DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

COALITION OF METAHEURISTICS THROUGH PARALLEL COMPUTING FOR SOLVING COMPLEX OPTIMIZATION PROBLEMS

by
Mümin Emre ŞENOL

July, 2022
İZMİR

COALITION OF METAHEURISTICS THROUGH PARALLEL COMPUTING FOR SOLVING COMPLEX OPTIMIZATION PROBLEMS

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Industrial Engineering, Industrial Engineering Program

by
Mümin Emre ŞENOL

July, 2022
İZMİR

Ph.D. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"COALITION OF METAHEURISTICS THROUGH PARALLEL COMPUTING FOR SOLVING COMPLEX OPTIMIZATION PROBLEMS"** completed by **MÜMİN EMRE ŞENOL** under supervision of **PROF. DR. ADIL BAYKASOĞLU** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

....

Prof. Dr. Adil BAYKASOĞLU

Supervisor

Prof.Dr.Latif SALUM	Assist.Prof.Dr. Kökten Ulaş BİRANT
Thesis Committee Member	Thesis Committee Member

Prof.Dr.Yiğit KAZANÇOĞLU	Prof.Dr.Gözde ULUTAGAY
Examining Committee Member	Examining Committee Member

Prof. Dr. Okan FISTIKOĞLU
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my advisor Prof. Dr. Adil BAYKASOĞLU for his continuous support and encouragement. Thanks to his guidance and immense knowledge, I overcame the challenges, and I was able to write this PhD. thesis.

I would like to present thanks to The Scientific and Technological Research Council of Turkey (TUBITAK - BIDEB) for funding this thesis within 2211A National Scholarship Programme for PhD Students.

I would like to offer thanks to my wife Merve ŞENOL and my daughter Ayşe ŞENOL for their understanding, and eternal support in my life. Lastly, I would like to express my special thanks to my parents Fatma ŞENOL, Bahaattin ŞENOL, and my sister Merve ŞENOL for their endless love and support in my whole life.

Mümin Emre ŞENOL

COALITION OF METAHEURISTICS THROUGH PARALLEL COMPUTING FOR SOLVING COMPLEX OPTIMIZATION PROBLEMS

ABSTRACT

Most of the real-life problems could be modeled as optimization problems and the need for effective solution of these problems is always in demand. In this context, efforts to develop effective approaches to solving optimization problems have been the subject of considerable research. These approaches usually combine general-purpose or particular rules in a logical scope and the methods that are formed by the logical combination of these rules are called optimization algorithms. In this study several metaheuristic algorithms are brought together to form a coalition under Weighted Superposition Attraction-Repulsion Algorithm (WSAR) in a parallel computing environment for solving complex optimization problems. The proposed approach runs different single solution based metaheuristic algorithms (SSBMAs) in parallel and employs WSAR (which a recently developed recently proposed swarm intelligence based optimizer) as controller. While SSBMAs are responsible for exploring the search space, WSAR controls the communication process between the SSBMAs. The presented method tested against some well-known complex optimization problems in three groups, namely, continuous optimization problems, binary optimization problems and combinatorial optimization problems. While CEC 2020 problems are selected as test problems for continuous optimization problems, the uncapacitated facility location problem (UFLP) and the set union knapsack problem (SUKP) are selected as test case for binary optimization problems. In addition, the Resource Constrained Project Scheduling Problem (RCPSP) and the Permutation Flow Shop Scheduling Problem (PFSP) are selected as test problems for combinatorial optimization. The obtained results are compared with some other optimization algorithms. The results of the comparison show that the proposed approach is competitive in terms of solution quality and solution time.

Keywords: Metaheuristic algorithms, parallel computing, optimization problems, coalition

KARMAŞIK OPTİMİZASYON PROBLEMLERİNİN ÇÖZÜMÜ İÇİN METASEZGİSEL ALGORİTMALARIN PARALEL HESAPLAMA YOLUYLA KOALİSYONU

ÖZ

Gerçek hayat problemlerinin çoğu optimizasyon problemleri olarak modellenebilir ve bu problemlerin etkin çözümüne olan ihtiyaç her zaman talep görmektedir. Bu bağlamda, optimizasyon problemlerinin çözümüne yönelik etkili yaklaşımlar geliştirme çabaları önemli araştırmaların konusu olmuştur. Bu yaklaşımlar genellikle genel amaçlı veya belirli kuralları mantıksal bir kapsamda birleştirir ve bu kuralların mantıksal birleşiminden oluşan yöntemlere optimizasyon algoritmaları denir. Bu çalışmada, karmaşık optimizasyon problemlerinin çözümü için çeşitli metasezgisel algoritmalar Ağırlıklı Süperpozisyon Çekim-İtme Algoritması (WSAR) kontrolünde bir paralel hesaplama ortamında bir araya getirilmiştir. Önerilen yaklaşım, paralel olarak farklı tek çözüm tabanlı meta-sezgisel algoritmaları (SSBMA'lar) çalıştırır ve denetleyici olarak WSAR'ı (yakın zamanda geliştirilen bir sürü zekası tabanlı metasezgisel algoritma) kullanır. SSBMA'lar arama alanını keşfetmekten sorumluyken, WSAR SSBMA'lar arasındaki iletişim sürecini kontrol eder. Sunulan yöntem, iyi bilinen bazı karmaşık optimizasyon problemlerine karşı sırasıyla, sürekli optimizasyon problemleri, ikili optimizasyon problemleri ve kombinatoryal optimizasyon problemleri olmak üzere üç grupta test edilmiştir. Sürekli optimizasyon problemleri için test problemleri olarak CEC 2020 problemleri seçilirken, ikili optimizasyon problemleri için test durumu olarak kapasitesiz tesis yerleşim problemi (UFLP) ve küme birleşimli sırt çantası problemi (SUKP) seçilmiştir. Ayrıca, Kaynak Kısıtlı Proje Çizelgeleme Problemi (RCPSP) ve Permütasyon Akış Atölye Çizelgeleme Problemi (PFSP), kombinatoryal optimizasyon için test problemleri olarak seçilmiştir. Elde edilen sonuçlar diğer bazı optimizasyon algoritmaları ile karşılaştırılmıştır. Karşılaştırma sonuçları, önerilen yaklaşımın çözüm kalitesi ve çözüm süresi açısından rekabetçi olduğunu göstermektedir.

Anahtar kelimeler: Metasezgisel algoritmalar, paralel hesaplama, optimizasyon problemleri, koalisyon

CONTENTS

Page

Ph.D. THESIS EXAMINATION RESULT FORM ii

ACKNOWLEDGEMENTS iii

ABSTRACT iv

ÖZ v

LIST OF FIGURES ix

LIST OF TABLES x

CHAPTER 1- INTRODUCTION 1

1.1 Motivation of the Thesis 3

1.2 Outline of the Thesis 5

CHAPTER 2 -LITERATURE REVIEW BASED ON COALITION OF METAHEURISTIC ALGORITHMS AND PARALLEL METAHEURISTIC ALGORITHMS 6

2.1 Literature Review based on Coalition of Metaheuristic Algorithms 6

2.2 Literature Review based on Parallel Metaheuristic Algorithms 7

CHAPTER 3 -p-WSAR ALGORITHM 11

3.1 Search phase 14

3.2 Information sharing phase 14

3.3 Reproduction phase 17

CHAPTER 4-p-WSAR ALGORITHM ON CONTINUOUS OPTIMIZATION PROBLEMS 19

4.1 Test problems and computational results 19

CHAPTER 5-p-WSAR ALGORITHM ON BINARY OPTIMIZATION PROBLEMS 50

5.1 Used benchmarking problems 50

5.1.1 Uncapacitated facility location problem 50

5.1.2 0-1 Knapsack problem 52

5.1.3 Set union knapsack problem 54

5.2 Modifications of p-WSAR for binary problems.. 55

5.2.1 Generating a binary superposition 55

5.2.2 Solution representation and generating neighborhood 57

5.2.3 Constraint handling mechanism 60

5.3 Computational Results 61

5.3.1 Results for UFLP 61

5.3.2 Results for 0-1 KP 64

5.3.3 Results for SUKP... 65

5.4 Concluding remarks 73

CHAPTER 6-p-WSAR ALGORITHM ON COMBINATORIAL OPTIMIZATION PROBLEMS 74

6.1 Used Bench-mark problems 74

6.1.1 Resource Constrained Project Scheduling problem (RCPSP)................. 74

6.1.2 Permutation Flow Shop Scheduling Problem (PFSP) 76

6.2 Modifications of p-WSAR for combinatorial problems.................................. 78

6.3 Experimental Results.. 83

6.3.1 Experimental results of RCPSP 83

6.3.2 Experimental results of PFSP 88

6.4 Conclusions 90

CHAPTER 7- CONCLUSION.. 91

REFERENCES **93**

APPENDICIES…………………………………………………………...**113**

LIST OF FIGURES

Page

Figure 3.1 Main steps of the p-WSAR algorithm 12
Figure 3.2 The simplified flow chart of the p-WSAR algorithm 13
Figure 3.3 Determination of the attractive and repulsive superpositions in p-WSAR for $\tau = 0.8$ (Adapted from Baykasoğlu (2020)) 16
Figure 3.4 Pseudo code for superposition determination 17
Figure 4.1 Boxplot of CEC 5D average results 33
Figure 4.2 Boxplot of CEC 10D average results 34
Figure 4.4 Boxplot of CEC 20D average results 35
Figure 4.5 Boxplot of SSBMAs on CEC 5D average results 47
Figure 4.6 Boxplot of SSBMAs on CEC 10D average results 47
Figure 4.7 Boxplot of SSBMAs on CEC 15D average results 48
Figure 4.8 Boxplot of SSBMAs on CEC 20D average results 48
Figure 5.1 An illustration for UFLP 52
Figure 5.2 An illustration for set union knapsack problem 55
Figure 5.3 Exponentially decreasing a) continuous step length and b) corresponding integer step length for different values of φ 59
Figure 5.4 Movement of the population 59
Figure 5.5 Pseudo code of p-WSAR for binary spaces 60
Figure 6.1 An example RCPSP with solution encoding and decoding 76
Figure 6.2 Determination of the superposition in p-WSAR in combinatorial problems 79
Figure 6.3 Determination of the superposition for permutation type vectors 80
Figure 6.4 Determination of the next moves of the SSBMAs in p-WSAR in combinatorial problems 81
Figure 6.5 LOX crossover 81
Figure 6.6 A simplified overall procedure of p-WSAR 83

LIST OF TABLES

Page

Table 1.1 Main differences between WSA and P-WSAR 4

Table 2.1 Coalition schemes of the studies 10

Table 3.1 Parameter settings of SSBMAs and WSAR 18

Table 4.1 Summary of the CEC2020 dataset 20

Table 4.2 Results of 5D case in CEC2020 dataset 23

Table 4.3 Results of 10D case in CEC2020 dataset 25

Table 4.4 Results of 15D case in CEC2020 dataset 27

Table 4.5 Results on 20D case in CEC2020 dataset 29

Table 4.6 Friedman and Wilcoxon signed-rank test results on 5D case 31

Table 4.7 Friedman and Wilcoxon signed-rank test results on 10D case 31

Table 4.8 Friedman and Wilcoxon signed-rank test results on 15D case 32

Table 4.9 Friedman and Wilcoxon signed-rank test results on 20D case 32

Table 4.10 5D case results of SSBMAs 37

Table 4.11 10D case results of SSBMAs 39

Table 4.12 15D case results of SSBMAs 41

Table 4.13. 20D case results of SSBMAs 43

Table 4.14 Friedman and Wilcoxon signed-rank test results on 5D case (SSBMAs) 45

Table 4.15 Friedman and Wilcoxon signed-rank test results on 10D case (SSBMAs) 45

