1D-Bin Packing with Stochastic Diffusion Search

Imran Khan [1] and **Mohammad Majid al-Rifaie** [2]

Abstract. For the first time, this paper introduces the application of a Swarm Intelligence algorithm, Stochastic Diffusion Search (SDS), to the One-Dimensional Bin Packing Problem (1D-BPP), demonstrating its power in finding diverse and encouraging solutions, rivalling well established meta-heuristic methods. Stochastic Diffusion Search is a multi-agent swarm technique with a strong mathematical framework. SDS utilises the exchange of information through one-to-one communication and its test and diffusion phases to lead to the convergence of solutions to theoretical optimums. The adapted SDS algorithm presented in this work also combines various swarm intelligence and evolutionary features, such as crossover and local search. In BPP, the algorithm is tasked to partition a set of items into a collection of disjoint subsets. The results of applying SDS agents to the 1D-BPP along with a comparison with several other techniques in order to demonstrate the algorithm's ability is also presented.

1 Introduction

The 1D-Bin Packing Problem is a well known NP-Hard problem in which a set of 1D items (lines or numbers) and the bins' capacity are given. The goal is to pack these items into the bins in such a way that minimises the number of bins used and that the number of items do not exceed the bins' maximum capacity. An important reason for tackling the Bin Packing Problem (BPP) is due to its wide range of industrial applications. Such applications include loading trucks with weight capacity constraints and creating file backups in media. The BPP has immense importance in today's world as the BPP can ultimately reduce wastage of items and reduce the time it takes to complete these applications to, thus, reduce cost.

Traditionally, the BPP has been tackled by fast heuristics. Fast heuristics would produce solutions relatively quickly, but the solutions produced would be sub-optimal. However, due to the emergence of swarm intelligence, many swarm techniques have been deployed to address this problem and have achieved optimal results for very small problem instances. These algorithms include Ant Colony Optimisation [12] and Artificial Bee Colony [4]. These swarm algorithms manipulate the intensification and diversification of solutions to produce theoretically optimal results (i.e. exact solutions to the problems).Though, it should be noted that there is no guarantee that such optimal results will be found.

Despite many swarm algorithms undertaking the BPP, Stochastic Diffusion Search has not yet been applied to the BPP. Therefore, in this study, the Stochastic Diffusion Search algorithm is proposed and applied to the 1D-BPP with additional swarm and evolutionary features (e.g. crossover) to enhance the produced solutions. Incepted in 1989, SDS has been inspired by one species of ants (*Leptothorax acervorum*). This inspiration is based on the "tandem calling" mechanism employed by the ants. The "tandem calling" mechanism is a one-to-one communication strategy where a forager ant will return to its nest with food and will recruit one other ant and from this recruitment, the location of the food is publicised [3, 14]. SDS has been applied to many real world problems, these include a Hybrid Stochastic Diffusion Network (HSDN) [7] and pattern recognition [5, 6]. SDS has attracted many researchers due to its strong mathematical framework [3] and its ability to converge to optimal solutions even if the data is noisy [3].

The paper is organised as follows: Section 2 provides a brief history of SDS, section 3 provides further details of the 1D-BPP and some related work on the 1D-BPP. Section 4 explains how SDS has been applied to the 1D-BPP. Section 5 reports on the experimental results. Results obtained by the SDS algorithm are compared against established Swarm and non-Swarm Intelligence algorithms. Section 5 also includes a discussion of the results obtained, as well as the strengths and weaknesses of SDS. Finally, section 6 provides a summary of the work presented and the potential future work on SDS.

2 Stochastic Diffusion Search

In this section, a brief history, the architecture and previous work on the SDS algorithm is introduced. Stochastic Diffusion Search, incepted in 1989 [5, 6], is part of a larger family of Swarm Intelligence algorithms. It is based on the collective intelligence of sentient agents to stochastically diffuse meaningful information to yield and ultimately converge to the optimum. The SDS algorithm has its origins in pattern recognition [6] utilising the behaviour of decentralised thinking and one-to-one communication to obtain the global optimal solution. A wide range of complex real world problems have been successfully tackled by SDS (e.g. HSDN [7], feature location [10] and medical imaging [2]).

