\mathcal{P}$ and inform other particles Π_j with $\Pi_i \in \mathcal{N}_j$ about the improvement so that they can update their best found solution $\mathcal{B}_{\mathcal{N}_j}$ in their neighborhood $\mathcal{N}_j$.
3. Update the best found solution $\mathcal{B}_{\mathcal{N}_i}$ in the particle's neighborhood $\mathcal{N}_i$.
4. Stop if the termination criterion is met.
5. Otherwise, opt for the direction in which to move by randomly generating $r_i \in [0,1]$ (see Eq. 3), i.e., choose whether a random move or an approach operation should be applied. In case of an approach operation, r_i also determines the position that should be approached (see Eq. 3).
6. Determine the new position $\mathcal{P}'$ by applying the selected move operation to $\mathcal{P}$.

Once all particles terminated, PSOPP returns the best found solution $\mathcal{B}$. Possible termination criteria are, e.g., a predefined amount of time, a predefined number of iterations (i.e., moves through the search space), a predefined threshold for the minimum fitness value, or a combination of these criteria.

3.3 Similarity of Partitionings

The purpose of an approach operation is to increase the *similarity* of two partitionings $\mathcal{P}$ and $\mathcal{Q}$ by assimilating characteristics from $\mathcal{Q}$ into $\mathcal{P}$. With regard to the search space, the intention is that the particle representing $\mathcal{P}$ might find better solutions in the neighborhood of $\mathcal{Q}$. In this section, we define the *similarity* of partitionings based on a definition by (Kudo and Murai, 2009). Note that the similarity does not give an indication of how many operations/moves are necessary to transfer one partitioning into another (i.e., to move from one position to another). Instead, it compares partitionings with regard to their composition. According to (Kudo and Murai, 2009), we base the definition of the similarity of two partitionings $\mathcal{P}, \mathcal{Q}$ on the definition of a *refinement* and the *intersection* of two partitionings.

Definition (Refinement)**.** *Partitioning $\mathcal{P}$ is a **refinement** $ref(\mathcal{P},\mathcal{Q})$ of partitioning $\mathcal{Q}$ if and only if all partitions $K \in \mathcal{P}$ are subsets of partitions $L \in \mathcal{Q}$:*

$$ref(\mathcal{P},\mathcal{Q}) :\Leftrightarrow \forall K \in \mathcal{P} : \exists L \in \mathcal{Q} : K \subseteq L \quad (4)$$

Hence, if $\mathcal{P}$ is a refinement of $\mathcal{Q}$, $\mathcal{P}$ does not contain less partitions than $\mathcal{Q}$ (i.e., $|\mathcal{P}| \geq |\mathcal{Q}|$). For instance, $\mathcal{P} = \{\{\boldsymbol{g}_1,\boldsymbol{g}_2\},\{\boldsymbol{g}_3\},\{\boldsymbol{g}_4\}\}$ is a refinement of $\mathcal{Q} = \{\{\boldsymbol{g}_1,\boldsymbol{g}_2,\boldsymbol{g}_3\},\{\boldsymbol{g}_4\}\}$.

Definition (Intersection of Partitionings)**.** *The **intersection** $\mathcal{P} \cap \mathcal{Q}$ of two partitionings $\mathcal{P},\mathcal{Q}$ is the set of all non-empty intersections of partitions in $\mathcal{P}$ and $\mathcal{Q}$:*

$$\mathcal{P} \cap \mathcal{Q} :\Leftrightarrow \{K \cap L \mid K \in \mathcal{P} \wedge L \in \mathcal{Q} \wedge K \cap L \neq \emptyset\}$$

Note that the intersection $\mathcal{P} \cap \mathcal{Q}$ is always a refinement of $\mathcal{P}$ and $\mathcal{Q}$. For example, the intersection $\mathcal{R} \cap \mathcal{Q} = \{\{\boldsymbol{g}_1,\boldsymbol{g}_2\},\{\boldsymbol{g}_3\},\{\boldsymbol{g}_4\}\}$, which equals $\mathcal{P}$ in the example above, is a refinement of $\mathcal{R} = \{\{\boldsymbol{g}_1,\boldsymbol{g}_2\},\{\boldsymbol{g}_3,\boldsymbol{g}_4\}\}$ and $\mathcal{Q} = \{\{\boldsymbol{g}_1,\boldsymbol{g}_2,\boldsymbol{g}_3\},\{\boldsymbol{g}_4\}\}$.

Definition (Similarity of Partitionings)**.** *The **similarity** $sim(\mathcal{P},\mathcal{Q}) \in]0,1]$ of two partitionings $\mathcal{P},\mathcal{Q}$ is based on the ratio of the sum of their cardinalities to the cardinality of their intersection:*

$$sim(\mathcal{P},\mathcal{Q}) := \frac{|\mathcal{P}| + |\mathcal{Q}|}{2 \cdot |\mathcal{P} \cap \mathcal{Q}|}$$

The intention of this definition of similarity is the following: If the elements of a partition $K \in \mathcal{P}$ are distributed over multiple partitions in $\mathcal{Q}$, the similarity decreases with the cardinality of the intersection $\mathcal{P} \cap \mathcal{Q}$. So the more elements are in the same partitions in $\mathcal{P}$ and $\mathcal{Q}$ (i.e., the more elements constitute the *partitions'* intersection), the smaller $|\mathcal{P} \cap \mathcal{Q}|$ and thus the more similar the partitionings. In other words, the intersection $\mathcal{P} \cap \mathcal{Q}$ (i.e., the refinement) should be as similar as possible to the two given partitionings. Resulting from this definition, $sim(\mathcal{P},\mathcal{Q}) = 1$ if and only if $\mathcal{P} = \mathcal{Q}$, because then $\mathcal{P} \cap \mathcal{Q} = \mathcal{P} = \mathcal{Q}$. While not required, note that this definition of similarity does not allow to compare two similarity values $sim(\mathcal{P},\mathcal{Q})$ and $sim(\mathcal{R},\mathcal{S})$ if they stem from four different partitionings. Regarding the two examples above, $sim(\mathcal{R},\mathcal{Q}) = \frac{2+2}{2\cdot 3} = \frac{4}{6}$ is smaller than $sim(\mathcal{P},\mathcal{Q}) = \frac{3+2}{2\cdot 3} = \frac{5}{6}$ since $\mathcal{P}$ is a refinement of $\mathcal{Q}$.

Based on these definitions, we show that the operations enabling particles to approach each other always increase the similarity of the represented partitionings (see Sect. 3.5). Before we explain these operations in detail, we introduce the basic operations by means of random moves in the search space.