Table 4.16 Friedman and Wilcoxon signed-rank test results on 15D case (SSBMAs) 46

Table 4.17 Friedman and Wilcoxon signed-rank test results on 20D case (SSBMAs) 46

Table 4.18 Average computational times 49

Table 5.1 A sample population for p-WSAR in binary spaces 56

Table 5.3 Computational results for UFLP 63

Table 5.4 Standard instances for 0-1 KP (Zou et. al., 2011; Bhattacharjee and Sarmah, 2014)... 64

Table 5.5 Friedman and Wilcoxon signed-rank test results on KP 65

Table 5.6 The first type instances results for SUKP 67

Table 5.7 The second type instances results for SUKP 69
Table 5.8 The third type instances results for SUKP 71
Table 6.1 Processing times for a 5 x 5 example problem 77
Table 6.2 Average deviations from optimal solutions for the J30 problem set 85
Table 6.3 Average deviations from CPM based lower bound for the J60 problem set 86
Table 6.4 Average deviations from CPM based lower bound for the J120 problem set.. 87
Table 6.5 Comparison of p-WSAR with some state of the art algorithms and WSAR 88
Table 6.6 Non-parametric test results on Taillard Instances...................................... 89
Table 6.7. Comparison of p-WSAR with SSBMAs.. 89
Table 6.8 Non-parametric test results on Taillard Instances p-WSAR vs. SSBMAs 90

CHAPTER 1
INTRODUCTION

Optimization algorithms are designed to find the best combination of the decision variables for one or more objective functions. Today, optimization algorithms are required to solve optimization problems from all disciplines, whether economics, sciences, or engineering. Researchers applied many optimization algorithms to numerous optimization problems. The efficiency of the optimization algorithms on real life problems are proved in many studies (Preitl et al., 2006), (Paliszkiewicz, 2007), (Zapata et al., 2020), (Precup et al., 2021). Optimization algorithms can be categorized into two groups: Exact and approximate algorithms. Exact algorithms are usually based on evaluating every possible alternative to the problem. Although, they ensure the optimum solution, they require excessive runtime, especially as the size of the problem increases. Lagrangian relaxation-based methods, dynamic programming, linear, integer programming based methods (branch-and-cut, branch and-price, and branch-and-cut-and price) and branch-and-bound (B&B) are in the group of exact algorithms for optimization problems (Laporte et al., 1987). Alternatively, approximate algorithms search through some operators in the domain of the optimization problem and try to find near optimum or optimum solutions in short time. Thanks to this advantage, researchers have preferred approximate algorithms to solve complex optimization problems which cannot be tackled effectively with exact algorithms.

Approximate algorithms could be categorized into two groups as heuristic and metaheuristic algorithms. Heuristic algorithms are developed for a particular optimization problem and mostly depend upon the structure of the studied problem. Alternatively, metaheuristic algorithms can be applied to any optimization problem, irrespective of the type of optimization problem (Baykasoğlu, 2001). In other words, heuristic algorithms have the capacity to solve a particular problem, while metaheuristic algorithms can be adapted to any optimization problem. Furthermore, metaheuristic algorithms may produce more effective solutions than heuristic algorithms by using their problem-independent structure.

In the related literature, researchers employed different metaheuristic algorithms to solve several optimization problems such as Particle Swarm Optimization (PSO) for constrained design optimization problems (Ang et al., 2020), Simulated Annealing (SA) for timetabling problem (Leite et al., 2019), Ant Colony Optimization (ACO) for vehicle routing problem (Li et al., 2019), Genetic Algorithm (GA) for max-cut problem (Kim et al., 2019), Iterated Local Search Algorithm (ILS) for portfolio optimization problem (Kizys et al., 2019), Grey Wolf Optimizer (GWO) for the flow shop scheduling problem (Lu et al., 2019), Tabu Search for assembly line-balancing problem (Yilmaz & Yilmaz, 2020), Weighted Super Position Attraction Algorithm (WSA) for resource constrained project scheduling problem (Baykasoğlu & Şenol, 2019) etc. Although metaheuristic algorithms are commonly used to solve complex optimization problems, there are also several disadvantages in their usage, such as slower speed and early convergence. Researchers should select the most suitable metaheuristic algorithm to solve the associated problem in order to overcome these disadvantages, which is not an easy task. Researchers handle this problem by combining different metaheuristic algorithms as hyper-heuristics. They aim to provide better performance by taking advantage of the superior properties of different metaheuristic algorithms. At this step, the reader may refer the following studies for detailed information about hyper-heuristics (Burke et al., 2013) (Ting et al., 2015). In another way, researchers may overcome this problem by combining different metaheuristic algorithms within a coalitional framework (Baykasoğlu et al., 2019). Baykasoğlu et al. (2019) proposed a metaheuristic algorithm that is named as single seekers society (SSS), which enables cooperation of several metaheuristic algorithms. SSS presented better performance than the algorithms, which are members of the coalition. However, SSS runs algorithms in serial that demands considerable computational time.

On the other hand, parallel computing is a sort of computer architecture in which a single application or calculation is executed or processed by several processors at the same time. By spreading the burden across numerous processors, all of which are operating at the same time, parallel computing allows us to do enormous computations. The majority of supercomputers operate on parallel computing concepts. Parallel processing is another name for parallel computing. Parallel computing can shorten our solution time, improve our application's energy economy, and allow us to tackle more complex problems (Almasi & Gottlieb, 1994) (Kohli & Krishnamurti, 1989).

1.1 Motivation of the Thesis

No Free Lunch Theorem that was proposed by Macready and Wolpert claims that no optimization algorithm performs better than all remaining optimization algorithms for all optimization problems, and if we try solving all optimization problems, there is no statistical difference between the performances of different metaheuristics (Wolpert & Macready, 1997). This theorem may be considered as a supporting point for combining various metaheuristic algorithms in solving complex optimization problems more efficiently. Bringing different metaheuristic algorithms together and running them in sequence will demand considerable computational time (Baykasoğlu et al., 2019). It is well known that most of the metaheuristic algorithms are developed to run sequentially, so parallel execution of the metaheuristic algorithms can reduce the run time and may improve the solution quality (Alba, 2005; Alba & Troya, 2002; Baykasoğlu & Şenol, 2021).

Initially, all the parallel metaheuristic algorithms in solving optimization problems are investigated, and the studies in the literature are overviewed related to this subject. After the detailed literature review, it is seen that parallel execution of different metaheuristic algorithms under a controller (*under another metaheuristic algorithm*) has not been receiving considerable attention in the literature. Based on this motivation, we propose a new approach, which employs a swarm intelligence based metaheuristic known as WSAR (Baykasoğlu, 2020) as controller to guide several SSBMAs in parallel to solve optimization problems.

The proposed algorithm is named as p-WSAR in this study. p-WSAR is developed in MATLAB by benefiting MATLAB parallel programming toolbox. p-WSAR brings together several SSBMAs, namely Random Search (RS) (Rogers, 1972), Threshold Accepting (TA) (Dueck & Scheuer, 1990), Great Deluge (GD) (Dueck, 1993), Greedy

Search (GS) (Feo & Resende, 1995), Simulated Annealing (SA) (Kirkpatrick et al., 1983), to run in parallel and achieves coalition through superposition principle and reproduction procedure of WSAR algorithm of Baykasoğlu (2020, 2021).

p-WSAR can be considered parallel version of the WSAR algorithm, however, p-WSAR have many distinguishing features that makes it considerable different from classical WSAR algorithm. The main distinguishing differences between WSAR and p-WSAR are briefly mentioned in Table 1.

Table 1.1 Main differences between WSA and P-WSAR

Characteristic	**WSAR**	**P-WSAR**
Agents	Solution vectors	SSBMAs
Generation	Serial	Parallel
Parameters	Fixed	Selected from a range randomly during the search
Role	The main optimizer	Serves as a controller of SSBMAs

The main contributions and motivation of the present study can be summarized as follows:

- The No Free Lunch Theorem claims that there is no statistical difference between the performances of optimization algorithms if we try to solve all optimization problems. This study is a research attempt that combines different metaheuristic algorithms in order to provide a satisfactory performance on solving the optimization problems by benefiting their diverse characteristics.
- Parallel execution of metaheuristic algorithms is very effective in solving large-scale optimization problems where sequential methods cannot provide solutions in reasonable computational times. In this study, execution times are shortened by parallel execution of SSBMAs under p-WSAR.

1.2 Outline of the Thesis

The thesis consists of seven chapters and the remainder of the thesis is organized as follows.

Existing studies in the literature about coalition of metaheuristic algorithms and parallel metaheuristic are overviewed in Chapter 2. The proposed p-WSAR algorithm is given in Chapter 3. The performance of the proposed algorithm on continuous optimization problems, binary optimization problems, and combinatorial optimization problems are examined in Chapter 4, Chapter 5, and Chapter 6 respectively. Conclusions and future research suggestions are given in Chapter 7.

CHAPTER 2
LITERATURE REVIEW BASED ON COALITION OF METAHEURISTIC ALGORITHMS AND PARALLEL METAHEURISTIC ALGORITHMS

In this chapter, the studies related to coalition of metaheuristic algorithms and parallel metaheuristic algorithms are overviewed in two parts. The studies which are related with coalition of metaheuristic algorithms are given in Section 2.1. In Section 2.2, the studies that consider the parallel metaheuristic algorithms are investigated.

2.1 Literature Review based on Coalition of Metaheuristic Algorithms

In the related literature, most of the studies aim to provide coalition between several metaheuristic algorithms through agent based systems. In one of the studies, Milano and Roli (2004) introduced the Multi-Agent-Metaheuristic-Architecture (MAGMA). MAGMA is organized in four different abstraction levels. One or more agents built up each level. Level-0 is made up from agents that generate initial solutions. These initial solutions are tried to be improved by the agents in Level-1 through local search operators. Level-2 agents are responsible for directing the search through promising solutions and generating strategies for escaping from the local optima. Level-3 agents provide the coalition. Their coalition is not dynamic because of their layer-dependent structure.

Talbi and Bachelet (2004) presented Co-Search approach that based on cooperation of multi agents. They aimed to balance between diversification and intensification. The proposed system is formed by three different agents, namely, search agent, diversifying agent and intensifying agent. The agents formed the coalition through an adaptive memory.

Jedrzejowicz and Wierzbowska (2006) developed JADE-A-Team (JABAT) framework. The framework composed of two different types of agents, namely, OptiAgent and SolutionManagers. While OptiAgents search the solution space, SolutionManagers is responsible for coalition. The all OptiAgents search simultaneously.

