A Generic Set-Based Particle Swarm Optimization Algorithm

Joost Langeveld, Andries P. Engelbrecht

Department of Computer Science, University of Pretoria,
Lynnwood Road, Hatfield, Pretoria, South Africa
jclangev@gmail.com, engel@cs.up.ac.za

Abstract
Several set-based particle swarm optimization algorithms have been proposed in the literature for solving discrete and combinatorial optimization problems. However, a simple but generic algorithm defined in terms of mathematical sets is still missing. In this paper a new algorithm called Set-Based PSO is proposed that fills this gap. The Multidimensional Knapsack Problem is used as a test problem to investigate the performance of the algorithm. Computational experiments are presented on a set of problems known from the literature both for parameter tuning and to compare the algorithm's performance to that of two alternative algorithms.

Key words
Discrete Optimization, Set-based Particle Swarm Optimization, Multidimensional Knapsack Problem

1 Introduction

The goal of this paper is to introduce a new generic set-based particle swarm optimization (PSO) algorithm called Set-Based PSO and demonstrate its viability as a generic optimization algorithm for set-based problems. The Multidimensional Knapsack Problem (MKP) was chosen as the test problem because it allows for straight-forward fitness evaluation of particles without using any domain specific features. Thus the Set-Based PSO can be evaluated, and compared to alternative algorithms, with the quality of the solutions determined by the proposed generic algorithm and not aided by a domain specific operator. It is acknowledged that problem specific algorithms can yield better solutions.

First the problem is stated, followed by a brief overview of the PSO algorithm and various discrete PSO algorithms. Then the Set-Based PSO algorithm is described, which is finally tested in a number of computational experiments and compared to two alternative algorithms.

2 Multidimensional Knapsack Problem

The MKP, also called the multidimensional zero-one knapsack or rucksack problem, is a well-know NP-complete optimization problem [12]. When solving the MKP, the aim is to maximize the total value of all items to be put in a knapsack, while satisfying multiple "weight" constraints. The problem is formulated as

$$\text{maximize} \quad \sum_{i=1}^{n} v_i x_i, \tag{1}$$

$$\text{subject to} \quad x_i \in \{0,1\}, \quad \forall i \in \{1,\dots,n\}, \tag{2}$$

$$\text{and} \quad \sum_{i=1}^{n} w_{i,j} x_i \leq C_j, \quad \forall j \in \{1,\dots,m\}. \tag{3}$$

There are n items in total each with value v_i, while the binary variable x_i indicates whether item i is present in the knapsack or not. The problem has m constraints, where for each constraint j item i has a weight $w_{i,j}$ and for each constraint the total weight $\sum_i w_{i,j}\, x_i$ may not exceed the capacity C_j. Various methods and algorithms have been used to solve the MKP: Integer Linear Programming [22], Polynomial Time Approximation Schemes [13], Genetic Algorithms [3], Ant Colony Optimization [18], and PSO [26].

3 Particle Swarm Optimization

Particle swarms were originally investigated by Kennedy and Eberhart [14] as a tool to model the behavior and movements of a flock of birds. This tool was quickly developed into a simple but elegant optimization algorithm called Particle Swarm Optimization. In the canonical PSO algorithm the search space for the optimization problem is a continuous, multidimensional space. Particles each have a position $\vec{x}$ in the search space and a velocity $\vec{v}$ indicating its direction and step-size. The position is a point in the search space, the velocity is represented by a vector in the same space. Each particle also keeps track of its own fitness (the quality of its solution to the optimization problem) as well as the best position $\vec{y}$ it has visited in the past. Positions are initialized uniformly random over the search space, and are updated by adding the velocity to the current position:

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1) \tag{4}$$

In their paper Kennedy and Eberhart [14] proposed two PSO algorithms, called global best (*gbest*) PSO and local best (*lbest*) PSO. This description will focus solely on the global best PSO variant, for which the velocity $v_{i,j}$ of particle i in dimension j is updated according to the following equation:

$$v_{i,j}(t+1) = v_{i,j}(t) + c_1 r_{1,j}(t)\ [y_{i,j}(t) - x_{i,j}(t)] + c_2 r_{2,j}(t)\ [\hat{y}_j(t) - x_{i,j}(t)], \tag{5}$$

where $j = 1\ldots,n$ are the dimensions of the search space, $r_{1,j}(t)$ and $r_{2,j}(t)$ are random numbers drawn uniformly from $[0,1]$, while c_1 and c_2 are constants. The previous velocity $v_{i,j}(t)$ indicates the momentum of the particle and is also referred to as the inertia component. The cognitive component, $c_1 r_{1,j}(t)\ [y_{i,j}(t) - x_{i,j}(t)]$, has the effect of pulling the particle back towards $\vec{y}_i(t)$; the point in the search space where it experienced the best fitness. The social component, $c_2 r_{2,j}(t)\ [\hat{y}_j(t) - x_{i,j}(t)]$, attracts the particle towards $\vec{\hat{y}}(t)$; the best solution so far.