Algorithm 1 SDS algorithm

```
Initialise agents
All agents set to '' inactive ''
While(Condition not met)
    Test phase()
        Determine active / inactive agents
    Diffusion phase()
        Exchange information through one−to−one communication
End While
```

2.1 SDS architecture

The SDS algorithm uses a population of agents. Each agent is given an activity, an agent's activity can either be "active" or "inactive".

[1] Corresponding author. Goldsmiths, University of London, UK, emails: ma301ik@gold.ac.uk
[2] Goldsmiths, University of London, UK

These activities determine whether an agent's solution is a successful solution or an unsuccessful solution. Agents that produce successful solutions become "active" and agents that produce unsuccessful solutions become "inactive".

As shown in Algorithm 1, SDS starts by initialising a population of agents. All agents are set to "inactive". Each "inactive" agent then constructs their solution to the problem (construction of solutions depends on the problem being solved); each solution is called a hypothesis (a potential solution to the problem). The agents then transfer to two phases, called a test phase and a diffusion phase.

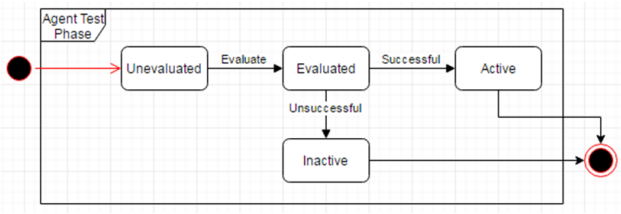

Figure 1. Test phase

The agents firstly enter the test phase. In the test phase, as shown in figure 1, each agent is partially evaluated to determine whether their solution is a successful solution (good solution) or an unsuccessful solution (bad solution). Agents that produce successful solutions become "active" and agents that produce unsuccessful solutions become "inactive". A partial evaluation is when a portion of an agent's solution is evaluated against a model/target. For example, if the problem to solve was to find a specific word (model/target), such as the word "computation", within a text document (search space), and an agent had found the word "amputation". Partial evaluation would be randomly selecting a letter from the agent's solution, for example, the letter "m" and comparing it to the 2nd letter of the target, which would be "o" (comparing the 2nd letter since the letter "m" is the second letter of the agent's solution). If the letter matches the target's letter, then the agent becomes "active", if the letter does not match, then the agent becomes "inactive". Once the evaluation of agents is complete, the agents move to the diffusion phase.

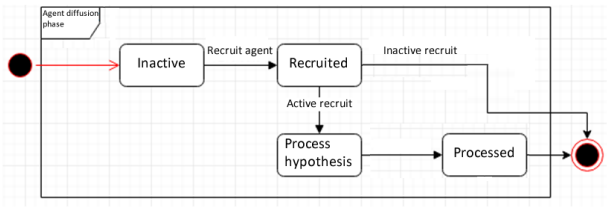

Figure 2. Diffusion phase

In the diffusion phase, as shown in figure 2, the "inactive" and "active" agents undergo different procedures. Each "inactive" agent randomly selects a single agent for potential information diffusion. The selected agent can be referred to as the recruited agent and the former agent as the recruiter agent. If the recruited agent is also "inactive", then the recruiter agent does not proceed to the next stage of the diffusion phase and once the next iteration commences, the "inactive" recruiter agent will once again create a new solution. Though, if the selected agent is "active", then the "inactive" recruiter agent will adopt the "active" recruited agent's solution (i.e. hypothesis).