Meignan et al. (2008) presented a platform that is named as Agent Metaheuristic Framework (AMF). AMF provides analyzing the existing algorithms and designing new coalition based metaheuristic algorithms. It also organizes the roles of different metaheuristic algorithms. In another study, Meignan et al. (2010) utilized AMF to address the Vehicle Routing Problem (VRP).

Martin et al. (2016) introduced an agent-based system in which each agent executes a different combination of local search/metaheuristic algorithm. They evaluated their approach by solving scheduling problems. Results proved the effectiveness of their implementation. In some of the problems the best known solutions are improved.

Ruiz et al. (2018) presented the PSO-CO algorithm to address continuous optimization problems. In the related study, the coalition between each solution of the PSO algorithm, which is a population-based metaheuristic algorithm, is trying to be achieved with an agent-based approach.

Baykasoğlu et al. (2019) proposed a metaheuristic algorithm that is named as single seekers society (SSS), which enables cooperation of several metaheuristic algorithms. SSS presented better performance than the algorithms, which are members of the coalition. However, SSS runs algorithms in serial that demands considerable computational time.

2.2 Literature Review based on Parallel Metaheuristic Algorithms

There have been many research attempts, especially in the last decade to analyze the performance of parallel metaheuristic algorithms. Memeti et al. (2019) analyzed and classified parallel machine learning and parallel metaheuristic algorithms in terms of compile time and runtime. They also stated the challenges and the research gaps. Gmys et al. (2020) compared several programming language performance at productivity, scalability and performance. In addition to the comparison, they provided feedback about the parallelization and implementation of each programming language. Abdelhafez et al. (2019) investigated the behavior of the parallel metaheuristic algorithms through two experiments, which have different termination conditions. The predefined number of solutions is determined as the termination condition in the first experiment. The performance of the sequential and parallel versions of several metaheuristic algorithms are compared in terms of the solution quality. The results proved the superiority of parallel versions. In the second experiment, the termination condition is defined as the predefined fitness quality. The parallel version of the metaheuristic algorithms also surpassed the sequential ones in the second experiment.

In addition to the studies that consider the performance of the parallel metaheuristic algorithms, there has been much research devoted to solve complex optimization problems by using parallel metaheuristic algorithms. These studies can be classified into in two groups as the single solution based parallel metaheuristic algorithms and population based parallel metaheuristic algorithms.

Although the single solution based metaheuristic algorithms (SSBMA) are capable of solving optimization problems, most of the researchers presented parallel version of them in order to enhance their searching capability further. In one of such studies, Wang et al. (2015) suggested a parallel SA (p-SA) algorithm for solving vehicle routing problem with simultaneous pickup-delivery and time windows (VRPSPDTW). In their proposed algorithm, they utilized multiple Markov chains to speed up their algorithm. The results revealed that p-SA algorithm produced more effective solutions than sequential SA and GA in respect of solution time and solution quality. In another study, Herrán et al. (2020) developed a parallel variable neighborhood search algorithm (PVNS) for solving the p-median problem. They incorporated several sequential VNS algorithms simultaneously through multiple processors in the computer. An extensive set of experiments was organized. According to the results, the proposed method can produce effective solutions. One can find more studies about single solution based parallel metaheuristic algorithms in the following references; Munguía et al. (2018), Rios et al. (2018) and Gonçalves-e-Silva et al. (2018). In most of these studies, researchers' utilized single solution based parallel metaheuristic algorithms to speed up the search process.

In the second group, researchers developed population based parallel metaheuristic algorithms. In this group, researchers took the advantage of parallelization in two aspects. The first one is acceleration and the second one is communication. For example, Palomo-Romero et al. (2017) developed an island model parallel GA to solve unequal area facility layout problems (UA-FLP). In their proposed algorithm, they produced multi populations that search in parallel and communicate by migration. Computational results showed that their method is capable of reaching good solutions in reasonable computational time. In another study, Wang et al. (2020) proposed an Ant Colony (ACO) based parallel co-evolutionary algorithm to schedule parallel non-identical batch machines. Their proposed algorithm consists of exploration and coordination searches of three colonies. They also designed a local search strategy to increase the diversity of the solutions. The effectiveness of the algorithm is measured through some statistical tests. The tests proved the proposed method's capability for solving the mentioned problem. For more studies on parallel population based metaheuristic algorithms, readers may refer the following studies; Dokeroglu et al. (2020), de Sousa Junior et al. (2020) and Abed-alguni & Alawad (2021).

The coalition schemes of the above-mentioned studies are depicted below in Table 2. As it can be seen in the Table 2, besides the above-mentioned metaheuristic algorithms, parallel execution of different metaheuristic algorithms under a controller (*under another metaheuristic algorithm*) has not been receiving considerable attention in the literature. Based on this motivation, we propose a new approach, which employs a swarm intelligence based metaheuristic known as WSAR (Baykasoğlu, 2020) as controller to guide several SSBMA in parallel to solve unconstrained continuous optimization problems.

Table 2.1 Coalition schemes of the studies

Study	Algorithm	Coalition scheme used
Milano and Roli (2004)	MAGMA	Agent based
Talbi and Bachelet (2004)	Co-Search	Adaptive memory through agents
Jedrzejowicz and Wierzbowska (2006)	JABAT	Agent based
Baykasoğlu et al. (2019)	SSS	Coalition with a heuristic
Meignan et al. (2010)	CBM	Agent based
Ruiz et al. (2018)	PSO-CO	Agent based
Martin et al. (2016)	Several local search algorithms	Agent based
Wang et al. (2015)	p-SA	Parallel execution with best solution sharing
Herrán et al. (2020)	PVNS	Parallel execution with best solution sharing
Palomo-Romero et al. (2017)	P-GA	Parallel execution with migration among multiple populations
Wang et al. (2020)	ACO	Parallel execution with pheromone sharing
The proposed study	p-WSAR	Coalition with an another metaheuristic

CHAPTER 3
p-WSAR ALGORITHM

In this section, p-WSAR algorithm is defined in detail. p-WSAR composes of five single solutions based metaheuristic algorithms, namely, Random Search (RS) (Rogers, 1972), Threshold Accepting (TA) (Dueck & Scheuer, 1990), Great Deluge (GD) (Dueck, 1993), Greedy Search (GS) (Feo & Resende, 1995), Simulated Annealing (SA) (Kirkpatrick et al., 1983) and the WSAR algorithm of Baykasoğlu (2020, 2021). Pseudo codes of SSBMAs are given in Appendix. SSBMA algorithms are implemented in their basic forms, details about them can be found easily from the literature. While SSBMAs are responsible for the exploration of the search space, WSAR algorithm serves as a controller and guides the whole searching process. p-WSAR has three main phases, which are named as "search phase", "information sharing phase (*determining attractive and repulsive superpositions*)" and "reproduction phase". Details of these phases can be found in subsections 3.1, 3.2 and 3.3 below. The major stages of p-WSAR are illustrated in Figure 3.1. A simplified flow chart of the proposed algorithm is depicted in Figure 3.2. Notations used in p-WSAR and their definitions are given below:

Notations of p-WSAR and their definitions:

Maxiter	Iteration number (stopping condition)
Iteration	Current iteration number
NS	Number of single-solution based metaheuristics
D	Number of dimensions of the problem
τ	User defined parameter for weight determination of the solution vectors
λ	User defined parameter for modification of the step size
ϕ	User defined parameter for the step size determination
UL	Upper limit for the dimensions
LL	Lower limit for the dimensions
f(i)	Fitness of the solution vector *i*
f(as)	Fitness of the attractive superposition
weight	Weight of a solution vector

$\vec{x}$	The current position vector of a solution
x_{as}	Position vector of the attractive superposition
x_{rs}	Position vector of the repulsive superposition
ss	Step length
t	Iteration number
AS	Attractive superposition
RS	Repulsive superposition

```
PROCEDURE p-WSAR
Randomly generate initial solutions
Match randomly generated initial solutions with randomly selected SSBMAs
WHILE iteration<MaxIter
    WHILE termination conditions for SSBMAs are not met
        Run SSBMAs in parallel
    END WHILE
    Sort the solutions returned by SSBMAs  according to their fitness values
    Determine attractive and repulsive superpositions
    Calculate attractive and repulsive superpositions' fitness
        PARFOR each solution returned by SSBMAs   //parallel FOR
            IF solution_fitness < f(AS)
              Randomly move solutions
            ELSE
                Move towards superposition
            END IF
            Randomly generate a new set of parameters for SSBMAs
            Randomly match solutions with SSBMAs
        END FOR
END WHILE
END PROCEDURE
```