3.4 Random Moves in the Search Space

The motion of particles is a key factor in PSO because it is the only measure to find better solution candidates. As a solution of the PP is a partitioning (that is a set of sets), the motion of particles in the search space can be realized by the two set operations *split* and *join* (Apt and Witzel, 2007). In each iteration, each PSOPP particle makes exactly one move, either in a random direction or by approaching a specific position in the search space in a target-oriented manner (see Sect. 3.5). The corresponding operator is randomly selected. In this section, we concentrate on *random moves*, i.e., operators that modify the partitioning at random. In case the selected operator cannot be applied without violating a constraint, another operator is chosen. Because of the partitioning constraints, there are situations in which neither the split nor the join operator can be applied. For such situations, we introduce an additional *exchange* operation.

Unless otherwise stated, we use $\mathcal{P}_* = \{\{g_1,g_2,g_4,g_5\},\{g_3,g_6\},\{g_7,g_8\}\}$ with $s_{min} = 2$, $s_{max} = 4$, $n_{min} = 2$, and $n_{max} = 3$ to illustrate the operators' application in our examples.

Random Split. The split operation divides a *randomly splitable partition* $K \in \mathcal{P}$ into two new non-empty disjoint partitions L and M such that $K = L \cup M$. For the resulting partitioning $\mathcal{P}'$, we have $\mathcal{P}' = (\mathcal{P} \setminus \{K\}) \cup \{L,M\}$. Note that a split operation can only be applied if the original partitioning K is big enough, i.e., if $|K| \geq 2 \cdot s_{min}$. That is because the resulting partitions L and M both have to fulfill the minimum size constraint, i.e., $|L|,|M| \geq s_{min}$. Furthermore, because the split operation increases the number of partitions $|\mathcal{P}'|$ of the resulting partitioning $\mathcal{P}'$ compared to the original partitioning $\mathcal{P}$ by one (i.e., $|\mathcal{P}'| = |\mathcal{P}| + 1$), it can only be applied if the number of partitions $|\mathcal{P}|$ in $\mathcal{P}$ is below the maximum number of partitions n_{max}. That way, it is ensured that $\mathcal{P}'$ also complies with n_{max}. Summarizing, the set of *randomly splitable partitions* $\sigma_{rdm}(\mathcal{P})$ is defined as:

$$\sigma_{rdm}(\mathcal{P}) :\Leftrightarrow \{K \mid K \in \mathcal{P} \wedge |K| \geq 2 \cdot s_{min} \wedge |\mathcal{P}| < n_{max}\}$$

In our example $\mathcal{P}_*$, only $K_* = \{g_1,g_2,g_4,g_5\}$ is randomly splitable, resulting, e.g., in a partitioning $\mathcal{P}_*{}' = \{\{g_1,g_2\},\{g_4,g_5\},\{g_3,g_6\},\{g_7,g_8\}\}$.

Random Join. The join operation merges a *randomly joinable partition* $K \in \mathcal{P}$ and a *randomly joinable counterpart* $L \in \mathcal{P}$ into a single new partition M such that $K \cup L = M$. For the resulting partitioning $\mathcal{P}'$, we have $\mathcal{P}' = (\mathcal{P} \setminus \{K,L\}) \cup \{K \cup L\}$. Because the resulting partition M has to meet the maximum size constraint, L must be a partition that can be integrated into K without exceeding the maximum allowed size, i.e., $|K| + |L| \leq s_{max}$. Since the join operator decreases the number of partitions in the resulting partitioning $\mathcal{P}'$ by one, the operator can further only be applied if $\mathcal{P}$ features a sufficient number of partitions, i.e., if $|\mathcal{P}| > n_{min}$. Otherwise, $\mathcal{P}'$ would violate the minimum number of partitions constraint. Summarizing, the sets of *randomly joinable partitions* $\iota_{rdm}(\mathcal{P})$ and *randomly joinable counterparts* $\iota^{\leftrightarrows}_{rdm}(K,\mathcal{P})$ are defined as follows:

$$\iota_{rdm}(\mathcal{P}) :\Leftrightarrow \{K \mid K \in \mathcal{P} \wedge \iota^{\leftrightarrows}_{rdm}(K,\mathcal{P}) \neq \emptyset \wedge |\mathcal{P}| > n_{min}\}$$

$$\iota^{\leftrightarrows}_{rdm}(K,\mathcal{P}) :\Leftrightarrow \{L \mid L \in \mathcal{P} \wedge |K| + |L| \leq s_{max} \wedge K \neq L\}$$

With regard to our example $\mathcal{P}_*$, randomly joinable partitions are $L_* = \{g_3,g_6\}$ and $M_* = \{g_7,g_8\}$ with randomly joinable counterparts $\{M_*\}$ and $\{L_*\}$, respectively. Merging L_* and M_* yields $\mathcal{P}_*{}' = \{\{g_1,g_2,g_4,g_5\},\{g_3,g_6,g_7,g_8\}\}$.

Random Exchange. Obviously, there are situations in which neither the split nor the join operator can be applied (particles must not violate the constraints temporarily since the search space only contains valid solutions). For example, if s_{min} and s_{max} or n_{min} and n_{max} are equal, not a single particle is able to make a move using the split or the join operation: For instance, consider a set of elements $\mathfrak{G} = \{g_1,\ldots,g_6\}$, boundaries $s_{min} = 2$, $s_{max} = 4$, $n_{min} = 2$, and $n_{max} = 2$, and a partitioning $\mathcal{P} = \{\{g_1,g_2,g_3\},\{g_4,g_5,g_6\}\}$. But even if $s_{min} \neq s_{max}$ and $n_{min} \neq n_{max}$, specific combinations of $s_{min}, s_{max}, n_{min}$, and n_{max} can cause individual particles to freeze, e.g., if we replace $n_{max} = 2$ by $n_{max} = 3$ in our example. To prevent the particles from becoming jammed, we additionally introduce an *exchange* operator that atomically swaps some of the elements of two partitions.

The exchange operation interchanges the proper subset $\hat{K} \subset K$ (with $\hat{K} \neq \emptyset$) and the subset $\hat{L} \subseteq L$ ($\hat{L}$ is allowed to be the empty set $\emptyset$) between a *randomly exchangeable partition* $K \in \mathcal{P}$ and a *randomly exchangeable counterpart* $L \in \mathcal{P}$. Using the non-empty proper subset $\hat{K}$ of K avoids that the operation has no effect at all (as would be the case if all or no elements of K were integrated into L and vice versa). We deliberately allow $\hat{L}$ to be empty in order to handle situations as given in the example above: Regarding parti-

tioning $\mathcal{P} = \{\{g_1, g_2, g_3\}, \{g_4, g_5, g_6\}\}$, we can simply move $\hat{K} = \{g_3\}$ from $K = \{g_1, g_2, g_3\}$ into $L = \{g_4, g_5, g_6\}$. If we did not allow $\hat{L} = \emptyset$, we would have to perform multiple consecutive exchange operations to achieve the same result. With respect to the example, we would need two operations, e.g., by exchanging $\{g_2, g_3\}$ and $\{g_4\}$, and finally $\{g_4\}$ and $\{g_2\}$.