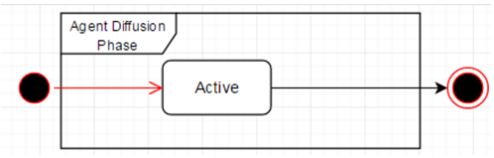

Figure 3. Active agent

As demonstrated in figure 3, each "active" agent, on the other hand, does not select another agent for one-to-one communication. The "active" agents instead keep their solutions. The diffusion phase is finished after this. During the next iteration, the agents that are either "active" or have adopted an "active" agents solution will transfer to the test phase and then the diffusion phase. The agents that are still "inactive" will instead re-construct their solutions then transfer to the test and then the diffusion phase.

These phases continue until a pre-defined condition is met, such as convergence to the global optimal (i.e. finding the target goal to the problem) or after a certain number of iterations.

3 Bin Packing Problem

The BPP, first introduced in 1979 by Garey and Johnson [9], is a highly popular and well researched combinatorial optimisation problem. Its high popularity is due to the wide range of real world applications in which it has been applied to. The BPP has been applied to the cutting and loading industry, the shipping industry in which products must be placed into containers in such a way that allows for more products to be shipped leading to the minimal amount of cost required. It has also been applied to loading trucks with weight capacity constraints, creating file backups in media and resource storage. Hence, with the proper management of space, a reduction of cost associated with the storage of products and transport is possible.

Within the BPP domain there exists the 1D, 2D and 3D-BPP. However, in this paper, the main focus will be on the 1D-BPP. Even though the 1D-BPP lends itself to being the easiest of BPPs due to its dimensionality, it still remains part of the NP-Hard class of problems.

The 1D-BBP can be described as follows: Given a set of items w_i where $i = \{1, 2, ..., n\}$ and each item is of size s and a number of bins b_j where $j = \{1, 2, ..., m\}$ of finite capacity c. Find a packing of items w_i in the bins b_j such that the total sum of a given used bin should be as close to the bin's capacity as possible, i.e. the amount of wasted space in each used bin is minimised and the total number of bins used is minimal. Wasted space means that if the total sum of a bin is 140 and the bins capacity is 150, then there is wasted space (leftover space) of 10. The 1D-BPP must also appease to certain constraints, such as each item is to be packed into only one bin and the total sum of each used bin should not exceed the bin's maximum capacity.

The Mathematical formulation of the Bin Packing Problem can be shown as follows:

$$Minimise : \sum_{j=1}^{m} y_j \tag{1}$$

With the following constraints:

$$\sum_{i=1}^{n} w_i x_{ij} \leq c y_j, \qquad \forall\, j \in \{1, ..., m\}, \tag{2}$$

$$\sum_{j=1}^{m} x_{ij} = 1, \qquad \forall\, i \in \{1, ..., n\}, \tag{3}$$

$$y_j \in \{0, 1\}, \qquad \forall\, j \in \{1, ..., m\}, \tag{4}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall\, i \in \{1, ..., n\}, j \in \{1, ..., m\} \tag{5}$$

where ij indicate the index of the items and bins respectively, w_i represents the weight of the item i to be packed and c denotes the bin's capacity.

Equation (1) denotes that the number of bins assigned is to be minimised, which is the objective function. Equation (2) specifies that the total sum of a given bin should not exceed the bin's predefined capacity. Equation (3) proposes that each item to be packed is packed into only one bin. Finally, equations (4) and (5) determine that y_j and x_{ij} are both binary variables.

3.1 Related work

Swarm Intelligence (SI), employed in works on Artificial Intelligence, is often used in optimisation problems due to its strong foundations and ability to construct diverse and useful solutions even in unstructured complex problems. Within the scope of the 1D-BPP, some SI works include a FireFly algorithm [17], inspired by how fireflies operate, it is based on the procedure of hill-climbing where solutions are changed each time fireflies move closer to each other. The results obtained from this algorithm outperforms well established heuristics such as the First-Fit heuristic.