Figure 3.1 Main steps of the p-WSAR algorithm

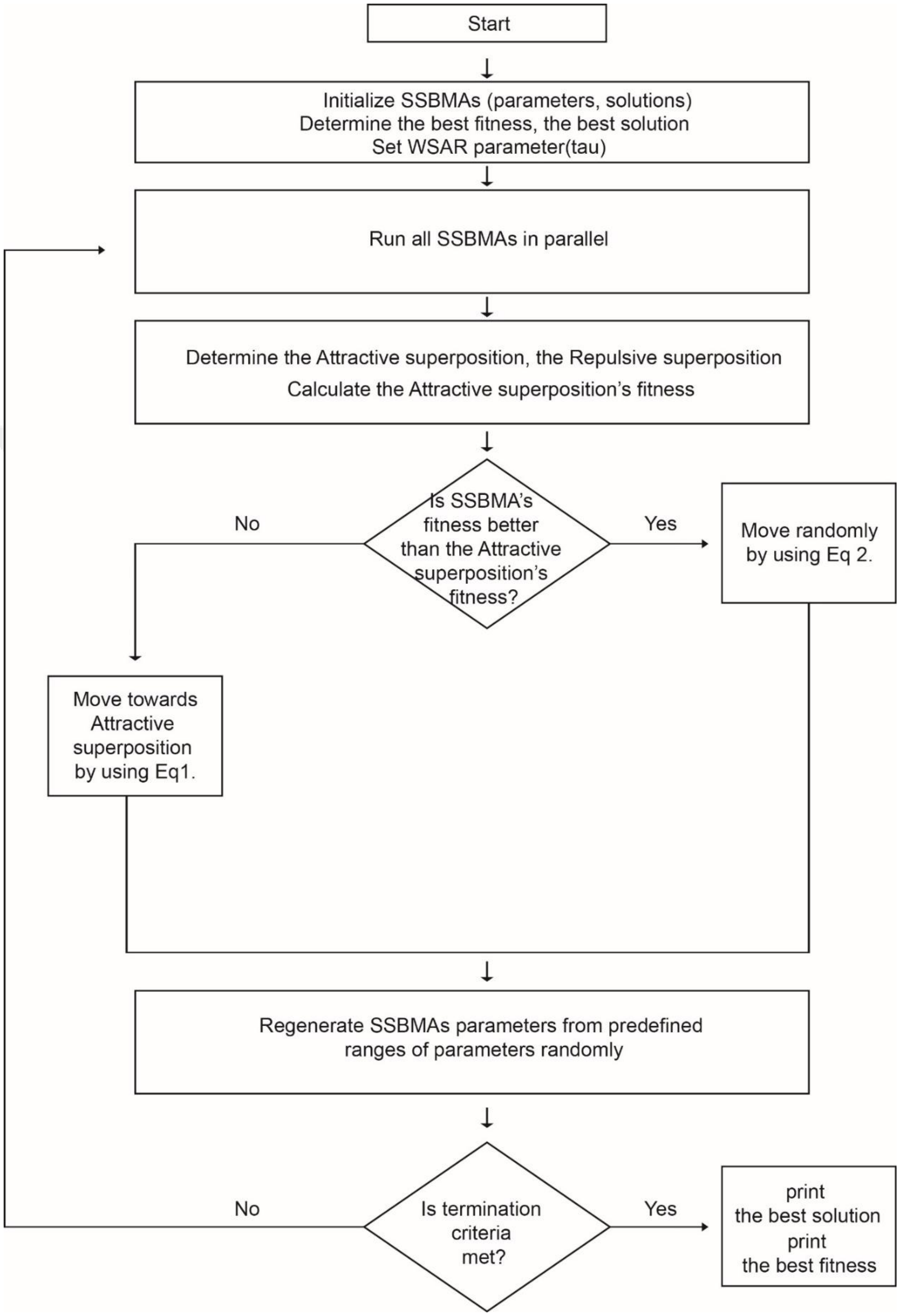


Figure 3.2 The simplified flow chart of the p-WSAR algorithm

3.1 Search phase

P-WSAR starts with randomly generating initial solutions and their matching with SSBMAs. Then, algorithm-specific parameters are assigned to each SSBMA from a predefined parameter set. The parameter set of each SSBMA is depicted in Appendix TableA1. Each SSBMA specific parameter is defined within a range (upper and lower levels for each parameter). The parameters' ranges are selected among the most used values in the metaheuristic literature. At every triggering of SSBMAs within the p-WSAR parameters are chosen randomly from these sets. After the initialization step, the search phase starts. In this phase, each SSBMA tries to improve its solution by itself in parallel with other SSBMAs until the termination conditions are met. At this point, several termination criteria can be defined, like convergence, time limit, the number of individuals ready for information sharing. These termination criteria vary according to SSBMA. In the present implementation, the termination criterion is determined as failure to improve the objective function over a certain number of iterations. After termination, solutions returned by SSBMAs are ranked according to their fitness values in an ascending and descending order. After ranking, weights for each solution are calculated by the following equation i^((-τ)), where i represents the solution rank and τ represents the weight ratio (greater values of τ encourages elitist behavior of the p-WSAR, on the contrary, lower values of τ yield to increase diversification of the algorithm). The information-sharing phase starts after calculating the weights.

3.2 Information sharing phase

In this phase, SSBMAs share their findings by utilizing the superposition principle. Detailed explanations about superposition principle and its relation with optimization are defined in Baykasoğlu and Akpınar (2015, 2017). The attractive and the repulsive superpositions are determined through a special iterative process, which is explained in detail by Baykasoğlu (2020). The solution vectors will try to adopt itself to the attractive superposition (by try to get closer to attractive superposition point) and try to escape the repulsive superposition (by try to get away from repulsive superposition point). Superpositions play the critical role of search direction determination, which is one of the problematic issue in metaheuristic search. The superposition determination process can be summarized as follows: at first, an empty vector that is to going to be the superposition at the end is created. Then, a random number is assigned to each cell of this empty vector. Afterwards, the assigned random numbers are compared with the weight of each solution vectors that were returned by SSBMAs. If the weight of the solution vector is greater than the generated random number for that cell, the corresponding cell value of the solution vector becomes a candidate for the corresponding cell of the superposition. When all of the candidate values are determined for the corresponding cell, the candidate values are evaluated through a roulette wheel approach and the winning value is assigned to the corresponding cell of the superposition vector. This iterative process continues until all of the cell values of the superposition vector are determined. The defined procedure above gives chance to each solution returned by SSBMA to determine the superposition vector in relation to their fitness values probabilistically. An example of determining attractive and repulsive superposition is depicted Figure 3.3. In this figure, grey cells show the candidate solution elements. Cells with asterisk represent the values that chosen among candidate solution elements. The superposition determination procedure is also outlined in Figure 3.4.

Ranks	Solution Vectors					*weight* (i) $= i^{-\tau}$
1	0.45*	0.23	0.15*	1.49	3.51*	1.000
2	0.24	0.46	0.47	0.99*	1.98	0.574
3	0.17	0.27*	0.16	1.87	2.01	0.415
4	0.28	0.59	0.09	0.64	2.14	0.329
5	0.37	0.28	0.24	0.15	2.19	0.275
6	0.21	0.54	0.37	2.23	2.06	0.238
random numbers	0.611	0.411	0.701	0.214	0.800	
Attractive superposition	0.45	0.27	0.15	0.99	3.51	

Ranks	Solution Vectors					*weight* (i) $= i^{-\tau}$
1	0.21*	0.54*	0.37	2.23	2.06*	1.00
2	0.37	0.28	0.24*	0.15	2.19	0.574
3	0.28	0.59	0.09	0.64*	2.14	0.415
4	0.17	0.27	0.16	1.87	2.01	0.329
5	0.24	0.46	0.47	0.99	1.98	0.275
6	0.45	0.23	0.15	1.49	3.51	0.238
random numbers	0.501	0.654	0.236	0.356	0.277	
Repulsive superposition	0.21	0.54	0.24	0.64	2.06	

Figure 3.3 Determination of the attractive and repulsive superpositions in p-WSAR for $\tau = 0.8$ (Adapted from Baykasoğlu (2020))

```
Procedure DETERMINE SUPERPOSITION(D,NS)
as =zeros (1,D) // an empty vector is created for AS
FOR (i= 1 to D) DO
    Candidatelist=[]
      prob=rand ()
        FOR (j=1 to NS) DO
            IF weight (1,j) > prob  (See Figure 3.3)
                Candidate_list (1,Counter)=x(j,i) // the corresponding cell is added to the candidate list
                Candidate_list (2,Counter)=weight (1,j)  // The weight is added to the list for roulette
wheel selection
             END IF
          END FOR
    as(1,i)= Candidatelist(1,ROULETTE(Candidatelist (2,: ) ) // cell value determined through
roulette wheel
 END FOR
```

Figure 3.4 Pseudo code for superposition determination

3.3 Reproduction phase

In this phase, the next starting solution of each SSBMA is determined. The fitness value of the attractive superposition and fitness values of solutions returned by SSBMAs is compared for this purpose. If the fitness value of the attractive superposition is better than the solution returned by SSBMA, the solution returned by SSBMA moves towards superposition by utilizing Equation (3.1) that was proposed by Baykasoğlu (2020). Otherwise, the solution returned by SSBMA makes a random move by using Equation (2) (Baykasoğlu, 2020). In equation (3.1), while the solution is moving towards superposition, it is also moving away from the repulsive superposition. Equation (3.3) utilizes a specially developed random walk function that was previously devised by Baykasoğlu (2020), and used in several complex optimization problems (Baykasoğlu & Baykasoğlu, 2020; Baykasoğlu & Baykasoğlu, 2021).

$$x_{ij}(s+1) = x_{ij}(s) + \left(rand(0,1) * \left(x_{as}(s) - |x_{ij}(s)|\right) - rand(0,1) * \left(x_{rs}(s) - |x_{ij}(s)|\right)\right) \quad (3.1)$$

$$x_{ij}(s+1) = x_{ij}(s) + ss(s) * unifrd(-1,1) * x_{ij}\ (s) \quad (3.2)$$

$$ss(s+1) = f(x) = \begin{cases} ss(s) - e^{-s\,/(s+1)} * \phi * ss(s)\ ,\ if\ rand(0,1) \le \lambda \\ ss(s) + e^{-s\,/(s+1)} * \phi * ss(s)\ ,\ if\ rand(0,1) > \lambda \end{cases} \quad (3.3)$$

Where, $x_{ij}(s)$ is the value of the position of the solution vector i on dimension j at iteration s; $x_{rs}(s)$ and $x_{as}(s)$ are the repulsive superpositions and the attractive at iteration s, $|.|$ Denotes absolute value, $ss(s)$ is the value of the step size at iteration s. ϕ and λ are the parameters of the random walk function that are used to adjust shape of this function. Their suitable setting is discussed in detail in Baykasoğlu (2020).

After determining the new positions of the solutions returned by SSBMA via Equations (3.1-3.3), these solutions are randomly matched with SSBMAs as the new starting points for their search. Before SSBMAs' begin their search, their algorithm specific parameters are also randomly assigned from predefined parameter sets. In this way, a SSBMA utilizes a variety of parameters during the whole p-WSAR search process that may decrease the need for parameter optimization. The predefined parameter set is given in the following Table 3.1.

Table 3.1 Parameter settings of SSBMAs and WSAR

Algorithm	Parameter	Lower Bound	Upper Bound
SA	Temperature (T)	500	1000
SA	Cooling rate (Cr)	0.8	0.99
SA	Iteration number	250	1000
TA	Threshold	2	10
TA	Threshold Update	0.8	0.99
TA	Iteration number	250	1000
GD	Level	0.8	0.9
GD	Level Update	0.9	0.99
GD	Iteration number	250	1000
GS	Iteration number	250	1000
RS	Iteration number	250	1000

CHAPTER 4
p-WSAR ALGORITHM ON CONTINUOUS OPTIMIZATION PROBLEMS

In the previous chapter, the p-WSAR algorithm is given. In this section, a brief information about performance measures related to parallel solution methods given first. Then, performance analysis of p-WSAR on recently introduced complex unconstrained continuous global optimization problems, namely CEC2020 (Yue et al., 2019) is performed.

Parallel solution methods have different characteristics than the classical methods. Therefore, it is important to determine performance measures before computational study. The main aim to design parallel solution methods is reducing the solution time required to solve the given problem exactly or approximately. For the exact solution methods that guarantee the optimal solution, the speed up, which is the ratio between the run time of the parallel algorithm and the best known sequential algorithm is the performance measure. In the case of approximate methods, especially metaheuristic algorithms, the speedup measure is difficult to define since the optimal solution is not guaranteed. In most of the cases, parallel metaheuristic algorithms find different solutions of their sequential versions (when they exist). Barr and Hickman defined some performance measures. According to their study, parallel metaheuristic algorithms should surpass their sequential versions with respect to solution quality and solution time. That's to say, parallel metaheuristic algorithm should not need higher computational time than its sequential version, or should provide better solutions (Barr & Hickman, 1993). In this study, p-WSAR's performance is also analyzed in respect of solution quality and solution time.

4.1 Test problems and computational results

Performance of p-WSAR is tested on the latest data set of the Competitions on Evolutionary Computation (CEC), CEC2020. Comparisons is made against 11 metaheuristic algorithms that have solved CEC2020 problems recently namely, Autonomous Groups Particles Swarm Optimization-3 (AGPSO3) (Mirjalili et al., 2014), Atom Search Optimization (ASO) (Zhao et al., 2019), Harris Hawk Optimization (HHO) (Heidari et al., 2019), Electrostatic Discharge Algorithm (ESDA) (Shadravan et al., 2019), Salp Swarm Algorithm (SSA) (Mirjalili et al., 2017), Coyote Optimization Algorithm (COA) (Pierezan & Dos Santos Coelho, 2018), Covariance Matrix Adaptation (CMAES) (Hansen & Ostermeier, 2001), Yin–Yang-Pair Optimization (YYPO), Opposition-based Yin–Yang-Pair Optimization (OOYO) (W. chuan Wang et al., 2021), and Weighted Superposition Attraction and Repulsion Algorithm (WSAR) (Baykasoğlu, 2021).