Basically, the exchange operation corresponds to a join that is followed by a split. Since $\hat{K}$ is a non-empty proper subset of K, as the split operation, it yields two non-empty partitions. Because an exchange between two partitions of size one would contradict this characteristic or not have any effect, we define the sets of *randomly exchangeable partitions* $\varepsilon_{rdm}(\mathcal{P})$ and *randomly exchangeable counterparts* $\varepsilon^{\leftrightarrows}_{rdm}(K, \mathcal{P})$ as:

$$\varepsilon_{rdm}(\mathcal{P}) :\Leftrightarrow \{K \mid K \in \mathcal{P} \wedge |K| > 1\}$$
$$\varepsilon^{\leftrightarrows}_{rdm}(K, \mathcal{P}) :\Leftrightarrow \mathcal{P} \setminus \{K\}$$

When integrating an arbitrary non-empty subset $\hat{K} \subset K$ into L, $\hat{K}$ as well as the subset $\hat{L} \subseteq L$ that is integrated into K must be specified in a way that the condition $|K'|, |L'| \in [s_{min}, s_{max}]$ holds for the resulting partitions K', L'. More precisely, while the size of $\hat{K}$ is randomly chosen between 1 and $|K| - 1$ to ensure that $\hat{K}$ is a non-empty proper subset of K, valid cardinalities of $\hat{L}$ are subject to $|\hat{K}|$. In detail, $|\hat{L}| \leq min\{|L|, min\{(|L| + |\hat{K}|) - s_{min}, s_{max} - (|K| - |\hat{K}|)\}\}$ and $|\hat{L}| \geq max\{max\{0, s_{min} - (|K| - |\hat{K}|)\}, (|L| + |\hat{K}|) - s_{max}\}$ must hold for the randomly determined set $\hat{L}$ to guarantee that the resulting partitioning $\mathcal{P}' = (\mathcal{P} \setminus \{K, L\}) \cup \{(K \setminus \hat{K}) \cup \hat{L}, (L \setminus \hat{L}) \cup \hat{K}\}$ respects s_{min} and s_{max}. In our example $\mathcal{P}_*$, we can, e.g., exchange $\hat{K}_* = \{g_1, g_2, g_5\}$ and $\hat{M}_* = \{g_7\}$ between K_* and L_*, resulting in $\mathcal{P}_*{}' = \{\{g_4, g_7\}, \{g_3, g_6\}, \{g_1, g_2, g_5, g_8\}\}$. If all partitions are singletons (i.e., if $|\mathcal{P}| = |\mathfrak{G}|$), it is not possible to apply the random exchange operator since $\varepsilon_{rdm}(\mathcal{P}) = \emptyset$. In such a case, PSOPP tries to apply either the random split or the random join operator.

These three operations allow to create new or dissolve existing partitions or exchange elements between them while maintaining the properties of a partitioning and complying with the partitioning constraints (see Sect. 3.1). In this section, we focused on random moves, where we cannot make any statement with regard to the change in similarity to another partitioning. In the next section, we explain how particles use the basic split, join, and exchange operators to approach a specific position in the search space.

3.5 Approach of Other Particles

When a particle Π_i approaches $\mathcal{B}_i$ or $\mathcal{B}_{\mathcal{N}_i}$, we ensure that the similarity of the modified partitioning $\mathcal{P}$ and the approached partitioning $\mathcal{Q} \in \{\mathcal{B}_i, \mathcal{B}_{\mathcal{N}_i}\}$ is increased. Recalling the definition of the similarity (see Sect. 3.3), this can, among other possibilities, be achieved by increasing $|\mathcal{P}|$, i.e., the number of partitions in $\mathcal{P}$ without changing $|\mathcal{P} \cap \mathcal{Q}|$ at all, or decreasing $|\mathcal{P} \cap \mathcal{Q}|$ (note that a decrease of $|\mathcal{P} \cap \mathcal{Q}|$ might come along with a decrease of $|\mathcal{P}|$). The former can be obtained by splitting a partition that contains elements that are members of two or more partitions in $\mathcal{Q}$, whereas the latter can be obtained by merging two partitions that both contain elements that are members of a single partition in $\mathcal{Q}$. In contrast to random moves, the applicability of the approach operations does not only depend on $\mathcal{P}$'s cardinality and the size of its partitions but also on $\mathcal{P}$'s and $\mathcal{Q}$'s composition. Obviously, an approach is not possible if $\mathcal{P} = \mathcal{Q}$.

Unless otherwise stated, we use a partitioning $\mathcal{P}_* = \{\{g_1, g_2, g_4, g_5\}, \{g_3, g_6\}, \{g_7, g_8\}\}$ that approaches $\mathcal{Q}_* = \{\{g_1, g_2\}, \{g_4, g_6, g_7\}, \{g_3, g_5, g_8\}\}$ with $s_{min} = 2$, $s_{max} = 4$, $n_{min} = 2$, and $n_{max} = 4$ to illustrate the operators' application in our examples.

Approach Split. Analogously to the definition of $\sigma_{rdm}(\mathcal{P})$, this operator can only be applied if $|\mathcal{P}| < n_{max}$. Furthermore, a partition K must fulfill $|K| \geq 2 \cdot s_{min}$ to be contained in the set of *splitable partitions* $\sigma(\mathcal{P}, \mathcal{Q})$. Here, this property results from the definition of *extractable subsets* $\sigma^{\uparrow}(K, \mathcal{P}, \mathcal{Q})$:

$$\sigma(\mathcal{P}, \mathcal{Q}) :\Leftrightarrow \{K \mid K \in \mathcal{P} \wedge \sigma^{\uparrow}(K, \mathcal{P}, \mathcal{Q}) \neq \emptyset \wedge |\mathcal{P}| < n_{max}\}$$
$$\sigma^{\uparrow}(K, \mathcal{P}, \mathcal{Q}) :\Leftrightarrow \{L \mid L \in (\mathcal{P} \cap \mathcal{Q}) \wedge L \subset K \wedge |K \setminus L| \geq s_{min} \wedge |L| \geq s_{min}\}$$

An extractable subset $L \in \sigma^{\uparrow}(K, \mathcal{P}, \mathcal{Q})$ is thus a *proper* subset of $K \in \mathcal{P}$, i.e., with respect to $\mathcal{Q}$, K contains further elements that are not contained in the same partition as the elements in L. Hence, the split operator cannot be applied to approach another particle if all partitions in $\mathcal{P}$ are subsets of partitions in $\mathcal{Q}$, i.e., if $\mathcal{P}$ is a refinement of $\mathcal{Q}$ (see Eq. 4). For the resulting partitioning, we have $\mathcal{P}' = (\mathcal{P} \setminus \{K\}) \cup \{K \setminus L, L\}$. Extracting the set L from K increases the similarity between $\mathcal{P}$ and $\mathcal{Q}$ because $|\mathcal{P}'| = |\mathcal{P}| + 1$, $\mathcal{Q}$ is not changed, and $\mathcal{P}' \cap \mathcal{Q} = \mathcal{P} \cap \mathcal{Q}$, i.e., the intersection of the partitionings does not change either. With regard to $\mathcal{P}_*$ and $\mathcal{Q}_*$, $K_* = \{g_1, g_2, g_4, g_5\}$ is the only splitable partition with extractable subset $L_* = \{g_1, g_2\}$ (L_* is the only element of $\sigma^{\uparrow}(K_*, \mathcal{P}_*, \mathcal{Q}_*)$). A split results in $\mathcal{P}_*{}' = \{\{g_1, g_2\}, \{g_4, g_5\}, \{g_3, g_6\}, \{g_7, g_8\}\}$.