Abdesslem Layeb et al. [11] used a Quantum Inspired Cuckoo Search Algorithm (QICSA) to tackle the 1D-BPP. QICSA integrates the First-Fit heuristic to produce different good random solutions, and QICSA outperformed the First-Fit Decreasing heuristic when tackling benchmark problem instances. A successful approach by Falkenauer's Hybrid Grouping Genetic Algorithm (HGGA) [8] has been applied to the 1D-BBP, where a local search and grouping Genetic Algorithm are combined. Tugrul Bayraktar et al. applied Artificial Bee Colony (ABC) [4], which, enhanced with memory-integration based on a tabu-list, outperforms standard ABC, First-Fit and the Best-Fit algorithms. A Grammatical Evolution based algorithm [16] has also addressed and tackled the 1D-BPP.

The most successful reviewed population based algorithm has been John Levine and Frederick Ducatelle's Hybrid Ant based algorithm [12]. Their algorithm uses a pheromone trail along with a local search procedure to produce optimal packing patterns. Their Hybrid ACO algorithm outperforms Falkenauer's HGGA algorithm and Martello and Toth's Reduction Algorithm (MTP) [13], to produce the best possible results to the benchmark problems they had tackled.

Though, many non-SI techniques have also tackled the BPP, these include Martello and Toth' reduction based algorithm [13]. They also presented a lower bounds and a dominance criterion to tackle the 1D-BPP. Their algorithm aims to identify bins that dominate other bins, so they can be fixed, resulting in a reduction of the size of the problem.

A hybrid solution procedure, called BISON, was proposed by Scholl et al[15]. Bison combined a tabu search and a branch and bound procedure based on known and new bound arguments and a new branching scheme. The results produced by Bison were very effective and outperfomed an MTP integration procedure.

4 Applying SDS to the 1D-BPP

This section describes how the SDS algorithm is applied to the 1D-BPP. Section 4.1 explains which heuristic is used and how the fitness function is applied. Section 4.2 explains the initial phase of SDS. Section 4.3 describes the test phase and section 4.4 explains how the diffusion phase is implemented and which operations were applied. Finally, section 4.5 describes how local search is applied.

4.1 Heuristic and fitness function

The SDS algorithm uses a heuristic function to construct the solutions. Though, the correct (best) choice of heuristic is needed. The best choice in heuristic will guide the algorithm to produce better solutions. Nurul Afza Hashim et al. [1] found that the First-Fit Decreasing (FFD) heuristic produced the best results in the least amount of processing time when applied to 8 different problem instances compared to other heuristics, such as First-Fit, Best-Fit, etc. Therefore, FFD is chosen as the heuristic for the SDS algorithm. The FFD heuristic sorts the items to pack in descending order and packs each item into the first bin that has enough room to accomodate the item.

A fitness function is also required in order to assess the quality of solutions produced. In the proposed fitness function, each agent is given a fitness value based on how densely packed the used bins in their solution is. To obtain a fitness value for an agent, the contents in each used bin of an agent is summed. If the summed total is equal to the bin's capacity, then the fitness value is incremented by one. This process is conducted for all used bins of an agent. Once the fitness value has been obtained, the fitness value is then divided by the total number of bins the agent has used, to obtain the final fitness value.

For example, the capacity of each bin is 150 and an agent has used 5 bins to construct its solution, and the sum total of each used bin is 150, 140, 150, 150 and 120, respectively. Then the initial fitness value of the agent would be 3, as 3 bins have a sum total of 150. To obtain the final fitness value for the agent, the value of 3 needs to be divided by the total number of bins used by the agent, which is 5. Therefore, the fitness value for the agent is 0.6.

The fitness equations are presented below.

Single bin of an agent:

$$f(b_j) = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} w_i x_{ij} = C \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

Overall fitness of an agent:

$$S_f = \frac{\sum_{j=1}^{m} f(b_j)}{m} \tag{7}$$

In equation 6, b_j denotes a single bin (bin j) of an agent and C denotes the bin's capacity; $f(b_j)$, which is the fitness of bin j, is equal to 1 only when the sum of items in b_j is equal to the capacity of the bin, otherwise the fitness is 0. In equation 7, S_f denotes an agent's solution fitness, which is the sum of the fitness values of all the bins of an agent that are equal to the bin's capacity, divided by the total number of bins used by an agent, which is represented by m.