The dataset consists of 10 single objective bound constrained numerical optimization problems (CEC1-CEC10) in four dimensions (5D, 10D, 15D, 20D). The characteristics of the problems and their optimal solutions are given in Table 4.1. One can refer to Yue et al. (2019) for detailed information about the CEC2020 dataset.

Table 4.1 Summary of the CEC2020 dataset

Function type	No.	Functions	$F_i^* = F_i(x^*)$
Unimodal Function	CEC1	Shifted and Rotated Bent Cigar Function	100
Basic Functions	CEC2	Shifted and Rotated Schwefel’s Function	1100
	CEC3	Shifted and Rotated Lunacek bi-Rastrigin Function	700
	CEC4	Expanded Rosenbrock’s plus Griewangk’s Function	1900
Hybrid Functions	CEC5	Hybrid Function 1	1700
	CEC6	Hybrid Function 2	1600
	CEC7	Hybrid Function 3	2100
Composition Function	CEC8	Composition Function 1	2200
	CEC9	Composition Function 3	2400
	CEC10	Composition Function 3	2500
Search Range: $[-100,100]^D$			

p-WSAR is implemented in MATLAB 2022a. MATLAB parallel programming toolbox which enables us to run multiple SSBMAs simultaneously through multiple processors is utilized. In other words, each SSBMA is assigned to a CPU core on which it will run. p-WSAR is tested on all problems in dimension 5,10,15,20 except for CEC6 and CEC7 for 5D. 30 independent runs for each problem are performed on a PC with 8 cores, 3.2 GHz CPU and 16 GB RAM. The effects of the random parameters on solution quality is considered by performing 30 independent runs for each problem for more fair analyses in parallel to common practices in the related literature. In addition, comparisons between different algorithms are made through mean values instead of the best values. For making fair comparisons the maximum number of function evaluations is set as same as the comparison algorithms (3×10^4, 5×10^4, 10^6 and 10^7 for 5D, 10D, 15D, 20D respectively). The best, the median, the standard deviation and mean of the experimental results are recorded. Performance is evaluated through the error between the best value obtained from p-WSAR and the global optimum. Moreover, median and the standard deviation of the obtained results are used for testing the stability. Mean values of the obtained results are used for determining the average performance of p-WSAR. Moreover, the performance of p-WSAR is statistically compared with the other algorithms through nonparametric statistical tests by using mean values.

Table 4.2 indicates that p-WSAR can find close results to the global optimum on all instances in the 5D benchmark dataset. It also has the lowest mean on all instances except CEC1. Table 4.6 shows that (from Friedman test) p-WSAR performs the best among other algorithms. In addition, based on the Wilcoxon signed-rank test, the difference between p-WSAR and COA, CMAES is found negligible as $p > 0.05$. Besides p-WSAR is performed slightly better than YYPO, OOYO, ESDA, HHO, SSA, AGPSO3 and WSAR as the $p<0.05$.

Table 4.3 shows that p-WSAR can find the best result on CEC9 and CEC10 instance and near optimum results on CEC4, CEC6 instances in the 10D benchmark dataset. It also has the lowest mean value on CEC2, CEC9 and CEC10 instances. Table 4.7 reveals that (based on the Friedman test results) p-WSAR performs the best among other algorithms. Moreover, based on the Wilcoxon signed-rank test, the difference between p-WSAR and COA, CMAES is found negligible as the $p > 0.05$. Besides p-WSAR is performed slightly better than OOYO, YYPO, ASO, AGPSO3, ESDA, HHO, SSA, CMAES and WSAR as $p<0.05$. Table 4.4 reveals that p-WSAR can find the best results on CEC9 instance and near optimum results on CEC4, and CEC6 instances in the 15D benchmark dataset. Table 4.8 presents that p-WSAR performs the second best among the other algorithms. In addition, based on the Wilcoxon signed-rank test, the difference between p-WSAR and COA, ASO, OOYO, YYPO is found negligible as p> 0.05. Besides p-WSAR is performed slightly better than AGPSO3, ESDA, HHO, SSA, CMAES and WSAR as p<0.05. Finally, Table 4.5 reveals that p-WSAR can find near optimum results on CEC4, and CEC9 instances for the 20D benchmark dataset. Based on the Friedman test results, Table 4.9 indicates that p-WSAR performs the second worst among other compared algorithms.

Table 4.2 Results of 5D case in CEC2020 dataset

Problems	Algorithm	OOYO	YYPO	ESDA	ASO	HHO	SSA	AGPSO3	COA	CMAES	WSAR	p-WSAR
CEC1	Best	4.56E – 01	1.14E + 02	0.00E + 00	9.86E + 00	2.32E + 02	2.40E + 00	1.36E + 00	0.00E + 00	0.00E + 00	3.59E+00	3.58E-01
	Median	1.14E + 03	3.31E + 03	0.00E + 00	2.37E + 03	7.05E + 03	5.46E + 03	1.69E + 03	0.00E + 00	0.00E + 00	7.54E+03	3.59E-01
	Mean	2.37E + 03	4.26E + 03	1.32E + 01	2.82E + 03	7.34E + 03	5.04E + 03	2.88E + 03	3.00E – 12	0.00E + 00	2.95E+04	3.61E-01
	Std. dev	3.13E + 03	3.39E + 03	7.22E + 01	2.59E + 03	5.12E + 03	3.44E + 03	3.20E + 03	1.09E – 11	0.00E + 00	8.80E+04	3.13E-03
CEC2	Best	1.25E – 01	1.25E – 01	3.75E – 01	1.25E – 01	1.25E + 02	2.50E – 01	1.25E – 01	1.25E – 01	2.50E – 01	2.71E+01	6.70E+00
	Median	1.34E + 01	7.06E + 00	1.60E + 02	1.25E + 02	3.64E + 02	2.40E + 02	3.02E + 01	1.04E + 01	3.85E + 02	6.50E+01	6.70E+00
	Mean	5.29E + 01	4.70E + 01	2.18E + 02	1.14E + 02	3.64E + 02	2.40E + 02	6.20E + 01	2.47E + 01	3.91E + 02	6.70E+01	6.70E+00
	Std. dev	5.83E + 01	6.01E + 01	1.42E + 02	9.29E + 01	1.78E + 02	1.22E + 02	6.55E + 01	2.89E + 01	2.03E + 02	3.08E+01	4.10E-06
CEC3	Best	1.61E + 00	9.95E – 01	7.02E + 00	1.61E + 00	7.51E + 00	3.27E + 00	0.00E + 00	0.00E + 00	5.84E + 00	5.23E+00	4.63E+00
	Median	6.22E + 00	6.66E + 00	1.39E + 01	7.33E + 00	1.75E + 01	1.02E + 01	6.03E + 00	5.41E + 00	1.03E + 01	7.35E+00	4.63E+00
	Mean	6.13E + 00	6.40E + 00	1.69E + 01	7.19E + 00	1.71E + 01	1.03E + 01	5.56E + 00	4.85E + 00	1.08E + 01	7.40E+00	4.63E+00
	Std. dev	1.76E + 00	2.52E + 00	7.88E + 00	2.40E + 00	5.55E + 00	3.20E + 00	2.23E + 00	2.45E + 00	3.43E + 00	1.27E+00	1.40E-07
CEC4	Best	0.00E + 00	9.88E – 03	4.93E – 02	1.27E – 01	3.33E – 04	5.01E – 02	9.88E – 03	0.00E + 00	4.96E – 02	3.95E-02	1.09E-01
	Median	1.74E – 01	2.30E – 01	4.22E – 01	2.78E – 01	8.43E – 01	2.92E – 01	2.64E – 01	1.87E – 01	4.66E – 01	1.58E-01	1.09E-01
	Mean	1.68E – 01	2.37E – 01	4.26E – 01	3.10E – 01	8.81E – 01	3.40E – 01	2.75E – 01	1.64E – 01	5.21E – 01	1.58E-01	1.14E-01
	Std. dev	1.69E – 01	1.12E – 01	2.31E – 01	1.04E – 01	6.39E – 01	1.66E – 01	1.82E – 01	9.78E – 02	3.50E – 01	7.55E-02	3.21E-02
CEC5	Best	9.99E – 01	9.42E – 03	0.00E + 00	9.95E – 01	1.11E + 01	3.88E + 01	0.00E + 00	0.00E + 00	6.24E – 01	1.09E+00	8.68E-08
	Median	1.59E + 01	7.49E + 00	2.53E + 01	8.58E + 00	3.94E + 02	3.12E + 02	2.98E + 00	0.00E + 00	1.90E + 01	3.27E+00	1.16E-07
	Mean	4.45E + 01	9.01E + 00	2.96E + 01	1.24E + 01	6.39E + 02	4.00E + 02	1.65E + 01	2.12E – 01	8.06E + 01	9.79E+00	1.71E-01
	Std. dev	7.68E + 01	6.75E + 00	3.69E + 01	1.26E + 01	6.71E + 02	4.61E + 02	3.30E + 01	3.40E – 01	1.09E + 02	2.61E+01	6.20E-01

Table 4.2 Continues

Problems	Algorithm	OOYO	YYPO	ESDA	ASO	HHO	SSA	AGPSO3	COA	CMAES	WSAR	p-WSAR
CEC8	Best	1.44E – 05	7.61E – 05	0.00E + 00	0.00E + 00	7.21E – 02	1.36E – 06	0.00E + 00	0.00E + 00	**0.00E + 00**	1.02E-04	1.07E-05
	Median	2.16E – 04	2.38E – 04	1.29E + 01	0.00E + 00	1.03E + 02	3.30E + 00	9.70E + 00	3.00E – 10	1.01E + 02	2.49E-04	1.07E-05
	Mean	2.20E – 01	1.39E + 00	1.22E + 01	2.26E + 00	7.07E + 01	1.60E + 01	2.06E + 01	3.31E + 00	8.18E + 01	2.11E+01	**2.77E-05**
	Std. dev	1.21E + 00	3.62E + 00	4.72E + 00	4.62E + 00	4.58E + 01	3.11E + 01	3.69E + 01	5.31E + 00	3.86E + 01	4.07E+01	2.46E-05
CEC9	Best	1.00E + 02	1.03E – 02	1.00E + 02	1.00E + 02	1.00E + 02	1.80E – 04	0.00E + 00	**2.50E – 10**	1.00E + 02	5.10E+01	7.92E-04
	Median	1.00E + 02	1.00E + 02	1.00E + 02	1.00E + 02	2.03E + 02	1.00E + 02	1.00E + 02	1.00E + 02	3.18E + 02	1.00E+02	8.85E-04
	Mean	1.00E + 02	9.33E + 01	1.06E + 02	1.00E + 02	1.57E + 02	1.15E + 02	1.79E + 02	1.01E + 02	2.88E + 02	1.87E+02	**7.22E+00**
	Std. dev	5.66E – 04	2.54E + 01	1.30E + 01	2.23E – 13	5.42E + 01	7.30E + 01	1.14E + 02	6.89E + 01	7.62E + 01	1.13E+02	1.41E+01
CEC10	Best	3.00E + 02	8.57E – 03	3.00E + 02	3.47E + 02	3.47E + 02	3.00E + 02	3.00E + 02	**4.15E – 09**	3.00E + 02	3.01E+02	6.20E-04
	Median	3.47E + 02	3.47E + 02	3.47E + 02	3.47E + 02	3.47E + 02	3.47E + 02	3.47E + 02	3.47E + 02	3.47E + 02	3.47E+02	6.20E-04
	Mean	3.35E + 02	3.06E + 02	3.46E + 02	3.47E + 02	3.49E + 02	3.39E + 02	3.44E + 02	3.23E + 02	3.46E + 02	3.46E+02	**1.00E+01**
	Std. dev	2.13E + 01	9.57E + 01	8.65E + 00	7.95E – 03	7.63E + 00	1.80E + 01	1.20E + 01	6.46E + 01	8.65E + 00	8.48E+00	5.48E+01