To improve the performance of the algorithm and better control the balance between exploration and exploitation, additions were made to the basic PSO algorithm: Eberhart et al. [10] proposed velocity clamping which restricts the velocity to a predetermined maximum in each dimension. Later Shi and Eberhart [23] introduced an inertia weight ω to scale the inertia component. The two additions combined result in the canonical velocity update equation for global best PSO:

$$v'_{i,j}(t+1) = \omega\, v_{i,j}(t) + c_1 r_{1,j}(t)\ [y_{i,j}(t) - x_{i,j}(t)] + c_2 r_{2,j}(t)\ [\hat{y}_j(t) - x_{i,j}(t)] \tag{6}$$

$$v_{i,j}(t+1) = \begin{cases} v'_{i,j}(t) & \text{if } |v'_{i,j}(t)| < V_{max,j} \\ V_{max,j} & \text{if } |v'_{i,j}(t)| \geq V_{max,j} \end{cases} \tag{7}$$

4 Discrete PSO approaches

PSO has proven itself to be an efficient algorithm for solving continuous optimization problems. Its early success led to a search to see if the algorithm could be adapted so that it could be applied to discrete and combinatorial optimization problems as well. Kennedy and Eberhart [15] introduced a discrete version of the PSO algorithm referred to as the Binary PSO (BPSO). In BPSO a particle's position is no longer a real-valued vector but a bit-string $\vec{x} = \{0,1\}^n$. A sigmoidal transformation converts the velocity $v_{i,j}$ of particle i in dimension j to a value in $[0,1]$, which is used to determine the value of $v_{i,j}$. The update equations for BPSO become:

$$v_{i,j}(t+1) = \omega\, v_{i,j}(t) + c_1 r_{1,j}(t)\ [y_{i,j}(t) - x_{i,j}(t)] + c_2 r_{2,j}(t)\ [\hat{y}_{i,j}(t) - x_{i,j}(t)] \tag{8}$$

$$S(v_{i,j}(t+1)) = \frac{1}{1+e^{-v_{i,j}(t+1)}} \tag{9}$$

$$x_{i,j}(t+1) = \begin{cases} 1 & \text{if } r_{3,j}(t) < S(v_{i,j}(t+1)), \\ 0 & \text{otherwise,} \end{cases} \tag{10}$$

where $r_{3,j}(t)$ is a random number drawn uniformly from $[0,1]$. Several variants and extensions of BPSO have been developed, see for example [17, 27].

Discrete PSO methods using fuzzy logic have also been proposed by Pang et al. [20] and Du et al. [8]. A different approach was taken by Tasgetiren et al. [24] who used ranking order value (ROV) to convert the continuous PSO to a discrete one. Pang et al. [21] proposed a similar method with a different conversion mechanism.

Clerc [4, 5] formulated a discrete-valued PSO algorithm by redefining the particles, velocities and operators used in PSO. A general mathematical specification is given as well as a specific implementation that is then applied to the Traveling Salesman Problem (TSP). Other PSO algorithms with a similar approach of redefining the operators have been proposed for the TSP [28], the MKP [2], and for topology design for local area networks [16].

Several set-based PSO algorithms have been proposed, but the opinion of this paper is that an efficient and generically applicable discrete PSO algorithm defined in terms of mathematical sets is still missing:

- The algorithm proposed by Correa et al. [7] and the related one by Bock and Hettenhausen [1] have set-like characteristics but both contain problem specific elements. Especially the concept of a so-called *personal likelihood* that is used instead of particle velocity is not universally applicable: each element in a particle's position must be assigned its own partial fitness, which is impossible for many problems such as the MKP.
- Veenhuis [25] proposed a generic, set-based definition of a PSO algorithm. The design of the velocity update equation leads to velocities and positions always increasing in size, an effect called set bloating. To counter

this, a relatively complex clustering mechanism is introduced in the reduction operator that requires a domain specific element distance function. This means the algorithm is no longer truly generic, and in its current form is not applicable to discrete problems like the MKP.

- Neethling and Engelbrecht [19] proposed the SetPSO algorithm for RNA structure prediction that is a set-based algorithm coupled with the application. The algorithm does not provide a mechanism for set-elements in a position X that are in either the personal best position Y or the neighborhood best position $\widehat{Y}$ to be removed, hence limiting the algorithm's exploratory capabilities. Also elements that are not part of the set $X \cup Y \cup \widehat{Y}$ and that are chosen to be added to the position, are chosen randomly, while a more informed choice improves performance.

5 Set-Based Particle Swarm Optimization

This section proposes a novel, generic Set-Based PSO algorithm. The description below assumes that the Set-Based PSO is used in a maximization task. A similar definition for a minimization task is easily derived from this. Let

- $U = \{e_n\}_{n\in N_U}$ be the universe of discourse containing all elements e_n,
- $\mathscr{P}(U)$ be the power set of U, or the set of all subsets of U, with $X \in \mathscr{P}(U) \Leftrightarrow X \subseteq U$.
- f be the fitness function $f : \mathscr{P}(U) \to \mathbb{R}$ to be optimized,
- $\forall$ particles $i \in I$ the position X_i be a set over U, or $X_i \in \mathscr{P}(U)$,
- S be a swarm of particles: $S = \{X_i\}_{i\in I}$, with $|S| = N_I$,
- V_i be the velocity of particle i, which is defined as a mapping $V_i : \mathscr{P}(U) \to \mathscr{P}(U)$ and refined below,
- Y_i be the personal best position for particle i, i.e. $Y_i = X_i(\tau)$ where $\tau = 1,\dots,t$ such that

$$f(Y_i) = f(X_i(\tau)) = \max\{f(X_i(s)\big| s = 1,\dots,t\} \text{ , and} \tag{11}$$

- $\widehat{Y}_i$ be the neighborhood best position for particle i, i.e. $\widehat{Y}_i = Y_j$ for particle j in X_i's neighborhood with personal best position Y_j that maximizes $f(Y_j)$.