4.2 Initialisation phase

The SDS algorithm has three phases, the Initialisation phase, the test phase and the diffusion phase. In the Initialisation phase, a population of agents are initialised; the amount of agents initialised is described in sub-section 5.1. All agents are initially set to "inactive". Each "inactive" agent has a copy of the items to pack and each agent starts from an arbitrary location within the items. For example, if there are 120 items to pack, an agent could start at the 30th item. Once all agents have selected their starting item, they then begin to pack the items into the bins in an iterative manner (pack the 30th item then the 31st item, etc) using the chosen heuristic to produce their solutions. Constructing solutions in this manner allows for a diverse set of solutions to be produced. Once all agents have constructed their individual solutions, they are then evaluated and given a fitness value.

4.3 Test phase

Within the test phase, every agent's fitness (value) is evaluated against a fitness threshold (value). The fitness threshold is determined in advance. If an agent's fitness value is greater than or equal to the threshold, the agent becomes "active". If the agent's fitness value does not meet the threshold, then the agent becomes "inactive". The fitness threshold determines which solutions produced are capable of later producing optimal solutions (i.e. best possible solutions). The value of the threshold is critical to the production of well-packed solutions, as changing the value of the fitness threshold leads to the construction of sub-optimal solutions.

if agentFitnessSolution $\geq$ fitnessThreshold **then**
  agent = "Active"
else
  agent = "Inactive"
end if

4.4 Diffusion phase

During the diffusion phase, each "inactive" agent randomly selects a single agent for potential diffusion of hypothesis. In this example, the selected agent is referred to as the recruited agent and the former agent as the recruiter agent. If the selected agent is "inactive", then the "inactive" recruiter agent does not proceed to the next stage of the diffusion phase, and the "inactive" recruiter agent will create a new solution during the next iteration of SDS. Though, if the selected (recruited) agent is "active", both the "inactive" recruiter agent and the "active" recruited agent transfer to the crossover mechanism.

The SDS algorithm constructs the crossover mechanism by conducting the following steps:

1. The "inactive" recruiter agent discards its current solution.
2. Two crossing sites are randomly selected from the "active" recruited agent's solution.
3. The bins from within the crossing site are selected.
4. The "inactive" recruiter agent then inherits the selected bins from the "active" recruited agent.
5. Lastly, all other bins not inherited are "destroyed" (removed) and the items are re-packed into new bins using the FFD heuristic and given to the "inactive" recruiter agent's solution.

This crossover mechanism was inspired by Genetic Algorithms (GA) [8]. The purpose of implementing crossover is to allow the "inactive" recruiter agent to inherit a select number of well-filled bins from the "active" recruited agent. However, given the stochasticity of the crossover procedure, the process of hill-climbing is also implemented during the diffusion phase. During the diffusion phase, the crossover operation is repeated n number of times between the "inactive" recruiter agent and the "active" recruited agent. If the quality of the newly produced solution for the "inactive" recruiter agent is better than its current best solution, then the new solution is accepted, though if the new solution provides no new improvements, then it is ignored and the process continues until the pre-defined iteration end is satisfied.

4.5 Local search

Local search was implemented to the SDS algorithm due to the benefits it provides, such as exploitation of solutions. The performance of meta-heuristics are greatly improved when coupled with local search, as observed by John Levine and Frederick Ducatelle [12]. The SDS local search is applied once all agents become "active".

The local search procedure is described as follows: for a given "active" agent, sum up the contents of each bin, if the total sum of the bin is equal to the bin's capacity, then keep the bin and its contents (i.e. do not "destroy" the bin). All of the bins that do not equal the bins capacity are "destroyed" (removed) and the items of the "destroyed" bins are re-packed into new bins and given to the agent. Though, the items are not re-packed using the FFD heuristic. The items are packed by getting the biggest item that can still fit into the bin. The local search procedure is applied n number of times.