Table 4.3 Results of 10D case in CEC2020 dataset

Problems	Algorithm	OOYO	YYPO	ESDA	ASO	HHO	SSA	AGPSO3	COA	CMAES	WSAR	p-WSAR
CEC1	Best	1.69E – 02	1.87E – 01	0.00E + 00	8.39E – 01	1.91E + 04	1.21E + 00	1.52E + 01	0.00E + 00	**0.00E + 00**	2,21E+06	1.71E+02
	Median	2.72E + 03	1.49E + 03	0.00E + 00	5.26E + 02	9.49E + 04	7.14E + 02	5.53E + 03	0.00E + 00	0.00E + 00	3,80E+07	1.75E+02
	Mean	3.25E + 03	3.03E + 03	2.34E + 06	1.14E + 03	9.58E + 04	1.41E + 03	2.34E + 06	4.21E + 02	**0.00E + 00**	3,93E+07	1.81E+02
	Std. dev	2.73E + 03	3.25E + 03	1.28E + 07	1.60E + 03	4.35E + 04	1.80E + 03	1.28E + 07	2.31E + 03	0.00E + 00	1,86E+07	2.70E+01
CEC2	Best	**6.83E + 00**	1.03E + 01	3.82E + 02	7.14E + 00	2.44E + 02	2.19E + 01	2.63E + 01	7.02E + 00	2.19E + 01	2,74E+02	2.18E+01
	Median	1.56E + 02	1.42E + 02	8.89E + 02	2.63E + 02	6.29E + 02	4.92E + 02	1.89E + 02	1.49E + 02	6.35E + 02	7,76E+02	2.18E+01
	Mean	1.94E + 02	1.74E + 02	9.03E + 02	2.93E + 02	6.26E + 02	5.07E + 02	2.07E + 02	1.47E + 02	6.16E + 02	7,16E+02	**5.73E+01**
	Std. dev	9.90E + 01	1.27E + 02	3.42E + 02	2.12E + 02	1.85E + 02	2.25E + 02	1.13E + 02	9.65E + 01	2.98E + 02	1,63E+02	9.42E+01
CEC3	Best	7.89E + 00	8.01E + 00	1.66E + 01	3.51E + 00	3.46E + 01	1.29E + 01	4.63E + 00	**5.43E + 00**	1.31E + 01	1,81E+01	2.38E+01
	Median	1.93E + 01	1.89E + 01	5.26E + 01	1.45E + 01	5.31E + 01	2.45E + 01	1.54E + 01	1.99E + 01	2.61E + 01	2,74E+01	2.38E+01
	Mean	1.89E + 01	1.91E + 01	5.56E + 01	**1.48E + 01**	5.31E + 01	2.56E + 01	1.55E + 01	1.90E + 01	2.71E + 01	2,82E+01	3.31E+01
	Std. dev	4.29E + 00	4.84E + 00	2.55E + 01	3.14E + 00	1.24E + 01	6.76E + 00	3.23E + 00	4.75E + 00	8.43E + 00	5,81E+00	1.35E+01
CEC4	Best	**9.88E – 03**	1.68E – 01	8.19E – 01	5.03E – 01	1.53E + 00	3.85E – 01	2.47E – 01	4.77E – 01	4.67E – 01	4,50E-01	3.33E+00
	Median	6.53E – 01	8.48E – 01	2.18E + 00	8.00E – 01	4.10E + 00	8.84E – 01	8.38E – 01	8.65E – 01	9.43E – 01	9,01E-01	3.46E+00
	Mean	6.33E – 01	8.95E – 01	6.09E + 00	8.32E – 01	4.31E + 00	9.12E – 01	2.59E + 00	**9.35E – 01**	1.00E + 00	1,10E+00	3.51E+00
	Std. dev	2.75E – 01	3.52E – 01	9.68E + 00	2.09E – 01	1.58E + 00	3.10E – 01	6.78E + 00	3.14E – 01	3.43E – 01	6,01E-01	1.82E-01
CEC5	Best	5.72E + 01	1.44E + 01	1.38E + 02	1.78E + 02	4.87E + 02	3.09E + 02	1.49E + 02	**2.08E – 01**	1.36E + 02	4,13E+02	5.31E+02
	Median	3.77E + 02	8.76E + 02	3.59E + 02	6.10E + 02	4.64E + 03	8.60E + 02	5.76E + 02	1.24E + 01	5.25E + 02	1,60E+03	5.77E+02
	Mean	6.74E + 02	1.55E + 03	1.81E + 03	6.34E + 02	4.73E + 03	9.90E + 02	9.94E + 02	**2.28E + 01**	5.73E + 02	4,49E+03	7.88E+02
	Std. dev	7.17E + 02	1.92E + 03	7.66E + 03	2.42E + 02	2.70E + 03	5.01E + 02	1.34E + 03	3.34E + 01	2.67E + 02	5,73E+03	4.06E+02

Table 4.3 Continues

Problems	Algorithm	OOYO	YYPO	ESDA	ASO	HHO	SSA	AGPSO3	COA	CMAES	WSAR	p-WSAR
CEC6	Best	**9.02E – 02**	1.92E – 01	2.93E – 02	8.68E – 09	5.69E – 02	3.60E – 02	2.45E – 01	3.04E – 01	5.13E – 01	5,28E+00	1.50E+00
	Median	5.19E – 01	7.15E – 01	8.56E – 01	2.23E – 05	1.06E + 00	8.47E – 01	4.69E – 01	6.98E – 01	1.72E + 01	2,63E+01	1.50E+00
	Mean	5.09E – 01	6.96E – 01	**9.10E – 01**	1.31E – 01	6.31E + 00	7.77E – 01	5.37E – 01	7.34E – 01	1.81E + 01	4,28E+01	1.59E+00
	Std. dev	3.59E – 01	3.92E – 01	4.71E – 01	2.03E – 01	7.83E + 00	3.95E – 01	2.68E – 01	2.69E – 01	1.53E + 01	4,55E+01	2.51E-01
CEC7	Best	6.42E + 00	1.76E + 00	8.45E – 01	5.91E + 00	3.10E + 02	6.44E + 01	8.19E – 01	**1.11E – 03**	9.24E – 01	6,70E+01	1.63E+02
	Median	1.59E + 02	3.11E + 02	1.97E + 02	9.69E + 01	2.36E + 03	3.18E + 02	1.20E + 02	3.29E – 01	1.80E + 02	3,79E+02	2.28E+02
	Mean	1.49E + 02	6.61E + 02	2.94E + 02	1.27E + 02	3.20E + 03	3.06E + 02	1.63E + 02	**6.63E + 00**	2.05E + 02	4,14E+02	2.73E+02
	Std. dev	8.93E + 01	7.85E + 02	3.49E + 02	1.07E + 02	3.03E + 03	1.35E + 02	1.52E + 02	1.31E + 01	1.96E + 02	2,22E+02	1.03E+02
CEC8	Best	**0.00E + 00**	2.00E + 01	5.89E + 01	0.00E + 00	2.27E + 01	1.70E + 01	1.16E + 01	3.99E + 01	1.00E + 02	1,54E+01	2.60E+01
	Median	0.00E + 00	1.02E + 02	1.03E + 02	0.00E + 00	1.12E + 02	1.04E + 02	1.01E + 02	1.01E + 02	1.00E + 02	1,05E+02	2.60E+01
	Mean	**4.52E + 00**	8.06E + 01	1.02E + 02	1.01E + 01	2.57E + 02	1.01E + 02	9.77E + 01	9.48E + 01	1.00E + 02	1,00E+02	3.27E+01
	Std. dev	1.85E + 01	3.34E + 01	8.49E + 00	2.51E + 01	3.92E + 02	1.60E + 01	3.10E + 01	1.78E + 01	2.31E – 01	2,12E+01	9.03E+00
CEC9	Best	1.00E + 02	1.00E + 02	1.00E + 02	1.00E + 02	1.00E + 02	1.00E + 02	1.00E + 02	1.00E + 02	3.29E + 02	3,42E+02	**1.00E+02**
	Median	1.00E + 02	1.00E + 02	1.00E + 02	1.00E + 02	3.64E + 02	3.37E + 02	3.51E + 02	3.51E + 02	3.38E + 02	3,51E+02	1.00E+02
	Mean	1.00E + 02	1.00E + 02	1.13E + 02	1.00E + 02	3.01E + 02	2.81E + 02	3.27E + 02	3.35E + 02	3.40E + 02	3,51E+02	**1.00E+02**
	Std. dev	0.00E + 00	1.93E – 01	4.82E + 01	4.31E – 13	1.49E + 02	1.02E + 02	7.72E + 01	6.43E + 01	6.41E + 00	5,40E+00	1.97E+00
CEC10	Best	3.98E + 02	3.98E + 02	3.98E + 02	3.98E + 02	2.17E + 02	3.98E + 02	3.98E + 02	3.98E + 02	3.98E + 02	4,01E+02	**1.08E+02**
	Median	3.98E + 02	3.99E + 02	4.44E + 02	3.99E + 02	4.44E + 02	3.98E + 02	4.27E + 02	3.99E + 02	4.44E + 02	4,21E+02	1.40E+02
	Mean	3.98E + 02	4.11E + 02	4.32E + 02	4.16E + 02	4.21E + 02	4.16E + 02	4.25E + 02	4.12E + 02	4.34E + 02	4,31E+02	**2.24E+02**
	Std. dev	1.66E – 01	2.11E + 01	2.86E + 01	2.25E + 01	4.46E + 01	2.28E + 01	2.29E + 01	2.14E + 01	2.01E + 01	2,65E+01	1.37E+02