The velocity mapping V_i is more narrowly defined as a set of additions and deletions of elements. Such a mapping is denoted using a set of pairs $\{v_{i,j}\}_{i,j} = \{(\pm, e_{n_{i,j}})\}_{i,j}$. Here $e_{n_{i,j}} \in U$ is an element and the preceding "$+$" or "$-$" sign denotes respectively whether the element $e_{n_{i,j}}$ is added to or removed from the position set on which the mapping V_i is applied. Alternatively, a velocity can be seen as an element in $\mathscr{P}(\{+,-\}\times U)$, the power set of the Cartesian product of $\{+,-\}$ and U. Note that this is a more narrow definition, as a mapping $V_i : \mathscr{P}(U) \to \mathscr{P}(U)$ that can not be described using only additions and deletions can be found. Take for example $U = \{0,1\}$, and mapping V such that $V(\emptyset) = \emptyset$ (so V can not contain any additions), $V(U) = U$ (so V can not contain any deletions). If then also $V(\{0\}) = \{1\}$ and $V(\{1\}) = \{0\}$ (so a deletion and an addition are needed in V in each case) this results in a valid mapping $V_i : \mathscr{P}(U) \to \mathscr{P}(U)$ that can not be denoted as a set of additions and deletions.

The following example illustrates the use of the velocity mapping. Assume that a particle with position X and velocity V are given by

$$X = \{a,b\}, \tag{12}$$
$$V = \{(+,c),(-,b),(-,d)\}. \tag{13}$$

Then applying the velocity V to X yields a new position $X' = \{a,c\}$. The pair $(-,d)$ has no impact on set X as there is no element d in X to remove. The velocity update equation for Set-Based PSO is defined as

$$\begin{aligned} V_i(t+1) \;=\; & c_1r_1 \otimes \big(Y_i(t) \ominus X_i(t)\big) \,\oplus\, c_2r_2 \otimes \big(\widehat{Y}_i(t) \ominus X_i(t)\big) \\ & \oplus \big(c_3r_3 \tfrac{1}{|A|} \odot^+ A\big) \\ & \oplus \big(c_4r_4 \odot^- (X_i(t) \cap Y_i(t) \cap \widehat{Y}_i(t))\big), \end{aligned} \tag{14}$$

where $A := U \backslash \big(X_i(t) \cup Y_i(t) \cup \widehat{Y}_i(t)\big)$. The position update equation is defined as

$$X_i(t+1) = X_i(t) \boxplus V_i(t+1). \tag{15}$$

In the remainder of this section definitions are given for the operators used in equations (14) and (15), followed by a description and motivation of the design choices made in constructing the algorithm. A pseudo-code of the Set-Based PSO algorithm applied to a maximization task is given at the end of the section.

5.1 Redefined operators

The operators $(\oplus, \ominus, \otimes, \odot, \boxplus)$ used in equations (14) and (15) have the following definitions:

$\oplus$ This is a mapping $\oplus : \mathscr{P}(\{+,-\}\times U)^2 \to \mathscr{P}(\{+,-\}\times U)$ that takes two velocities and yields a velocity, defined as the simple union of the two sets of operation pairs:

$$V_1 \oplus V_2 := V_1 \cup V_2. \tag{16}$$

$\ominus$ This is a mapping, $\ominus : \mathscr{P}(U)^2 \to \mathscr{P}(\{+,-\} \times U)$, that takes two positions to yield a velocity. The mapping is defined as

$$X_1 \ominus X_2 := \big(\{+\} \times (X_2 \backslash X_1)\big) \cup \big(\{-\} \times (X_1 \backslash X_2)\big), \tag{17}$$

i.e. the union of the product of $\{+\}$ and all elements in X_2 not in X_1 (all such elements are added) with the product of $\{-\}$ and all elements in X_1 not in X_2 (all such elements are removed). This operator yields the velocity V required to get from X_1 to X_2:

$$V = X_1 \ominus X_2 \implies V \text{ applied to } X_1 \text{ yields } X_2. \tag{18}$$

$\otimes$ This operator is defined such that $\alpha \otimes V$ means picking a subset of $\lfloor \alpha \times |V| \rfloor$ elements at random from a velocity V to yield a new velocity. Here $\lfloor x \rfloor$ for $x \in \mathbb{R}^+$ denotes the largest $n \in \mathbb{N}$ for which $x \geq n$. In general, the operator $\otimes$ is defined as a mapping

$$\mathbb{R}_0^+ \times \mathscr{P}(\{+,-\} \times U) \to \mathscr{P}(\{+,-\} \times U), \tag{19}$$

but working with sets that do not allow multiple instances of the same element, it makes sense to restrict this mapping to scalars of at most 1, leading to

$$[0,1] \times \mathscr{P}(\{+,-\} \times U) \to \mathscr{P}(\{+,-\} \times U). \tag{20}$$

Note that $0 \otimes V = \emptyset$ and $1 \otimes V = V$.

$\odot$ This operator is defined such that $\alpha \odot X$ means picking a subset of $\lfloor \alpha \times |X| \rfloor$ elements from a position X, and converting these elements to a set of operation pairs (additions and deletions of the elements selected), yielding a velocity. Following the argument for $\otimes$ above, this can be seen as a mapping

$$[0,1] \times \mathscr{P}(U) \to \mathscr{P}(\{+,-\} \times U), \tag{21}$$

where the product of a scalar and a position is mapped to a velocity.
Note that equation (14) contains two versions of this operator: $\odot^+$ that yields a velocity consisting of only additions and $\odot^-$ of only deletions. Both are described in more detail below.