This local search places emphasis on well-filled bins, as algorithms that minimise the leftover space per bin often produce optimal solutions (best solutions) as proved by the ACO [12] and HGGA [8] algorithms.

5 Experiments and Results

The following are described in this section: First, in section 5.1, the parameter values required for the SDS algorithm is described. Then in 5.2, the benchmark problem instances are introduced. In 5.3, the SDS algorithm is compared against different population and non-population based techniques, including the HGGA [8], Hybrid ACO and pure ACO approaches [12], to name a few. The results obtained from this comparison are also shown. 5.4 provides a discussion of the results and 5.5 shows the strengths and weaknesses of SDS.

5.1 SDS parameters

Before commencing with comparing SDS to other algorithms, certain parameters need to be initialised (i.e. iteration numbers, number of agents initialised, etc). The first parameter that needs to be set is the number of agents used for each problem. The number of agents used will be equal to the number of items in each problem. So if there are 120 items in a problem, then there will be 120 agents deployed to tackle the problem. The fitness threshold value is also initialised, and set to the empirically suggested value of 0.6. The iteration number for the crossover and local search mechanisms are the final parameters that need to be set. A total of 80 iterations are set for these mechanisms. These values were empirically suggested following the experiments on different benchmark problems.

5.2 Benchmark instances

The problem instances used and shown in the results tables were obtained from the OR-library[3]. There are 4 problem instance sets (u120, u250, u500 and u1000); each problem instance set contains 20 problems and there are 80 problems overall. u120, u250, u500 and u1000 require 120, 250, 500 and 1000 items to be packed per problem. The size of each item is uniformly distributed in the range of (20,100) and must be packed into bins of capacity 150. Theoretically optimal bins are also provided. Theoretically optimal bins are the calculated best number of bins for a problem. This is calculated by summing up the items values and dividing it by the bins capacity. Theo = (totsize/binsize), this was calculated by Emanuel Falkenauer [8].

5.3 Results

To conduct the experiments, the SDS algorithm is compared against a number of algorithms, ranging from swarm to non-swarm algorithms. The SDS algorithm is compared against algorithms including Falkenauer's HGGA algorithm [8], John Levine and Frederick Ducatelle's Hybrid ACO and pure ACO approaches [12], Martello and Toth's MTP algorithm [13], as well as the Grammatical Evolution based algorithms [16].

Two separate tables will be shown. At the last column of each table, the performance of the SDS algorithm is demonstrated. In the reported result tables, PSO stands for Particle Swarm Optimisation and PESO stands for Particle Evolutionary Swarm Optimisation. The results in Table 1(a) are borrowed from [12] and the results reported in Table 1(b) are from [16].

As explained in [12], in table 1(a), HACO, ACO, MTP and HGGA were run once per problem, therefore, SDS was also run once per problem. The population size for ACO is equal to the number of items per problem. HACO simply uses a population of 10. HGGA uses a population size of 100. The population size for SDS is explained in section 5.1.

The authors of PSO and PESO in table 1(b) had run their algorithms 33 times and the median from their results were used. The population size for each PSO and PESO algorithms is 50.

Within each table, the "prob" column refers to the type of problem instance set the algorithms are tackling. The "+ bins" column refers to how many extra bins were needed for all of the problems in that particular problem instance set. So the optimal bins for u120 are 981 and if an algorithm has a +2 in their column for the u120 problem set, this means that the algorithm used 983 bins in total. The "Exec time" column refers to the execution time of the algorithms and it is in seconds, and only the algorithms in Table 1(a) had provided the execution time, the reason for this is that Levine and Ducatelle [12] were the only ones that provided the execution time for their algorithm and the algorithms they compared. The optimal bins for u120, u250, u500 and u1000 are 981, 2032, 4024 and 8011, respectively. Optimal bins refers to how many bins are needed for the best results; these optimal bins, as described in section 5.2, were calculated by Falkenauer [8].