Table 4.4 Results of 15D case in CEC2020 dataset

Problems	Algorithm	OOYO	YYPO	ESDA	ASO	HHO	SSA	AGPSO3	COA	CMAES	CMAES	P-WSAR
CEC1	Best	1.67E + 01	8.83E + 01	0.00E + 00	1.77E + 01	1.16E + 05	3.97E + 00	7.66E – 03	0.00E + 00	**0.00E + 00**	1,08E+08	8.88E+02
	Median	2.38E + 03	2.23E + 03	0.00E + 00	4.99E + 03	2.84E + 05	1.62E + 03	9.53E + 03	0.00E + 00	0.00E + 00	4,02E+08	9.25E+02
	Mean	6.73E + 03	6.87E + 03	6.16E – 10	5.35E + 03	2.84E + 05	3.66E + 03	1.69E + 08	0.00E + 00	**0.00E + 00**	4,05E+08	9.27E+02
	Std. dev	7.52E + 03	8.53E + 03	2.11E – 09	4.94E + 03	9.32E + 04	5.33E + 03	6.14E + 08	6.98E – 15	3.73E – 15	1,45E+08	4.74E+01
CEC2	Best	**4.76E + 00**	5.45E + 01	4.67E + 02	1.13E + 01	3.49E + 02	1.38E + 02	3.89E + 01	1.24E + 01	3.66E + 02	8,09E+02	5.89E+02
	Median	3.63E + 02	3.63E + 02	1.41E + 03	3.69E + 02	6.33E + 02	8.91E + 02	4.11E + 02	1.88E + 02	8.92E + 02	1,77E+03	5.89E+02
	Mean	**3.67E + 02**	3.90E + 02	1.43E + 03	3.89E + 02	6.42E + 02	9.43E + 02	3.99E + 02	2.02E + 02	9.03E + 02	1,81E+03	5.93E+02
	Std. dev	1.78E + 02	2.04E + 02	4.46E + 02	2.44E + 02	2.07E + 02	3.48E + 02	2.31E + 02	1.33E + 02	2.65E + 02	3,73E+02	6.75E+00
CEC3	Best	2.12E + 01	**1.84E + 01**	5.69E + 01	1.90E + 01	5.32E + 01	2.33E + 01	1.88E + 01	2.08E + 01	2.43E + 01	5,71E+01	1.16E+02
	Median	3.11E + 01	2.67E + 01	1.35E + 02	2.53E + 01	8.84E + 01	4.88E + 01	2.17E + 01	2.65E + 01	3.74E + 01	8,06E+01	1.68E+02
	Mean	3.18E + 01	2.78E + 01	1.86E + 02	**2.53E + 01**	8.36E + 01	4.89E + 01	2.22E + 01	2.66E + 01	3.73E + 01	8,04E+01	1.49E+02
	Std. dev	6.17E + 00	4.96E + 00	1.15E + 02	3.69E + 00	1.55E + 01	1.16E + 01	2.86E + 00	4.03E + 00	6.73E + 00	1,41E+01	2.53E+01
CEC4	Best	3.18E – 01	5.45E – 01	2.32E + 00	5.34E – 01	1.90E + 00	**8.78E – 01**	5.27E – 01	4.81E – 01	7.78E – 01	8,72E-01	5.56E+00
	Median	9.82E – 01	1.41E + 00	4.53E + 00	1.22E + 00	1.03E + 01	1.62E + 00	1.39E + 00	1.06E + 00	1.40E + 00	2,68E+00	5.57E+00
	Mean	**1.00E + 00**	1.57E + 00	6.59E + 00	1.24E + 00	1.02E + 01	1.85E + 00	3.58E + 02	1.08E + 00	1.50E + 00	2,79E+00	6.22E+00
	Std. dev	2.52E – 01	7.46E – 01	5.71E + 00	3.92E – 01	3.92E + 00	7.27E – 01	1.94E + 03	3.28E – 01	5.38E – 01	9,69E-01	2.46E+00
CEC5	Best	9.78E + 01	2.82E + 01	3.38E + 02	7.12E + 01	1.20E + 03	5.12E + 02	7.66E + 01	**1.47E + 01**	1.34E + 02	1,15E+04	9.88E+02
	Median	3.29E + 02	3.22E + 03	7.42E + 02	4.50E + 02	6.53E + 03	1.42E + 03	6.00E + 02	1.34E + 02	7.30E + 02	3,64E+04	1.30E+03
	Mean	3.23E + 02	6.67E + 03	**1.28E + 03**	4.50E + 02	6.74E + 03	2.01E + 03	6.85E + 02	1.33E + 02	7.08E + 02	6,41E+04	1.50E+03
	Std. dev	1.48E + 02	9.55E + 03	3.00E + 03	1.78E + 02	2.34E + 03	1.83E + 03	4.09E + 02	1.05E + 02	2.57E + 02	6,63E+04	7.49E+02

Table 4.4 Continues

Problems	Algorithm	OOYO	YYPO	ESDA	ASO	HHO	SSA	AGPSO3	COA	CMAES	WSAR	p-WSAR
CEC6	Best	1.42E + 01	2.94E + 01	6.64E – 01	**0.00E + 00**	6.84E – 01	6.94E + 00	6.65E + 01	3.33E + 01	5.07E – 01	7,79E+01	1.38E+00
	Median	6.66E + 01	2.09E + 02	8.11E – 01	0.00E + 00	6.78E + 00	7.79E + 01	9.67E + 01	5.79E + 01	1.01E + 02	2,35E+02	1.38E+00
	Mean	9.49E + 01	1.73E + 02	7.41E + 01	**0.00E + 00**	3.27E + 01	1.53E + 02	1.23E + 02	6.15E + 01	1.59E + 02	2,70E+02	1.38E+00
	Std. dev	7.67E + 01	1.07E + 02	1.58E + 02	0.00E + 00	4.29E + 01	1.50E + 02	5.89E + 01	2.63E + 01	1.44E + 02	1,14E+02	2.80E-08
CEC7	Best	7.40E + 00	1.43E + 00	6.01E + 01	1.49E + 00	6.57E + 02	3.07E + 02	1.38E + 02	**4.34E – 01**	5.01E + 00	1,74E+03	1.26E+03
	Median	2.67E + 02	2.14E + 02	5.42E + 02	1.21E + 02	1.41E + 03	1.24E + 03	5.06E + 02	3.93E + 01	4.38E + 02	1,21E+04	1.56E+03
	Mean	2.45E + 02	2.06E + 02	6.47E + 02	1.08E + 02	1.64E + 03	1.30E + 03	5.40E + 02	**8.78E + 01**	4.70E + 02	2,29E+04	2.00E+03
	Std. dev	1.41E + 02	1.27E + 02	4.48E + 02	9.05E + 01	7.58E + 02	5.44E + 02	3.11E + 02	9.87E + 01	4.17E + 02	2,81E+04	1.13E+03
CEC8	Best	0.00E + 00	1.00E + 02	1.00E + 02	**0.00E + 00**	1.04E + 02	1.01E + 02	2.96E + 01	1.00E + 02	1.00E + 02	1,22E+02	1.12E+02
	Median	1.87E + 01	1.01E + 02	1.02E + 02	1.00E + 02	1.08E + 02	1.02E + 02	1.01E + 02	1.01E + 02	1.00E + 02	1,34E+02	1.12E+02
	Mean	**4.81E + 01**	1.01E + 02	1.05E + 02	5.49E + 01	3.42E + 02	1.03E + 02	1.42E + 02	1.43E + 02	1.00E + 02	4,81E+02	1.12E+02
	Std. dev	5.00E + 01	6.16E – 01	1.08E + 01	4.94E + 01	4.52E + 02	1.20E + 00	1.99E + 02	1.22E + 02	2.40E – 01	7,95E+02	1.04E-01
CEC9	Best	1.00E + 02	1.00E + 02	1.00E + 02	1.00E + 02	1.01E + 02	3.00E + 02	4.03E + 02	3.93E + 02	3.88E + 02	4,00E+02	**1.00E+02**
	Median	1.00E + 02	1.01E + 02	4.20E + 02	1.00E + 02	4.19E + 02	4.04E + 02	4.47E + 02	3.96E + 02	4.03E + 02	4,18E+02	1.00E+02
	Mean	1.10E + 02	1.69E + 02	4.12E + 02	2.20E + 02	3.98E + 02	4.01E + 02	4.50E + 02	3.97E + 02	4.05E + 02	4,22E+02	**1.01E+02**
	Std. dev	5.54E + 01	1.12E + 02	8.94E + 01	1.34E + 02	8.15E + 01	2.51E + 01	3.43E + 01	2.03E + 00	8.76E + 00	1,40E+01	2.99E-01
CEC10	Best	**4.00E + 02**	4.00E + 02	4.00E + 02	4.00E + 02	4.01E + 02	4.00E + 02	4.00E + 02	4.00E + 02	4.00E + 02	4,41E+02	4.31E+02
	Median	4.00E + 02	4.00E + 02	4.00E + 02	4.00E + 02	4.02E + 02	4.00E + 02	4.58E + 02	4.00E + 02	4.00E + 02	5,38E+02	4.32E+02
	Mean	**4.00E + 02**	4.15E + 02	4.34E + 02	4.00E + 02	4.27E + 02	4.04E + 02	4.74E + 02	4.17E + 02	4.36E + 02	5,45E+02	4.61E+02
	Std. dev	3.48E – 13	5.83E + 01	8.03E + 01	4.70E – 13	7.57E + 01	2.12E + 01	7.70E + 01	4.62E + 01	8.20E + 01	4,81E+01	7.22E+01