Approach Join. As before, a join can only be applied if $|\mathcal{P}| > n_{min}$. Similarly to the definition of *randomly* joinable partitions, *joinable parti-*

tions $\iota(\mathcal{P},Q)$ are those partitions for which *joinable counterparts* $\iota^{\leftrightarrows}(K,\mathcal{P},Q)$ exist:

$$\iota(\mathcal{P},Q) :\Leftrightarrow \{K \mid K \in \mathcal{P} \wedge \iota^{\leftrightarrows}(K,\mathcal{P},Q) \neq \emptyset \wedge |\mathcal{P}| > n_{min}\}$$

$$\iota^{\leftrightarrows}(K,\mathcal{P},Q) :\Leftrightarrow \{L \mid L \in \mathcal{P} \wedge |K|+|L| \leq s_{max} \wedge K \neq L \wedge \underbrace{\exists M \in Q : (M \cap K \neq \emptyset \wedge M \cap L \neq \emptyset)}_{C}\}$$

Please note that the definition of joinable counterparts $\iota^{\leftrightarrows}(K,\mathcal{P},Q)$ is very similar to the definition of *randomly* joinable counterparts $\iota^{\leftrightarrows}_{rdm}(K,\mathcal{P})$. To ensure that $\mathcal{P}$ approaches Q, we introduce an additional condition C that implies $M \nsubseteq K$ because M does not only contain elements of K but also of L (with $M \in Q$ and $K,L \in \mathcal{P}$). That way, we bring together elements that are in a single partition in Q but spread over two or more partitions K,L in $\mathcal{P}$. Note that Q cannot be approached by a join if all partitions in $\mathcal{P}$ are supersets of partitions in Q, i.e., if Q is a refinement of $\mathcal{P}$ (see Eq. 4). For the resulting partitioning, we have $\mathcal{P}' = (\mathcal{P} \setminus \{K,L\}) \cup \{K \cup L\}$. The similarity between $\mathcal{P}$ and Q is increased because $|\mathcal{P}'| = |\mathcal{P}| - 1$, Q is not changed, and $|\mathcal{P}' \cap Q| \leq |\mathcal{P} \cap Q| - 1$. Note that we have to write "$\leq$" since K and L might both contain elements that are contained in a further partition $M' \in Q$ with $M' \neq M$. With regard to $\mathcal{P}_*$ and Q_*, $K_* = \{\boldsymbol{g}_3, \boldsymbol{g}_6\}$ and $L_* = \{\boldsymbol{g}_7, \boldsymbol{g}_8\}$ are joinable partitions with counterparts L_* and K_*, respectively. A join results in $\mathcal{P}_*{}' = \{\{\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_4, \boldsymbol{g}_5\}, \{\boldsymbol{g}_3, \boldsymbol{g}_6, \boldsymbol{g}_7, \boldsymbol{g}_8\}\}$.

Approach Exchange. If neither a split nor a join can be used to approach a partitioning $Q \neq \mathcal{P}$, PSOPP falls back on the exchange operator that swaps one or more elements between a partition K contained in the set of *exchangeable partitions* $\varepsilon(\mathcal{P},Q)$ and one of K's *exchangeable counterparts* $\varepsilon^{\leftrightarrows}(K,\mathcal{P},Q)$:

$$\varepsilon(\mathcal{P},Q) :\Leftrightarrow \{K \mid K \in \mathcal{P} \wedge \varepsilon^{\leftrightarrows}(K,\mathcal{P},Q) \neq \emptyset\}$$

$$\varepsilon^{\leftrightarrows}(K,\mathcal{P},Q) :\Leftrightarrow \{L \mid L \in \mathcal{P} \wedge K \neq L \wedge \exists M \in Q : \exists \hat{K} \subset K : \big(\hat{K} \cap M \neq \emptyset \wedge L \cap M \neq \emptyset \wedge \hat{K} \in 2^{\mathcal{P} \cap Q} \wedge \big(\exists \hat{L} \subseteq L : \hat{L} \cap M = \emptyset \wedge \hat{L} \in 2^{\mathcal{P} \cap Q} \wedge s_{min} \leq |(K \setminus \hat{K}) \cup \hat{L}| \leq s_{max} \wedge s_{min} \leq |(L \setminus \hat{L}) \cup \hat{K}| \leq s_{max}\big)\big)\}$$

Note that $\hat{K} \cap M \neq \emptyset \wedge L \cap M \neq \emptyset$ implies that K as well as L contain elements that are contained in the same partition $M \in Q$ and should be brought together by the exchange operation. Also note that $\hat{L} \subseteq L$ might be an empty set $\emptyset$, whereas $\hat{K} \subset K$ is always non-empty.

On the one hand, $\hat{K} \subset K$ and $\hat{K} \in 2^{\mathcal{P} \cap Q}$ ($2^{\mathcal{P} \cap Q}$ denotes the power set of $\mathcal{P} \cap Q$) ensure that we increase the similarity of $\mathcal{P}$ and Q when integrating $\hat{K}$ into L. That is because we leave $|\mathcal{P}|$ and $|Q|$ unchanged and reduce $|\mathcal{P} \cap Q|$ by ≥ 1. On the other hand, since $\hat{L}$ represents the elements that should be integrated back into K, $\hat{L} \cap M = \emptyset$ ensures that we merge elements of M. Further, $\hat{L} \in 2^{\mathcal{P} \cap Q}$ ensures that we do **not** spread a set of elements $V \in (\mathcal{P} \cap Q)$ (V is thus contained in a single partition in $\mathcal{P}$ and Q) over K and L by merging $\hat{L}$ into K. This has to be avoided because it would decrease the similarity of $\mathcal{P}$ and Q. The conditions $s_{min} \leq |(K \setminus \hat{K}) \cup \hat{L}| \leq s_{max}$ and $s_{min} \leq |(L \setminus \hat{L}) \cup \hat{K}| \leq s_{max}$ restrict the size of the resulting partitions to the allowed range.