5.4 Discussion

As observed from the table of results, the SDS algorithm outperforms many well established algorithms. A reason for the SDS algorithm performing superior to many algorithms is perhaps attributable to the diverse solutions produced by the SDS population.

Though, when compared to superior packing algorithms (i.e. HACO and HGGA), the SDS algorithm is outperformed. These algorithms (HACO and HGGA) use the benefits of Genetic Algorithms and Ant Colony and they improve these benefits by applying extra features such as local search, inversion of bins, etc. Acknowledging the outperformance of these two (out of 10) algorithms, including the variants, the purpose of the study is to present a proof of principle to the applicability of a simple algorithm to compete and outperform several other algorithms (8 out of 10, including the variants).

Also, the fitness function and the threshold used to decide if an agent is active or not is highly sensitive to the granularity of item weights. They work well for the problem instance sets used, because the capacity C=150 for all of problems and it is relatively easy to fill up a bin completely with item weights between 25 and 100. They may not work well if the capacity C=100,000 and items weights are between 20,000 and 35,000. In these instances, very few bins may be full, and thus all agents may be inactive.

5.5 Strengths and Weaknesses

SDS has many strengths, one of which is the diffusion phase which allows good solutions to spread throughout the agent population quickly and efficiently, while the discarding of bad solutions is generally reliable. Although SDS exhibited an overall outperformance over the majority of the algorithms reported in this work, it failed to outperform two of the algorithms, namely HACO and HGGA. A possible cause for this could be sought in the balance between exploration and exploitation which is a topic for future research. SDS does offer two mechanisms to cater for this balance (i.e. context-free and context-sensitive mechanism) which will be explored in future research.

The current implementation of SDS [4], in addition to using the inherent exploration-exploitation balance which is facilitated through the test and diffusion phases, the local search allows for further investigation of solutions.

One of the current weaknesses of the fitness function is that a solution can have non-zero fitness only if at least one bin is exactly filled, which despite being feasible in some instances, could be hard or impossible in others. Overcoming this weakness in the fitness function could potentially contribute to a better performance of the algorithm.

6 Conclusion and future work

In the work presented, an SDS algorithm was proposed and applied to the 1D-BPP. The purpose of the study was to introduce a proof of principle for the applicability of this simple swarm intelligence algorithm when compared with other swarm intelligence and evolutionary computation algorithms over standard benchmark problem instances.

The presented algorithm produced encouraging results, as shown in section 5.3 where SDS is demonstrated to outperform 80% of the algorithms and their variants: ACO, MTP, as well as the variants of PESO and PSO algorithms over the benchmarks. Having stating the above, SDS algorithm falls short when compared to two algorithm, the Hybrid ACO and HGGA.

In summary, while this work introduces promising results, there are several open research questions which could contribute to further enhancement of the algorithm's performance, such as: introducing

[3] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/binpackinfo.html

[4] The source code for applying SDS to 1D-BPP can be accessed through the following link: https://www.dropbox.com/sh/odi2twj9336xic6/AABaWduBqi8vHbvS_6Reb0pLa?dl=0.

Table 1. Comparing SDS with other techniques.

(a)

Prob	HGGA +Bins	MTP +Bins	ACO +Binss	HACO +Bins	SDS +Bins
u120	+2 1	+2	+2	**+0**	+2
u250	+3	+12	+12	**+2**	+8
u500	+0	+44	+42	**+0**	+7
u1000	+0	+78	+70	**+0**	+12

(b) Grammatical Evolutionary algorithms using Grammars 1-3 (i.e. G1, G2 and G3) + SDS

Prob	PSO			PESO			
	G1 +Bins	G2 +Bins	G3 +Bins	G1 +Bins	G2 +Bins	G3 +Binss	SDS +Bins
u120	+63	+150	+391	+63	+150	+14	**+2**
u250	+130	+310	+819	+130	+310	+30	**+8**
u500	+231	+590	+1623	+231	+590	+54	**+7**
u1000	+419	+1143	+3242	+419	+1143	+97	**+12**

more lenient fitness function which reflects the fitness of a solution more proportionately to their quality vs a binary evaluation of bins, possibly it could be helpful to evaluate a cost function contingent on the relative fullness of each bin. Improving the balance between exploration and exploitation using alternative recruitment strategies in the diffusion phase, exploring the performance of varying local search mechanisms and applying the algorithm to other more complex benchmarks an comparing with other emerging BPP algorithms.