$\boxplus$ This operator takes a velocity and a position to yield a new position, and is a mapping

$$\boxplus : \mathscr{P}(U) \times \mathscr{P}(\{+,-\} \times U) \to \mathscr{P}(U). \tag{22}$$

It is specified as the action of applying the velocity mapping V to a position X:

$$X \boxplus V := V(X). \tag{23}$$

Two different implementations for the operators $\odot^+$ and $\odot^-$ are proposed. The operator $\odot^+$ is used in selecting elements to add to X_i, while $\odot^-$ is used in selecting elements to remove from X_i. Random selection is used in case of operator $\odot^-$, which can be written as

$$\alpha \odot^- X := \{-\} \times (\alpha \otimes X). \tag{24}$$

Note that operator $\odot^-$ in equation (14) is applied to the set $(X_i(t) \cap Y_i(t) \cap \widehat{Y}_i(t))$, i.e. elements that are present in each of the three positions X_i, Y_i, and $\widehat{Y}_i$. Such elements are candidates for removal from X_i using a separate mechanism under $\odot^-$, as the process of moving closer to either Y_i or $\widehat{Y}_i$ can not remove such elements from X_i.

Marginal fitness information for particle i is used by operator $\odot^+$ to choose which elements from U to add to X_i. The marginal fitness is defined as the fitness of a new particle with its position equal to its current position plus a single element e, i.e. $X_i \cup \{e\}$. A k-tournament selection algorithm, outlined in Algorithm 1, incorporating this marginal fitness is used to select elements from U to add to X_i. This special case of the operator is denoted by $\odot_i^{+,k}$ to indicate that the operator depends on the particle i and parameter k. In algorithm 1 the set to select new elements from is called A, which in equation (14) is defined as $A := U \backslash \big(X_i(t) \cup Y_i(t) \cup \widehat{Y}_i(t)\big)$, i.e. all elements in U not present in either $X_i(t)$, $Y_i(t)$, or $\widehat{Y}_i(t)$. The parameter $R \in [0,\infty)$ indicates how many elements are selected from A: an element e is chosen $\lfloor R \rfloor$ times from A, where each chosen element in turn is the best (it maximizes the fitness of $X_i \cup \{e\}$) in a tournament of k elements that are randomly selected from A. A larger value of R leads to more elements from outside of A being added to the position $X_i(t)$, while a larger value of k means the algorithm is more greedy in selecting which elements to add.

5.2 Velocity update equation

The velocity update equation (14) for Set-Based PSO is similar to the velocity update equation of the original PSO in equation (5) as both contain a social and a cognitive component. The inertia term is dropped because in

Algorithm 1: k-Tournament Selection $\frac{R}{|A|} \odot_i^{+,k} A$

```
Set $V_{temp,i} = \emptyset$;
for $n = 1, \dots, \lfloor R \rfloor$ do
    for $j = 1, \dots, k$ do
        Randomly select $e_j$ from $A$;
        Set $score_j = f(X_i \cup \{e_j\})$;
    end
    Select $m \in \{1, \dots, k\}$ such that $score_m = \max_j \{score_j\}$;
    Set $V_{temp,i} = V_{temp,i} \oplus (\{+\} \times e_m)$;
end
Return $V_{temp,i}$;
```

Set-Based PSO the velocity $V(t+1)$ is rebuilt at each step while all operation pairs in $V(t)$ have been applied to the position update resulting in $X(t)$. Applying any operation pairs from $V(t)$ to $X(t)$ has no impact on the position and can hence be left out.

Two separate terms are added to the velocity update equation in Set-Based PSO compared to the canonical PSO. For the first term note that if an element e is present in position X_i, but also in its cognitive and social best particles Y_i and $\widehat{Y}_i$, then moving towards either of these two best positions can not remove the element e from X_i. To counteract this phenomenon, a mechanism is introduced to add "new" elements to the particle that are not in $Y_i \cup \widehat{Y}_i$ using the term $c_3 r_3 \frac{1}{|A|} \odot^+ A$, where $A := U \setminus \big(X_i(t) \cup Y_i(t) \cup \widehat{Y}_i(t)\big)$.

Similarly, moving towards the cognitive and social best particles Y_i and $\widehat{Y}_i$ can not remove elements that are not in $Y_i \cup \widehat{Y}_i$. In order to allow exploration of subsets in U that do not contain these elements, the following term is added to the velocity update equation: $c_4 r_4 \odot^- (X_i(t) \cap Y_i(t) \cap \widehat{Y}_i(t))$.

5.3 Initialization

Velocities are initialized as empty sets, and positions are initialized as random subsets over U. To be more precise, for each particle i the position is initialized by drawing a random number r uniformly from $[0,1]$. Then the position X_i is calculated as $r \otimes U$. The complete Set-Based PSO algorithm is described in Algorithm 2.

Algorithm 2: Set-Based PSO for maximization task

```
Set $N$ equal to the number of particles in the swarm;
for $i = 1, \dots, N$ do
    Initialize $V_i := \emptyset$;
    Initialize $X_i :=$ random subset of $U$;
end
while stopping condition is false do
    for $i = 1, \dots, N$ do
        if $f(X_i) > f(Y_i)$ then
            $Y_i := X_i$;
        end
        if $\big(f(Y_i) > f(\widehat{Y})\big)$ then
            $\widehat{Y} := Y_i$;
        end
    end
    for $i = 1, \dots, N$ do
        Update $V_i$ according to equation (14);
        Update $X_i$ according to equation (15);
        Calculate fitness $f(X_i)$ for particle $i$;
    end
end
Return best position found $\widehat{Y}$;
```

6 Experimental Results

For the experiments performed in this section various MKP known from the literature were used to test the Set-Based PSO algorithm, namely the 55 test problems described by Chu and Beasley [3]. For each of the 55 problems the optimal solution is known. The problems range in complexity from 6 to 105 variables and 2 to 40 constraints. All problems used are available on-line at the Operations Research library at `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html`.

The MKP is a constrained maximization problem. For the fitness evaluation, if all constraints are satisfied, the fitness is set equal to the sum of the values of the elements in the particle position. If one or more constraints are not satisfied, the fitness is set to minus infinity.