For the resulting partition, we have $\mathcal{P}' = (\mathcal{P} \setminus \{K,L\}) \cup \{(K \setminus \hat{K}) \cup \hat{L}, (L \setminus \hat{L}) \cup \hat{K}\}$. The similarity between $\mathcal{P}$ and Q is increased because $|\mathcal{P}'| = |\mathcal{P}|$, Q is not changed, and $|\mathcal{P}' \cap Q| \leq |\mathcal{P} \cap Q| - 1$. With regard to $\mathcal{P}_*$ and Q_*, e.g., $K_* = \{\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_4, \boldsymbol{g}_5\}$ is an exchangeable partition with exchangeable counterparts $L_* = \{\boldsymbol{g}_3, \boldsymbol{g}_6\}$ and $M_* = \{\boldsymbol{g}_7, \boldsymbol{g}_8\}$. For instance, we can exchange $\hat{K}_* = \{\boldsymbol{g}_4\}$ and $\hat{L}_* = \emptyset$ between K_* and L_* by which we obtain $\mathcal{P}_*{}' = \{\{\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_5\}, \{\boldsymbol{g}_3, \boldsymbol{g}_4, \boldsymbol{g}_6\}, \{\boldsymbol{g}_7, \boldsymbol{g}_8\}\}$.

However, there are situations in which the exchange operator cannot be applied: For example, consider a partitioning $Q = \{N,O,P\}$, where each partition has a cardinality of 100 and $s_{min} = s_{max} = 100$. A partitioning $\mathcal{P} = \{K,L,M\}$ cannot approach Q, e.g., if K contains 39, 27, and 34, L contains 25, 40, and 35, and M contains 36, 33, and 31 elements of N, O, and P, respectively. In such situations, one might relax the constraint $\hat{L} \in 2^{\mathcal{P} \cap Q}$ to $\hat{L} = U \cup V$, where $U \in 2^{\mathcal{P} \cap Q}$ and $V \subset W \in (\mathcal{P} \cap Q)$. While this relaxation allows to apply the operator in each situation, it only guarantees to *not decrease* the similarity of $\mathcal{P}$ and Q because we spread the elements of W over two partitions.

4 EVALUATION

In our evaluation, we analyze PSOPP's performance in various scenarios: We **1)** identify suitable values for $c_{rdm}, c_{\mathcal{B}_i}, c_{\mathcal{B}_{\mathcal{N}_i}}$, **2)** investigate the influence of the numbers of elements $|\mathfrak{G}|$ and particles on PSOPP's performance with regard to the quality of the result and the number of moves particles perform, **3)** examine PSOPP's convergence, **4)** compare its behavior with less and more constrained partitionings on the basis of the partitioning constraints introduced in Sect. 3.1, **5)** make these investigations for clustering as well as anticlustering, and **6)** compare our results to those achieved with IBM ILOG CPLEX[3] and an x-means implementation[4] as, to the best of our knowl-

[3]IBM ILOG CPLEX Optimizer, Version 12.4, 2011: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[4]Weka, Version 3.6: http://weka.sourceforge.net

edge, there is no other renowned algorithm supporting more of our partitioning constraints out of the box.

Where not otherwise stated, we performed 500 simulation runs for each evaluation scenario. All presented results are average values. For evaluation, we used a Java implementation of PSOPP in which each particle runs in its own thread, thereby allowing for the parallel examination of the search space. Because we expected PSOPP to achieve good results with a relatively small number of particles, we used particle neighborhoods $\mathcal{N}_i$ that contain all particles in the system. As mentioned in Sect. 2.1, $\mathcal{B}_{\mathcal{N}_i}$ thus corresponds to $\mathcal{B}$. In each setting, the examined algorithms solved the PP for a set of elements $\mathfrak{G} = \{0, 1, 2, \ldots, n\}$, where $n = |\mathfrak{G}|$ is the number of elements to partition.

All evaluation scenarios were performed for both clustering as well as anticlustering (see Sect. 1). In both cases, the goal is to maximize the fitness. In case of clustering, we therefore use the negative Euclidean distance (this corresponds to the "classic" k-means distance measure) as fitness value. To assess the fitness of a partitioning in case of anticlustering, for each partition, we calculate the mean of the contained elements. As the goal is to have homogeneous clusters of heterogeneous elements, i.e., to equalize these mean values, the standard deviation σ_{part} of the mean values should be minimized. Because we can normalize the elements' values to the interval $[0, 1]$, σ_{part} is between 0 and $\sqrt{0.5}$. We thus define a partitioning's fitness $\mathcal{F}_{anti}(\sigma_{\text{part}})$ in case of anticlustering as follows:

$$\mathcal{F}_{anti}(\sigma_{\text{part}}) = \left(\frac{1}{\sigma_{\text{part}}+1} - \frac{1}{\frac{1}{\sqrt{2}}+1}\right) \cdot \left(1 - \frac{1}{\frac{1}{\sqrt{2}}+1}\right)^{-1}$$

$\mathcal{F}_{anti}(\sigma_{\text{part}})$ monotonically decreases on the interval $[0, \sqrt{0.5}]$. We have $\mathcal{F}_{anti}(0) = 1$ for optimal partitionings where each partition has the same mean value, and $\mathcal{F}_{anti}(\sqrt{0.5}) = 0$ for the maximum σ_{part}. Because we expect PSOPP to perform well, $\mathcal{F}_{anti}(\sigma_{\text{part}})$ is particularly sensitive to changes when σ_{part} is small.

Where not otherwise stated, we used a time limit of 10s as termination criterion. Further, apart from $s_{min} = 2$ (i.e., each partition has to consist of more than one element) and $n_{min} = 2$, which prevents the "grand coalition", we did not restrict valid partitionings (i.e., $s_{max} = n$, $n_{max} = \frac{n}{2}$). As discussed in Sect. 1, such restrictions enable hierarchical decomposition. The influence of restrictions on PSOPP's behavior is examined in a separate scenario.

Identification of Suitable Parameters. First, we identified suitable parameter sets for different numbers of elements $n \in \{100, 500, 1000\}$ and particles $\#P \in \{4, 16\}$ for clustering as well as anticlustering. Values for $c_{rdm}, c_{\mathcal{B}_i}$, and $c_{\mathcal{B}}$ were taken from the set $\{0.0, 0.1, 0.2, \ldots, 1.0\}$. For each combination, we performed 100 simulation runs. The results for clustering with $\#P = 4$ and $n = 1000$ are depicted in Fig. 2. The plateau of moderate to good fitness values for $0.0 < c_{rdm} < 0.5$ indicates the trade-off between exploration and exploitation. In case of clustering, $c_{rdm} = 0.3, c_{\mathcal{B}_i} = 0.0$, and $c_{\mathcal{B}} = 0.7$ turned out to be useful parameters for all combinations of n and $\#P$. For anticlustering, $c_{rdm} = 0.2, c_{\mathcal{B}_i} = 0.7$, and $c_{\mathcal{B}} = 0.1$ are suitable parameters. Consequently, approaching $\mathcal{B}_i$ is far more important for anticlustering than clustering. We assume that the fitness landscape of anticlustering contains more spikes that are worth to be explored in more detail, while clustering requires all particles to work together in order to improve a specific solution candidate (less but more prominent spikes in the fitness landscape). In both cases, the greater n, i.e., the problem to solve, the more important the parametrization. Furthermore, we examined the effect of allowing particles to apply a random move operator if an approach is not possible. In case of clustering, this feature yields to a very high robustness with regard to different values for $c_{rdm}, c_{\mathcal{B}_i}, c_{\mathcal{B}}$: For $\#P = 4$ and $n = 1000$, for instance, the fitness improves by an average of 51.65% (standard deviation $\sigma = 50.76\%$) for $0.0 < c_{rdm} < 0.5$, and by 21.67% for the best parametrization. With regard to anticlustering, we could not observe such an increase in robustness. In contrast, the fitness values slightly decrease by an average of 0.39% ($\sigma = 1.33\%$). We used the identified parameters in our following investigations.