Furthermore, the performance of SDS in the BPP with higher dimensionality (i.e. the 2D and 3D-BPPs) will be evaluated, and potentially SDS could be applied to other NP-Hard class of problems.

References

[1] N Afza, F Zulkipli, S Sarah, and S Radiah, 'An alternative heuristics for bin packing problem', in *Proceedings of the 2014 International Conference on Industrial Engineering and Operations Management*, pp. 7–9, (2014).

[2] Mohammad Majid al-Rifaie, Ahmed Aber, Robert Sayers, Edward Choke, and Mathew Bown, 'Deploying swarm intelligence in medical imaging', in *Bioinformatics and Biomedicine (BIBM), 2014 IEEE International Conference on*, pp. 14–21. IEEE, (2014).

[3] Mohammad Majid al-Rifaie and John Mark Bishop, 'Stochastic diffusion search review', *Paladyn, Journal of Behavioral Robotics*, **4**(3), 155–173, (2013).

[4] Tugrul Bayraktar, Mehmet Emin Aydin, and Muharrem Dugenci, 'A memory-integrated artificial bee algorithm for 1-d bin packing problems'.

[5] J Bishop, *Anarchic techniques for pattern classification.*, Ph.D. dissertation, University of Reading, 1989.

[6] JM Bishop, 'Stochastic searching networks', in *Proc. 1st IEE Conf. on Artifical neural networks*, pp. 329–331, (1989).

[7] JM Bishop and Phil Torr, 'The stochastic search network', in *Neural networks for vision, speech and natural language*, 370–387, Springer, (1992).

[8] Emanuel Falkenauer, 'A hybrid grouping genetic algorithm for bin packing', *Journal of heuristics*, **2**(1), 5–30, (1996).

[9] Michael R Garey and David S Johnson, 'A guide to the theory of np-completeness', *WH Freemann, New York*, (1979).

[10] E Grech-Cini, 'Locating facial features', *Phd, University of Reading*, (1995).

[11] Abdesslem Layeb and Seriel Rayene Boussalia, 'A novel quantum inspired cuckoo search algorithm for bin packing problem', *International Journal of Information Technology and Computer Science (IJITCS)*, **4**(5), 58, (2012).

[12] John Levine and Frederick Ducatelle, 'Ant colony optimization and local search for bin packing and cutting stock problems', *Journal of the Operational Research Society*, 705–716, (2004).

[13] Silvano Martello and Paolo Toth, 'Lower bounds and reduction procedures for the bin packing problem', *Discrete applied mathematics*, **28**(1), 59–70, (1990).

[14] M Möglich, U Maschwitz, and B Hölldobler, 'Tandem calling: a new kind of signal in ant communication', *Science*, **186**(4168), 1046–1047, (1974).

[15] Armin Scholl, Robert Klein, and Christian Jürgens, 'Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem', *Computers & Operations Research*, **24**(7), 627–645, (1997).

[16] Marco Aurelio Sotelo-Figueroa, Héctor José Puga Soberanes, Juan Martín Carpio, Hector J Fraire Huacuja, Laura Cruz Reyes, and Jorge Alberto Soria-Alcaraz, 'Improving the bin packing heuristic through grammatical evolution based on swarm intelligence', *Mathematical Problems in Engineering*, **2014**, (2014).

[17] R Yesodha and T Amudha, *International Journal of Emerging Technologies in Computational and Applied Sciences (IJETCAS)*, **2**(1), 63–67, (2012).