6.1 Parameter Tuning

A subset of 33 out of the 55 test problems was used to tune the parameters of the Set-Based PSO algorithm. In total 6 parameters were tuned: the swarm size N, the weight parameters c_1, c_2, c_3, and c_4 and the tournament size k. The first phase of parameter tuning was performed according to the method of Franken [11]. This entailed using Sobol pseudo-random numbers to generate low-discrepancy sequences that cover the 6-dimensional parameter space. The initial parameter ranges were set at the values in Table 1.

parameter	N	c_1	c_2	c_3	c_4	k
minimum	5	0.0	0.0	0.0	0.0	1
maximum	40	1.0	1.0	5.0	1.0	10

Table 1: Initial parameter ranges used in tuning the Set-Based PSO algorithm.

An initial set of 64 parameter combinations was generated. For each parameter combination the Set-Based PSO algorithm was run 30 times for each of the 33 problems. A score was calculated by taking the average over the 30 runs of the best fitness recorded in each case, and normalizing this average fitness score by dividing it by the known optimum fitness value. Then, using the FluxViz tool developed by Franken [11], a visual exploration was performed using polar coordinates plots. Based on this visual exploration, the ranges of each parameter were adjusted to zoom in on promising regions of the parameter space.

This process was repeated with 4 additional sets of 64 parameter combinations using different Sobol pseudo-random numbers until the algorithm was run for a total of 320 parameter combinations. Each run was terminated once the known optimum was found, or if there had been no progress in the best fitness found for 2500 iterations, or if a total of 5000 iterations had passed. The visual exploration and parameter range adjustment were performed on the total cumulative set of parameter combinations available for each problem, resulting in 64 parameter combinations for the first exploration, and 128, 192, and 256 for later explorations.

Based on the 320 scores per problem, the best 8 parameter combinations were selected and the parameter values averaged for each of the 33 problems separately. This constituted the estimate of a good parameter value for that problem. In order to see if a single average value existed for each of the 6 parameters that worked for all problems, Kruskal-Wallis statistical tests were performed. The null-hypotheses for these tests were that the average parameter value found was the same for each problem. Note that for each of the six parameters a separate null-hypothesis was tested. For parameter c_1 this can be stated as

$$H_0(c_1) := \{\text{avg}_{p_1}(c_1) = \text{avg}_{p_2}(c_1) = \ldots = \text{avg}_{p_{33}}(c_1)\}, \tag{25}$$

where $p_1, \ldots, p_{33}$ denote the 33 tuning problems and avg_p indicates the average over the best 8 out of 320 parameter values for problem p.

The Kruskal-Wallis test was performed on 33 groups of 8 observations, using a confidence level of $\alpha = 5\%$. As the Kruskal-Wallis tests involved making $33 - 1 = 32$ comparison of average parameter values, the confidence level was adjusted using Bonferroni correction to $\alpha = 5\%/32 = 0.16\%$. The p-values for the six tests are given in Table 2.

parameter	N	c_1	c_2	c_3	c_4	k
average	25.8	0.199	0.759	2.383	0.363	4.693
p-value	55.21%	**0.025%**	4.12%	**0.001%**	**0.008%**	98.65%

Table 2: Average of best 8 parameter values over all 33 tuning problems and Kruskal Wallis test p-value. The bold p-values are less than the Bonferroni corrected confidence level of 0.16% and indicate a significant difference between the average parameter values over the set of problems.

The null-hypothesis was rejected for c_1, c_3, and c_4, indicating that the mean parameter value in these cases varied for the different tuning problems. For the parameters N, c_2, and k no statistically significant evidence was found that the average value was different for any of the tuning problems. Hence for the latter three parameters the (rounded) average values from Table 2 were used for the further computational experiments.

From observing the variance of the average parameter values for c_1, c_3, and c_4 over the set of test problems, the hypothesis came that the found parameter value was influenced by the number of variables in the MKP. Using trial and error the group of 33 tuning problems was subdivided into four separate groups based on the number of variables: 0 to 27, 28 to 30, 31 to 50, and 51 to 105. Further Kruskal-Wallis tests on these groups showed that significant variation of the average parameter value only occurred for c_3 in the 28 to 30 variable group. See Table 3 for details.

One problem (mknap2-43) showed a significantly different average value of $c_3 = 3.90$ compared to $c_3 = 2.44$ for the rest of the 28 to 30 variables group. Even for that specific variable choice of $c_3 = 3.90$, however, performance of the Set-Based PSO algorithm on the mknap2-43 problem is relatively poor: none of the runs were able to find the known optimum and the algorithm got stuck in a local optimum. For further experiments the average parameter values of the 28 to 30 variables group *excluding* mknap2-43 were used. Even the 28 to 30 variables group excluding mknap2-43, however, shows a small but statistically significant deviation in the value of c_3 with a p-value of 0.45% versus a Bonferroni corrected confidence level of 0.56%.