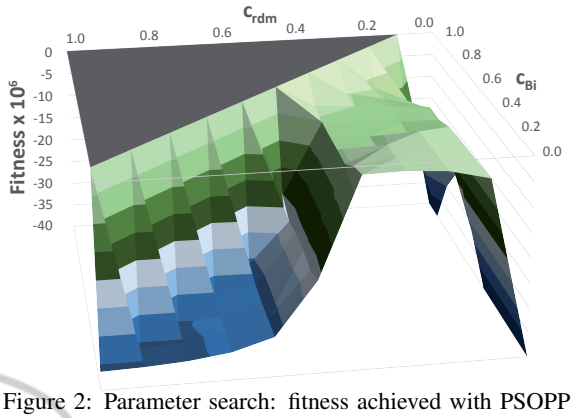

Figure 2: Parameter search: fitness achieved with PSOPP when partitioning 1000 elements using the k-means fitness function and 4 particles with different combinations of $c_{rdm}, c_{\mathcal{B}_i}, c_{\mathcal{B}}$. Results are averaged over 100 runs.

Influence of the Number of Elements to Partition. We evaluated the influence of n for the set of problem sizes $\mathfrak{N} = \{100, 250, 500, 1000, 2000, 3000, 4000\}$. As shown in Table 1, an increase of n comes along with a decrease in the no. of moves particles make in

Table 1: Selected clustering and anticlustering results obtained with PSOPP using a time limit of 10s for different values of the number of elements and particles. All values are averages over 500 runs. Values in parentheses denote standard deviations.

	clustering						anticlustering			
#Elements	250		500		1000		1000		3000	
#Particles	4	32	4	32	4	32	4	32	4	32
Optimal Fitness	-62.50		-125.00		-250.00		1.00		1.00	
Fitness	−78.16 (6.14)	−106.88 (12.23)	−3694.88 (440.71)	−95462.66 (15732.15)	$-1.10\cdot10^6$ ($1.12\cdot10^5$)	$-2.25\cdot10^7$ ($2.26\cdot10^6$)	1.00 (0.00)	1.00 (0.00)	0.96 (0.08)	0.99 (0.03)
#Partitions	121.43 (1.33)	121.81 (1.37)	242.82 (2.43)	243.44 (2.98)	478.67 (7.59)	468.55 (7.78)	2.03 (0.67)	2.01 (0.12)	147.21 (259.87)	24.56 (48.55)
Partition Size	2.06 (0.25)	2.05 (0.22)	2.06 (0.24)	2.05 (0.23)	2.09 (0.29)	2.06 (0.23)	492.61 (322.56)	496.52 (320.91)	20.38 (166.51)	122.17 (402.50)
#Total Moves [in 1000]	690.16 (12.14)	509.69 (8.84)	292.85 (3.69)	176.73 (8.93)	90.22 (2.31)	102.74 (13.60)	212.10 (34.47)	224.29 (20.99)	54.36 (22.67)	63.33 (8.49)
#Rdm. Moves [in 1000]	321.34 (6.28)	219.13 (4.73)	134.09 (1.72)	59.97 (2.47)	38.58 (0.93)	31.37 (3.97)	89.42 (21.20)	113.32 (16.31)	24.94 (10.31)	35.88 (5.12)
#Appr Moves [in 1000]	368.81 (5.90)	290.57 (4.62)	158.76 (2.07)	116.76 (6.56)	51.65 (1.49)	71.37 (9.63)	122.68 (20.21)	110.97 (12.70)	29.43 (13.82)	27.45 (4.41)
#Moves/Particle [in 1000]	172.54 (3.06)	15.93 (0.94)	73.21 (0.99)	5.52 (0.49)	22.56 (0.78)	3.21 (0.65)	53.03 (10.99)	7.01 (6.82)	13.59 (10.58)	1.98 (2.22)

the search space in clustering as well as anticlustering. Evidently, that is because the application of move operators (especially the approach operators) needs more time. Since the size of the search space grows exponentially with n (see Sect. 1), it is therefore not surprising that the achieved fitness drops with greater n (please note that, in case of clustering, the fitness values *cannot* be compared between two runs with different n): While PSOPP obtains good results for $n = 100$ and moderate fitness values for $n = 250$ (only 16.60% and 25.06% worse than the optimal fitness, respectively) in case of clustering, we need a higher time limit (i.e, more than 10s) for $n \geq 500$ (see convergence evaluation). Nevertheless, PSOPP notices that it is a good idea to establish small clusters (of size two in the best case) for all n. In anticlustering, PSOPP scales much better with n. For $\#P = 4$ and $n = 4000$, PSOPP's fitness is still 93.15% ($\sigma = 9.42\%$) of the optimal fitness value. For $n \leq 1000$, PSOPP realizes that a partitioning with two partitions is appropriate and achieves optimal results in all runs ($\sigma = 0.00\%$).

Influence of the Number of Particles. For all $n \in \mathfrak{N}$, we also varied the no. of particles $\#P \in \{2,4,8,16,32\}$ (see Table 1). We observed that the total no. of moves only shows slight increases (if not decreases) for $\#P > 4$ in case of clustering as well as anticlustering (the threshold of 4 can be attributed to our 4-core Xeon machines). In most cases, the coefficient of variation of the no. of moves per particle increases significantly with $\#P > 4$, meaning that some particles made many and others only few moves. This characteristic together with a remarkable drop of the average no. of moves per particle seems to be rather problematic for clustering since the search space has to be explored more systematically, which is reflected in lower fitness values, whereas a greater no. of particles yield better fitness values for $n > 1000$ in case of anticlustering. Summarizing, there is certainly a trade-off between the no. of moves per particle and the provision of diversity through a greater no. of particles that represent and improve different solution candidates. In the following, we use $\#P = 4$ as it yields good results in both cases. Because of our 4-core machines, compared to $\#P = 2$, $\#P = 4$ allows to make 86.15% ($\sigma = 27.06\%$) or 58.18% ($\sigma = 34.67\%$) more moves in total and improves the fitness by 19.99% ($\sigma = 21.46\%$) or 2.47% ($\sigma = 3.21\%$) in case of clustering or anticlustering, respectively.

Influence of Partitioning Constraints. To examine the influence of constrained partitionings on PSOPP's behavior, we additionally used $n_{max} = \frac{n}{2}$, $n_{min} = 0.98 \cdot n_{max}$, $s_{min} = 2$, and $s_{max} = n - (n_{min} - 1) \cdot s_{min}$ for clustering, and $n_{min} = 2$, $n_{max} = n \cdot 0.02$, $s_{min} = \frac{n}{n_{max}}$, and $s_{max} = \frac{n}{n_{min}}$ for anticlustering. These parametrizations are compatible to the average number and size of partitions PSOPP found in the other evaluation scenarios (see Table 1): In clustering, it is preferred to create partitions that contain two very similar elements (e.g., a partition containing i and $i+1$), whereas it is preferred to create two big partitions with similar mean values in anticlustering. We observed that the greater restrictions allow PSOPP to improve the fitness by an average of 16.84% ($\sigma = 4.43\%$) in case of clustering, and 4.10% ($\sigma = 2.84\%$) for $n > 1000$ in case of anticlustering. In anticlustering, the greater restrictions thus allowed PSOPP to find optimal solutions even for $n > 1000$ in all runs. This improvement is accompanied by an average increase of the total no. of moves by 8.77% ($\sigma = 15.24\%$) in clustering, and even 73.90% ($\sigma = 31.26\%$) in anticlustering. This shows that PSOPP cannot only deal with constrained partitionings but also benefits from them. While the former is not to be taken for granted (as outlined in Sect. 1, to the best of our knowledge, there is no partitioning algorithm that supports all of these constraints out of the box), the

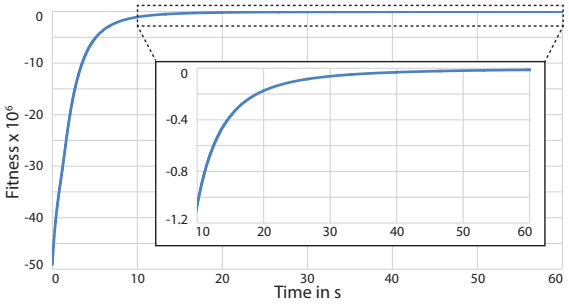

Figure 3: PSOPP's convergence for $n = 1000$, $\#P = 4$, and a time limit of 60s in case of clustering (average of 500 runs).

latter is mainly because the partitioning constraints reduce the size of the search space.

Convergence. For the evaluation of PSOPP's convergence, we ran experiments for additional time limits of 30s and 60s for all $n \in \mathfrak{N}$. Especially clustering benefits from higher time limits in case of $n > 250$: On average, the fitness is 67.97% ($\sigma = 25.99\%$) and 84.92% (16.67%) higher after 30s and 60s, respectively, compared to a limited runtime of 10s. The total no. of moves increases by an average of 191.41% ($\sigma = 88.75\%$) and 512.77% ($\sigma = 261.02\%$), respectively. In anticlustering and $n > 1000$, PSOPP already yields very high-quality results after 10s. The fitness therefore only improves by 3.08% ($\sigma = 1.91\%$) and 3.89% ($\sigma = 2.65\%$) after 30s and 60s, respectively, while the total no. of moves grows by 261.68% ($\sigma = 22.98\%$) and 700.54% ($\sigma = 92.06\%$). Fig. 3 illustrates PSOPP's convergence, i.e., the mean development of $f(\mathcal{B})$, for clustering over a time frame of 60s.

Comparison with CPLEX and x-means. As stated before, we used CPLEX and an x-means implementation for comparison. We evaluated 100 runs for the following scenarios. Regarding CPLEX, we formulated the clustering and anticlustering problems as 0-1 programming problems. For $n = 100$, CPLEX obtains a solution after 30s with an average fitness of -38382.75 and 0.55 that is improved to -23014.93 and 0.63 after 60s in case of clustering and anticlustering, respectively (in all cases, $\sigma = 0.00$). PSOPP already yields far better results after 10s (-29.15 with $\sigma = 1.93$ and 1.00 with $\sigma = 0.00$, respectively). While CPLEX even needs about 720s to calculate a solution for $n = 250$, results for $n = 500$ cannot be obtained since it exceeds our 32GB of available memory. This demonstrates the problem's complexity and the need for metaheuristics. In contrast to PSOPP which solves the PP in a *general manner*, x-means is specialized to a specific problem, i.e., clustering, and is *not* able to solve, e.g., the anticlustering problem. Moreover,

Table 2: The avg. time needed by PSOPP to find 99% / 100% optimal solutions for anticlustering / clustering (optimal fitness: 1.00 / 0.00) in 99% of the 500 runs, and the avg. fitness obtained by x-means after the avg. time PSOPP needed to find the optimum for clustering ($s_{min} = n_{min} = 1$, $s_{max} = n_{max} = n$). Parentheses contain standard deviations.

#Elements	PSOPP anticlustering Runtime in s	PSOPP clustering Runtime in s	x-means clustering Fitness
100	0.032 (0.029)	0.027 (0.019)	-176.44 (516.95)
250	0.14 (0.12)	0.12 (0.078)	-349.78 (477.80)
500	0.57 (0.49)	0.43 (0.29)	-738.00 (491.30)
1000	2.30 (2.25)	1.71 (1.19)	-5040.00 (0.00)
2000	11.88 (13.21)	8.53 (6.15)	-10080.00 (0.00)
3000	34.29 (41.96)	20.31 (15.76)	-34232.00 (0.00)
4000	81.38 (97.29)	45.88 (34.58)	-20160.00 (0.00)

PSOPP allows to restrict valid partition sizes, which is not possible in x-means. Because it is not obvious how to extend x-means by this feature, we used $s_{min} = 1$ to compare PSOPP to x-means. In this situation, we have to admit that the highly specialized x-means performs much better than PSOPP for $n \geq 250$ (with $n \in \mathfrak{N}$). After 10s, PSOPP's fitness is 19.03% higher for $n = 100$. In case of $n = 250$, x-means obtains a fitness of -68.00 ($\sigma = 0.00$), whereas PSOPP yields a fitness of -195.44 ($\sigma = 25.51$). PSOPP achieves a comparable value of -81.38 ($\sigma = 2.70$) after 60s. However, we observed that PSOPP outperforms x-means when we do not constrain valid partitionings at all (here, $s_{min} = n_{min} = 1$ and $s_{max} = n_{max} = n$). Table 2 depicts the average time $\overline{r_n}$ PSOPP needed to find optimal solutions in case of clustering and solutions with a fitness of 99% of the optimum in case of anticlustering for all $n \in \mathfrak{N}$ (please note the cubic growth of $\overline{r_n}$ with n). In contrast to PSOPP, x-means was not able to find optimal solutions for clustering; the fitness values obtained after $\overline{r_n}$ seconds are also depicted in Table 2.