# variables			**p-value Kruskal-Wallis test**							**average parameter value**					
min	max	cases	N	c_1	c_2	c_3	c_4	k	α^*	N	c_1	c_2	c_3	c_4	k
0	105	33	55.21%	**0.025%**	4.12%	**0.001%**	**0.008%**	98.65%	0.16%	**25.8**	0.199	**0.759**	2.383	0.363	**4.693**
0	27	6	66.83%	35.97%	9.56%	8.12%	31.83%	97.40%	1.00%	25.6	**0.231**	0.766	**2.475**	**0.473**	4.479
28	30	11	39.53%	16.79%	1.36%	**0.01%**	1.03%	98.92%	0.50%	27.0	0.229	0.763	2.577	0.423	4.625
31	50	8	54.95%	33.26%	99.19%	3.15%	94.89%	18.25%	0.71%	25.7	**0.200**	0.737	**2.429**	**0.336**	4.516
51	105	8	68.01%	38.74%	19.12%	52.53%	56.18%	66.53%	0.71%	24.7	**0.132**	0.771	**2.003**	**0.225**	**5.125**
28	30†	10	32.75%	19.64%	1.69%	**0.45%**	0.95%	98.97%	0.56%	26.9	**0.226**	0.755	**2.444**	**0.432**	4.575
29	29†	1	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	27.8	0.259	0.841	3.898	0.335	5.125

Table 3: Average of best 8 parameter values over four groups of the 33 tuning problems and Kruskal-Wallis test p-value. The bold p-values are less than the Bonferroni corrected confidence level for that group and indicate a significant difference between the average parameter values over the group of problems. The bold faced parameter values were used in further computational experiments. In the bottom two rows, indicated by †, the results are shown for the 28 to 30 variable group excluding problem mknap2-43, and for that single problem separately.

6.2 Experimental Results

The performance of the Set-Based PSO algorithm was compared to that of the Binary PSO (BPSO) algorithm described in section 4 and the SetPSO algorithm of Neethling and Engelbrecht [19]. The parameters for BPSO were chosen based on the work by Eberhart and Shi [9] and Clerc [6] and set to $c_1 = c_2 = 2.0, \omega = 0.721$ and $V_{max} = 6.0$. The swarm size was set to 25, equal to that for Set-Based PSO.

For SetPSO parameter tuning similar to that for the Set-Based PSO was performed, while the swarm size was fixed at 25. Kruskal-Wallis statistical tests with Bonferroni correction showed that for the random add probability P_R a significant difference in the average parameter value existed for at least some of the 33 tuning problems. The set of tuning problems was again by trial and error divided into groups based on the number of variables. For the two groups of 0 to 35 variables and 36 to 105 variables the intra-group averages for P_R did not show any significant differences, and these averages, shown in bold in Table 4, were used in the further computational experiments.

# variables			**p-value Kruskal-Wallis test**					**average parameter value**			
min	max	cases	N	P_I	P_C	P_R	α^*	N	P_I	P_C	P_R
0	105	33	*NA*	89.03%	41.69%	0.05%	0.15%	**25**	**0.644**	**0.207**	0.079
0	35	22	*NA*	73.45%	99.43%	28.29%	0.23%	25	0.613	0.163	**0.102**
36	105	11	*NA*	93.71%	96.67%	34.94%	0.45%	25	0.678	0.254	**0.075**

Table 4: Average of best 8 parameter values over 33 tuning problems and Kruskal-Wallis test p-value for SetPSO.

For all three algorithms 30 independent runs were performed for the 55 test problems and the same stopping conditions were used: a run was terminated once the known optimum was found, or if there had been no progress in the best fitness found for 2500 iterations, or if a total of 5000 iterations had passed.

The results of the comparison of the three algorithms on all the 55 test MKP are given in Table 5. The average errors of the three algorithms on each problem were compared in a pair-wise fashion using the Wilcoxon sum rank statistical test with a confidence level of 5%. If on a problem one algorithm statistically outperformed the other two, the average fitness error for the outperforming algorithm is indicated in bold.

The Set-Based PSO algorithm was able to find the optimum solution in at least one of the 30 runs for 41 out