5 CONCLUSION AND FUTURE WORK

In this paper, we introduced PSOPP, a PSO that solves the partitioning problem (PP) outlined in Sect. 1. In contrast to the majority of other approaches, PSOPP solves the PP in a general manner and is thus applicable to diverse problems – comprising strict partitioning clustering (with outliers), anticlustering, and coalition structure generation, among others – in various domains (see Sect. 1 for examples). It can be customized to a specific application by defining an appropriate fitness function that evaluates the quality of solution candidates. Valid partitionings can be specified in terms of a minimum and maximum number and size of partitions. This clearly distinguishes PSOPP from other partitioning methods. To explore the search space, PSOPP uses basic set operations like

split, join, and exchange. Our evaluation shows that PSOPP finds high-quality solutions respecting specified partitioning constraints in different evaluation scenarios with a low number of particles.

In this paper, we assumed that PSOPP partitions a *set* of elements (see Sect. 1). In future work, we will revise the definition of the similarity of partitionings and adjust PSOPP's approach operations so that it can solve multiset partitioning problems. In this context, we want to examine which influence these changes have on PSOPP's performance. Furthermore, we will extend PSOPP with regard to multi-objective optimization to gather solutions lying on a pareto frontier.

ACKNOWLEDGEMENT

This work is partly sponsored by the research unit FOR 1085 of the German Research Foundation.

REFERENCES

Abdallah, S. and Lesser, V. (2004). Organization-Based Cooperative Coalition Formation. *Int. Conference on Intelligent Agent Technology*, pages 162–168.

Al Faruque, M. A., Krist, R., and Henkel, J. (2008). ADAM: run-time agent-based distributed application mapping for on-chip communication. In *Proc. of the 45th annual Design Automation Conf.*, pages 760–765. ACM.

Alam, S., Dobbie, G., and Riddle, P. (2008). An Evolutionary Particle Swarm Optimization Algorithm for Data Clustering. In *IEEE Swarm Intelligence Symposium, 2008*, pages 1–6.

Anders, G., Seebach, H., Nafz, F., Steghöfer, J.-P., and Reif, W. (2011). 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)*, pages 20–31.

Anders, G., Siefert, F., Steghöfer, J.-P., and Reif, W. (2012). A decentralized multi-agent algorithm for the set partitioning problem. In *PRIMA 2012: Principles and Practice of Multi-Agent Systems*, volume 7455 of *Lecture Notes in Computer Science*, pages 107–121. Springer Berlin / Heidelberg.

Apt, K. R. and Witzel, A. (2007). A Generic Approach to Coalition Formation. *Proc. of the Int. Workshop on Computational Social Choice COMSOC*, 11(3).

Äyrämö, S. and Kärkkäinen, T. (2006). Introduction to partitioning-based clustering methods with a robust example. Technical report, Reports of the Department of Mathematical Information Technology, Series C. Software and Computational Engineering of the University of Jyväskylä.

Bender, C., Brody, D., and Meister, B. (1999). Quantum field theory of partitions. *Journal of Mathematical Physics*, 40:3239.

Buccafurri, F., Rosaci, D., Sarnè, G., and Ursino, D. (2002). An Agent-Based Hierarchical Clustering Approach for E-commerce Environments. In *E-Commerce and Web Technologies*, volume 2455 of *LNCS*, pages 109–118. Springer.

Chu, P. and Beasley, J. (1998). Constraint handling in genetic algorithms: The set partitioning problem. *Journal of Heuristics*, 4(4):323–357.

Consoli, S., Moreno-Pérez, J., Darby-Dowman, K., and Mladenović, N. (2010). Discrete Particle Swarm Optimization for the minimum labelling Steiner tree problem. *Natural Computing*, 9(1):29–46.

Di Marzo Serugendo, G., Gleizes, M.-P., and Karageorgos, A. (2005). Self-organization in multi-agent systems. *The Knowledge Engineering Review*, 20:165–189.

Garcia, F. and Perez, J. (2008). Jumping frogs optimization: a new swarm method for discrete optimization. Technical Report 3, Documentos de Trabajo del DEIOC, Department of Statistics, O.R. and Computing, University of La Laguna, Tenerife, Spain.

Horling, B. and Lesser, V. (2004). A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(04):281–316.

Ishioka, T. (2005). An expansion of x-means for automatically determining the optimal number of clusters. In *Proceedings of International Conference on Computational Intelligence*, pages 91–96.

Kennedy, J. and Eberhart, R. (1995). Particle Swarm Optimization. In *Proc. of the IEEE Int. Conf. on Neural Networks, 1995*, volume 4, pages 1942 –1948.

Kennedy, J. and Eberhart, R. (1997). A Discrete Binary Version of the Particle Swarm Algorithm. In *IEEE International Conference on Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation.*, volume 5, pages 4104–4108.

Kudo, Y. and Murai, T. (2009). On a criterion of similarity between partitions based on rough set theory. In *Rough Sets, Fuzzy Sets, Data Mining and Granular Computing*, volume 5908 of *LNCS*, pages 101–108. Springer Berlin / Heidelberg.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations.

Ogston, E., Overeinder, B., Steen, M. V., and Brazier, F. (2003). A Method for Decentralized Clustering in Large Multi-Agent Systems. In *Proc. of the 2nd Int. Joint Conference on Autonomous Agents and Multiagent Systems*, pages 789–796.

Rahwan, T., Ramchurn, S. D., Jennings, N. R., and Giovannucci, A. (2009). An Anytime Algorithm for Optimal Coalition Structure Generation. *Journal of Artificial Intelligence Research*, 34:521–567.

Seren, C. (2011). A Hybrid Jumping Particle Swarm Optimization Method for High Dimensional Unconstrained Discrete Problems. In *2011 IEEE Congress on Evolutionary Computation*, pages 1649–1656.

Shehory, O. and Kraus, S. (1998). Methods for Task Allocation via Agent Coalition Formation. *Artificial Intelligence*, 101(1-2):165–200.

Steghöfer, J.-P., Behrmann, P., Anders, G., Siefert, F., and Reif, W. (2013). HiSPADA: Self-Organising Hierarchies for Large-Scale Multi-Agent Systems. In *Pro-*

ceedings of the Ninth International Conference on Autonomic and Autonomous Systems (ICAS). IARIA.

Valev, V. (1998). Set partition principles revisited. In *Advances in Pattern Recognition*, volume 1451 of *LNCS*, pages 875–881. Springer.

Van der Merwe, D. and Engelbrecht, A. P. (2003). Data clustering using particle swarm optimization. In *The 2003 Congress on Evolutionary Computation*, volume 1, pages 215–220. IEEE.

Younis, O. and Fahmy, S. (2004). HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks. *IEEE Transactions on Mobile Computing*, 3:366–379.