# Problems			**Success Rate**			**Average Fitness Error**		
File	**Nr.**	**n / m**	**SB-PSO**	**BPSO**	**SetPSO**	**SB-PSO**	**BPSO**	**SetPSO**
mknap1	1	6 / 10	100%	100%	93%	0.00%	0.00%	0.18%
mknap1	2	10 / 10	100%	100%	17%	0.00%	0.00%	4.48%
mknap1	3	15 / 10	100%	100%	7%	0.00%	0.00%	3.19%
mknap1	4	20 / 10	100%	100%	0%	0.00%	0.00%	11.75%
mknap2	44	20 / 10	27%	53%	0%	0.89%	**0.63%**	10.10%
mknap2	41	27 / 4	3%	33%	0%	0.97%	**0.51%**	5.17%
mknap2	3	28 / 2	0%	0%	0%	0.44%	0.46%	10.31%
mknap2	4	28 / 2	100%	100%	0%	0.00%	0.00%	14.16%
mknap2	5	28 / 2	90%	77%	0%	0.08%	0.17%	17.35%
mknap2	6	28 / 2	0%	0%	0%	3.21%	3.21%	13.02%
mknap2	7	28 / 2	97%	100%	0%	0.01%	0.00%	16.55%
mknap2	8	28 / 2	93%	80%	0%	0.02%	0.06%	12.96%
mknap2	47	28 / 4	7%	43%	0%	1.05%	**0.31%**	5.03%
mknap1	5	28 / 10	13%	60%	0%	0.17%	**0.04%**	10.77%
mknap2	43	29 / 2	0%	0%	0%	3.80%	3.76%	8.44%
mknap2	11	30 / 5	97%	90%	0%	0.05%	0.11%	20.22%
mknap2	12	30 / 5	80%	67%	0%	0.04%	0.10%	18.05%
mknap2	13	30 / 5	93%	83%	0%	0.10%	0.21%	18.37%
mknap2	14	30 / 5	100%	100%	0%	0.00%	0.00%	25.51%
mknap2	15	30 / 5	100%	93%	0%	0.00%	0.08%	22.40%
mknap2	42	34 / 4	3%	3%	0%	2.08%	**0.96%**	7.90%
mknap2	48	35 / 4	3%	7%	0%	2.31%	**1.11%**	7.40%
mknap2	46	37 / 30	7%	10%	0%	**1.50%**	2.56%	25.27%
mknap1	6	39 / 5	0%	0%	0%	0.80%	**0.50%**	4.03%
mknap2	16	40 / 5	30%	3%	0%	**0.31%**	1.06%	19.68%
mknap2	17	40 / 5	47%	7%	0%	**0.30%**	1.17%	23.49%
mknap2	18	40 / 5	37%	3%	0%	**0.16%**	0.93%	19.07%
mknap2	19	40 / 5	83%	7%	0%	**0.13%**	1.26%	25.80%
mknap2	45	40 / 30	47%	3%	0%	**1.62%**	8.04%	34.74%
mknap1	7	50 / 5	0%	0%	0%	1.29%	**1.16%**	8.69%
mknap2	20	50 / 5	27%	0%	0%	**0.55%**	4.13%	28.04%
mknap2	21	50 / 5	27%	0%	0%	**0.50%**	6.93%	27.51%
mknap2	22	50 / 5	40%	0%	0%	**0.62%**	4.96%	29.36%
mknap2	23	50 / 5	57%	0%	0%	**0.35%**	4.99%	28.29%
mknap2	24	60 / 5	20%	0%	0%	**0.54%**	8.59%	29.68%
mknap2	25	60 / 5	30%	0%	0%	**0.56%**	8.94%	30.22%
mknap2	26	60 / 5	23%	0%	0%	**0.22%**	6.29%	31.90%
mknap2	27	60 / 5	3%	0%	0%	**0.54%**	3.56%	22.88%
mknap2	1	60 / 30	27%	0%	0%	**0.36%**	7.54%	37.63%
mknap2	2	60 / 30	0%	0%	0%	**0.54%**	2.66%	24.68%
mknap2	28	70 / 5	0%	0%	0%	**1.25%**	5.87%	26.61%
mknap2	29	70 / 5	27%	0%	0%	**0.71%**	12.05%	33.93%
mknap2	30	70 / 5	0%	0%	0%	**0.86%**	8.30%	31.67%
mknap2	31	70 / 5	17%	0%	0%	**0.63%**	9.40%	29.31%
mknap2	32	80 / 5	7%	0%	0%	**0.82%**	13.97%	37.58%
mknap2	33	80 / 5	3%	0%	0%	**1.26%**	15.31%	37.93%
mknap2	34	80 / 5	0%	0%	0%	**1.66%**	7.24%	27.49%
mknap2	35	80 / 5	0%	0%	0%	**1.23%**	9.82%	30.39%
mknap2	36	90 / 5	3%	0%	0%	**1.27%**	17.15%	38.55%
mknap2	37	90 / 5	13%	0%	0%	**0.82%**	16.95%	38.48%
mknap2	38	90 / 5	3%	0%	0%	**1.37%**	18.56%	39.24%
mknap2	39	90 / 5	0%	0%	0%	**1.38%**	17.96%	39.18%
mknap2	40	90 / 5	0%	0%	0%	**1.67%**	9.63%	31.06%
mknap2	9	105 / 2	0%	0%	0%	**9.87%**	12.66%	11.82%
mknap2	10	105 / 2	0%	0%	0%	**6.26%**	22.01%	40.64%
Average			34.2%	25.9%	2.1%	1.04%	5.16%	21.97%

Table 5: Results for all 55 MKP for the Set-Based PSO (SB-PSO), the BPSO, and the SetPSO algorithm. The success rate indicates the percentage of the 30 runs that found the known optimum, the average fitness error is the deviation between the average best fitness found and the known optimum.

of 55 problems, while BPSO was able to do so in 25 cases and SetPSO in only 3 cases. For 8 problems BPSO outperformed the other two algorithms by a statistically significant margin. For 31 out of 55 problems the Set-Based PSO algorithm statistically outperformed the other two algorithms, which included all problems with 60 or more variables. SetPSO did not outperform the other two algorithms on any problem. For Set-Based PSO the average fitness error measured over all 55 problems was only 1.04%, which is substantially better than the results for BPSO and SetPSO of 5.16% and 21.97% respectively.

7 Conclusion

In this paper a new generic set-based PSO algorithm called Set-Based PSO was introduced and tested computationally on the MKP. The following conclusions were drawn:

1. The parameters of Set-Based PSO can be tuned effectively using the method outlined by Franken [11], but no single "best" parameter setting could be found that worked for all 33 test problems considered.
2. Set-Based PSO was able to find good quality (less than 2% deviation from the known optimum) solutions for the tested MKP with up to 90 variables. Note that good quality solutions were found without using problem specific heuristics and despite the fact that a fitness function was chosen that makes the fitness landscape more rugged by assigning a fitness of minus infinity to any infeasible solution.
3. For smaller MKP (up to 30 variables) Set-Based PSO and BPSO performed well, but little difference was found in the quality of solutions as the problems were not sufficiently difficult to yield a significant difference in performance. SetPSO did significantly underperform for this set of problems.
4. For medium-sized MKP (60 - 105 variables), the Set-Based PSO algorithm performed particularly well compared to BPSO and SetPSO, outperforming the other two algorithms by a statistically significant margin.
5. SetPSO performed poorly on all but the least complex MKP.

Further research is needed to investigate if the Set-Based PSO algorithm scales well and can be applied effectively to more complex MKP. Also more study is required to determine if dynamically changing Set-Based PSO algorithm's parameter values while the algorithm is running can help yield better results.

Due to the generic formulation of the Set-Based PSO algorithm, it is applicable to any optimization problem that can be stated as a set optimization problem. Further investigations are needed to see if optimization of such problems using Set-Based PSO is feasible and competitive with existing methods.

Bibliography

[1] Bock, J. and Hettenhausen, J. (2010). Discrete particle swarm optimisation for ontology alignment. *Information Sciences*, In Press, Corrected Proof.

[2] Chen, W.-N., Zhang, J., Chung, H., Zhong, W.-L., Wu, W.-G., and Shi, Y. (2010). A novel set-based particle swarm optimization method for discrete optimization problems. *Evolutionary Computation, IEEE Transactions on*, 14(2):278–300.

[3] Chu, P. and Beasley, J. (1998). A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86. 10.1023/A:1009642405419.

[4] Clerc, M. (2000). Discrete particle swarm optimization illustrated by the traveling salesman problem. Unpublished, available online only at `http://clerc.maurice.free.fr/pso/pso_tsp/Discrete_PSO_TSP.htm` (accessed 2010/10/08).

[5] Clerc, M. (2004). Discrete particle swarm optimization illustrated by the traveling salesman problem. In Onwubolu, G. C. and Babu, B., editors, *New optimization techniques in engineering*, pages 219–239.

[6] Clerc, M. (2006). *Particle Swarm Optimization*. ISTE.

[7] Correa, E., Freitas, A., and Johnson, C. (2006). A new discrete particle swarm optimization algorithm applied to attribute selection in a bioinformatics data set. In *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO-2006*, pages 35–42. ACM Press.

[8] Du, J.-X., Huang, D.-S., Zhang, J., and Wang, X.-F. (2005). Shape matching using fuzzy discrete particle swarm optimization. pages 405–408.

[9] Eberhart, R. C. and Shi, Y. (2001). Particle swarm optimization: developments, applications and resources. In *Congress on Evolutionary Computation*, volume 1, pages 81–86.

[10] Eberhart, R. C., Simpson, P. K., and Dobbins, R. W. (1996). *Computational intelligence PC tools*. AP Professional, Boston.

[11] Franken, N. (2009). Visual exploration of algorithm parameter space. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 389–398.

[12] Gens, G. and Levner, E. (1980). Complexity of approximation algorithms for combinatorial problems: a survey. *SIGACT News*, 12:52–65.

[13] Kellerer, H. (1999). A polynomial time approximation scheme for the multiple knapsack problem. In Hochbaum, D., Jansen, K., Rolim, J., and Sinclair, A., editors, *Randomization, Approximation, and Combinatorial Optimization. Algorithms and Techniques*, volume 1671 of *Lecture Notes in Computer Science*, pages 51–62. Springer Berlin / Heidelberg. 10.1007/978-3-540-48413-4_6.

[14] Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimisation. In *Proceedings of the International Conference on Neural Networks*, pages 1942–1948.

[15] Kennedy, J. and Eberhart, R. C. (1997). A discrete binary version of the particle swarm algorithm. In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics*, pages 4101–4109.

[16] Khan, S. and Engelbrecht, A. P. (2010). A fuzzy particle swarm optimization algorithm for computer communication network topology design. *Applied Intelligence*, pages 1–17. 10.1007/s10489-010-0251-2.

[17] Khanesar, M. A., Teshnehlab, M., and Shoorehdeli, M. A. (2007). A novel binary particle swarm optimization. In *2007 Mediterranean Conference on Control and Automation*.

[18] Kong, M., Tian, P., and Kao, Y. (2008). A new ant colony optimization algorithm for the multidimensional knapsack problem. *Computers & Operations Research*, 35(8):2672–2683. Queues in Practice.

[19] Neethling, C. M. and Engelbrecht, A. P. (2006). Determining RNA secondary structure using set-based particle swarm optimization. In Yen, G. G., Lucas, S. M., Fogel, G., Kendall, G., Salomon, R., Zhang, B.-T., Coello, C. A. C., and Runarsson, T. P., editors, *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 1670–1677. IEEE Press.

[20] Pang, W., Wang, K.-P., Zhou, C.-G., and Dong, L.-J. (2004a). Fuzzy discrete particle swarm optimization for solving traveling salesman problem. pages 796–800.

[21] Pang, W., Wang, K.-P., Zhou, C.-G., Dong, L.-J., Liu, M., Zhang, H.-Y., and Wang, J.-Y. (2004b). Modified particle swarm optimization based on space transformation for solving traveling salesman problem. volume 4, pages 2342–2346 vol.4.

[22] Puchinger, J., Raidl, G. R., and Pferschy, U. (2010). The multidimensional knapsack problem: Structure and algorithms. *INFORMS J. on Computing*, 22:250–265.

[23] Shi, Y. and Eberhart, R. C. (1998). A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 69–73.

[24] Tasgetiren, M. F., Sevkli, M., Liang, Y.-C., and Gencyilmaz, G. (2004). Particle swarm optimization algorithm for single machine total weighted tardiness problem. volume 2, pages 1412–1419 Vol.2.

[25] Veenhuis, C. (2008). A set-based particle swarm optimization method. In Rudolph, G., Jansen, T., Lucas, S., Poloni, C., and Beume, N., editors, *Parallel Problem Solving from Nature – PPSN X*, volume 5199 of *Lecture Notes in Computer Science*, pages 971–980. Springer Berlin / Heidelberg. 10.1007/978-3-540-87700-4_96.

[26] Wang, L., Wang, X., Fu, J., and Zhen, L. (2008). A novel probability binary particle swarm optimization algorithm and its application. *Journal of Software*, 3(9).

[27] Yang, S., Wang, M., and Jiao, L. (2004). A quantum particle swarm optimization. volume 1, pages 320–324 Vol.1.

[28] Zhong, W.-L., Zhang, J., and Chen, W.-N. (2007). A novel discrete particle swarm optimization to solve traveling salesman problem. pages 3283–3287.