Table 4.5 Results on 20D case in CEC2020 dataset

Problems	Algorithm	OOYO	YYPO	ESDA	ASO	HHO	SSA	AGPSO3	COA	CMAES	WSAR	p-WSAR
CEC1	Best	8.70E + 00	5.33E – 02	0.00E + 00	1.32E + 01	2.78E + 05	1.08E + 00	2.64E-02	0.00E + 00	**0.00E + 00**	6,87E+08	3.18E+03
	Median	1.17E + 03	2.02E + 03	9.95E – 13	5.67E + 02	5.28E + 05	1.17E + 03	3.17E + 00	0.00E + 00	0.00E + 00	1,09E+09	3.38E+03
	Mean	2.58E + 03	3.38E + 03	1.06E + 03	1.04E + 03	5.20E + 05	1.82E + 03	3.37E + 08	0.00E + 00	**0.00E + 00**	1,19E+09	5.54E+08
	Std. dev	2.88E + 03	3.60E + 03	3.25E + 03	1.09E + 03	1.54E + 05	1.97E + 03	6.46E + 08	1.35E-14	1.09E-14	3,09E+08	1.13E+09
CEC2	Best	1.39E + 02	1.80E + 00	7.23E + 02	1.30E + 02	2.88E + 02	5.83E + 02	5.18E + 01	6.83E + 00	5.94E + 02	1,29E+03	7.88E+02
	Median	5.02E + 02	3.05E + 02	2.26E + 03	5.15E + 02	6.48E + 02	1.40E + 03	5.06E + 02	1.27E + 02	1.44E + 03	2,74E+03	7.89E+02
	Mean	5.31E + 02	3.56E + 02	2.26E + 03	5.51E + 02	6.41E + 02	1.39E + 03	5.64E + 02	1.12E + 02	1.46E + 03	2,69E+03	9.43E+02
	Std. dev	2.00E + 02	2.03E + 02	7.04E + 02	2.53E + 02	2.12E + 02	4.36E + 02	3.47E + 02	8.24E + 01	3.58E + 02	4,20E+02	2.00E+02
CEC3	Best	3.70E + 01	2.32E + 01	6.60E + 01	2.52E + 01	6.94E + 01	3.86E + 01	2.17E + 01	**3.58E + 00**	3.35E + 01	1,01E+02	1.96E+02
	Median	4.90E + 01	3.33E + 01	1.64E + 02	3.21E + 01	9.22E + 01	6.93E + 01	2.77E + 01	3.03E + 01	4.82E + 01	1,32E+02	1.97E+02
	Mean	4.89E + 01	3.47E + 01	2.32E + 02	3.26E + 01	9.38E + 01	7.10E + 01	2.83E + 01	3.05E + 01	4.95E + 01	1,32E+02	2.01E+02
	Std. dev	7.38E + 00	5.65E + 00	1.30E + 02	4.08E + 00	1.50E + 01	2.10E + 01	3.63E + 00	8.37E + 00	1.01E + 01	1,62E+01	1.44E+01
CEC4	Best	1.20E + 00	7.48E – 01	4.33E + 00	1.21E + 00	7.32E + 00	1.40E + 00	1.09E + 00	4.98E-01	1.46E + 00	3,37E+00	8.80E+00
	Median	1.76E + 00	1.81E + 00	1.15E + 01	1.82E + 00	1.52E + 01	2.41E + 00	2.55E + 00	1.19E + 00	2.24E + 00	5,26E+00	8.82E+00
	Mean	1.79E + 00	2.09E + 00	1.36E + 01	1.81E + 00	1.63E + 01	2.61E + 00	1.98E + 01	1.31E + 00	2.38E + 00	5,48E+00	1.15E+01
	Std. dev	3.71E – 01	9.70E – 01	8.21E + 00	3.31E – 01	6.20E + 00	1.00E + 00	9.28E + 01	5.08E-01	6.62E-01	1,89E+00	1.36E+01
CEC5	Best	4.64E + 02	1.00E + 03	6.96E + 02	1.85E + 02	2.24E + 03	1.01E + 03	1.55E + 03	1.44E + 02	2.57E + 02	1,72E+04	4.91E+03
	Median	1.15E + 03	2.00E + 04	2.79E + 03	7.71E + 02	2.07E + 04	1.83E + 03	4.54E + 03	4.40E + 02	1.12E + 03	1,47E+05	5.10E+03
	Mean	1.51E + 03	3.24E + 04	7.94E + 04	7.90E + 02	2.39E + 04	2.32E + 03	1.96E + 04	4.30E + 02	1.07E + 03	3,87E+05	6.34E+03
	Std. dev	9.80E + 02	3.57E + 04	2.21E + 05	3.08E + 02	1.65E + 04	1.19E + 03	6.21E + 04	1.40E + 02	4.13E + 02	5,63E+05	4.66E+03

Table 4.5 Continues

Problems	Algorithm	OOYO	YYPO	ESDA	ASO	HHO	SSA	AGPSO3	COA	CMAES	WSAR	p-WSAR
CEC6	Best	9.60E + 01	4.91E + 01	2.44E + 01	**0.00E + 00**	9.03E + 01	3.52E + 00	2.44E + 01	2.44E + 01	5.17E-01	3,21E+01	1.28E+01
	Median	2.28E + 02	4.26E + 02	1.67E + 02	0.00E + 00	2.28E + 02	1.10E + 02	5.25E + 01	7.64E + 01	7.61E + 01	5,39E+01	1.55E+02
	Mean	2.85E + 02	4.08E + 02	1.77E + 02	**0.00E + 00**	2.28E + 02	2.05E + 02	1.14E + 02	1.30E + 02	1.14E + 02	1,13E+02	1.51E+02
	Std. dev	1.85E + 02	2.19E + 02	1.35E + 02	0.00E + 00	1.13E + 02	2.10E + 02	1.54E + 02	1.47E + 02	1.41E + 02	9,45E+01	1.06E+02
CEC7	Best	2.46E + 02	3.33E + 02	4.37E + 02	3.31E + 01	1.17E + 03	6.65E + 02	1.43E + 02	**3.83E-01**	1.85E + 02	2,26E+04	7.00E+02
	Median	5.51E + 02	4.17E + 03	8.92E + 02	2.02E + 02	3.05E + 03	1.68E + 03	1.07E + 03	1.53E + 02	4.75E + 02	9,90E+04	2.03E+03
	Mean	5.61E + 02	5.27E + 03	1.77E + 03	2.12E + 02	3.48E + 03	1.68E + 03	1.02E + 04	6.73E + 02	5.08E + 02	1,48E+05	2.65E+03
	Std. dev	2.15E + 02	6.10E + 03	2.73E + 03	1.06E + 02	1.64E + 03	6.61E + 02	4.16E + 04	2.47E + 03	2.09E + 02	1,49E+05	1.65E+03
CEC8	Best	0.00E + 00	6.17E + 01	1.00E + 02	**0.00E + 00**	1.08E + 02	5.41E + 01	1.00E + 02	1.00E + 02	1.00E + 02	1,60E+02	1.04E+02
	Median	1.00E + 02	1.00E + 02	1.02E + 02	1.00E + 02	1.10E + 02	1.02E + 02	1.70E + 02	1.00E + 02	1.00E + 02	2,36E+02	1.07E+02
	Mean	9.15E + 01	9.95E + 01	1.02E + 02	**6.06E + 01**	9.59E + 02	9.99E + 01	2.47E + 02	4.32E + 02	1.00E + 02	1,39E+03	1.10E+02
	Std. dev	2.77E + 01	7.22E + 00	1.65E + 00	4.71E + 01	1.34E + 03	8.71E + 00	3.12E + 02	6.20E + 02	2.10E-01	1,58E+03	9.52E+00
CEC9	Best	**1.00E + 02**	1.00E + 02	4.34E + 02	3.00E + 02	5.50E + 02	3.32E + 02	1.00E + 02	4.35E + 02	4.05E + 02	4,60E+02	1.07E+02
	Median	4.14E + 02	4.52E + 02	4.82E + 02	4.10E + 02	6.82E + 02	7.02E + 02	4.96E + 02	4.50E + 02	4.26E + 02	4,95E+02	4.71E+02
	Mean	3.60E + 02	4.28E + 02	4.84E + 02	4.05E + 02	6.91E + 02	8.03E + 02	5.02E + 02	4.53E + 02	4.27E + 02	4,92E+02	3.77E+02
	Std. dev	1.19E + 02	1.15E + 02	3.41E + 01	6.86E + 01	8.64E + 01	1.71E + 01	9.37E + 01	1.23E + 01	1.36E + 01	1,56E+01	1.66E+02
CEC10	Best	4.10E + 02	3.99E + 02	4.01E + 02	3.99E + 02	4.00E + 02	4.02E + 02	4.10E + 02	4.03E + 02	4.10E + 02	4,40E+02	4.15E+02
	Median	4.14E + 02	4.14E + 02	4.64E + 02	4.14E + 02	4.68E + 02	4.14E + 02	4.14E + 02	4.14E + 02	4.15E + 02	4,60E+02	4.31E+02
	Mean	4.13E + 02	4.26E + 02	4.60E + 02	4.22E + 02	4.61E + 02	4.21E + 02	4.32E + 02	4.23E + 02	4.32E + 02	4,73E+02	4.41E+02
	Std. dev	1.39E + 00	2.71E + 01	3.41E + 01	2.05E + 01	3.70E + 01	1.84E + 01	4.59E + 01	2.05E + 01	2.75E + 01	3,12E+01	1.91E+01

Table 4.6 Friedman and Wilcoxon signed-rank test results on 5D case

Average rankings achieved by Friedman test		Wilcoxon signed-rank test between p-WSAR and other algorithms	
Algorithms	Sum of Ranks	p-WSAR vs.	p-value
OOYO	4.3125 (4)	OOYO	0.0078
YYPO	3.8750 (3)	YYPO	0.0078
ESDA	7.3125 (8)	ESDA	0.0078
ASO	6.1875 (5)	ASO	0.0078
HHO	10.1250 (11)	HHO	0.0078
SSA	7.8750 (9)	SSA	0.0078
AGPSO3	6.2500 (6)	AGPSO3	0.0078
COA	3.000 (2)	COA	0.0547
CMAES	8.8125 (10)	CMAES	0.0156
WSAR	7.0000 (7)	WSAR	0.0078
p-WSAR	1.2500 (1)		

Table 4.7 Friedman and Wilcoxon signed-rank test results on 10D case

Average rankings achieved by Friedman test		Wilcoxon signed-rank test between p-WSAR and other algorithms	
Algorithms	Sum of Ranks	p-WSAR vs.	p-value
OOYO	3.5000 (3)	OOYO	0.0039
YYPO	5.2000 (5)	YYPO	0.0020
ESDA	9.1500 (9)	ESDA	0.0020
ASO	3.4500 (2)	ASO	0.0039
HHO	9.3000 (10)	HHO	0.0020
SSA	6.5500 (7)	SSA	0.0020
AGPSO3	6.3500 (6)	AGPSO3	0.0039
COA	4.200 (4)	COA	0.0645
CMAES	7.0000 (8)	CMAES	0.0273
WSAR	9.6000 (11)	WSAR	0.0020
p-WSAR	1.700 (1)		

Table 4.8 Friedman and Wilcoxon signed-rank test results on 15D case

Average rankings achieved by Friedman test		Wilcoxon signed-rank test between p-WSAR and other algorithms	
Algorithms	Sum of Ranks	p-WSAR vs.	p-value
OOYO	3.5500 (4)	OOYO	0.2754
YYPO	5.9000 (5)	YYPO	0.0645
ESDA	7.8000 (10)	ESDA	0.0059
ASO	3.1500 (1)	ASO	0.4922
HHO	7.2000 (7)	HHO	0.0137
SSA	7.2000 (7)	SSA	0.0020
AGPSO3	7.7000 (9)	AGPSO3	0.0039
COA	3.4500 (3)	COA	0.9219
CMAES	6.4500 (6)	CMAES	0.0059
WSAR	10.4000 (11)	WSAR	0.0020
p-WSAR	3.2000 (2)		

Table 4.9 Friedman and Wilcoxon signed-rank test results on 20D case

Average rankings achieved by Friedman test		Wilcoxon signed-rank test between p-WSAR and other algorithms	
Algorithms	Sum of Ranks	p-WSAR vs.	p-value
OOYO	4.2000 (3)	OOYO	0.2324
YYPO	5.9000 (4)	YYPO	0.1309
ESDA	6.2000 (5)	ESDA	0.0020
ASO	8.1000 (9)	ASO	0.8457
HHO	7.2000 (7)	HHO	0.0195
SSA	2.7000 (1)	SSA	0.0020
AGPSO3	7.7000 (9)	AGPSO3	0.0840
COA	8.9000 (10)	COA	0.8457
CMAES	6.9000 (6)	CMAES	0.0273
WSAR	7.3000 (8)	WSAR	0.0020
p-WSAR	3.7500 (2)		

For representing results in a better way, boxplots are provided for the results obtained from various metaheuristic algorithms based on the average performances in

Figure 4.1- 4.4. As it can be seen in the figures, p-WSAR has the lowest median and the most compact box.

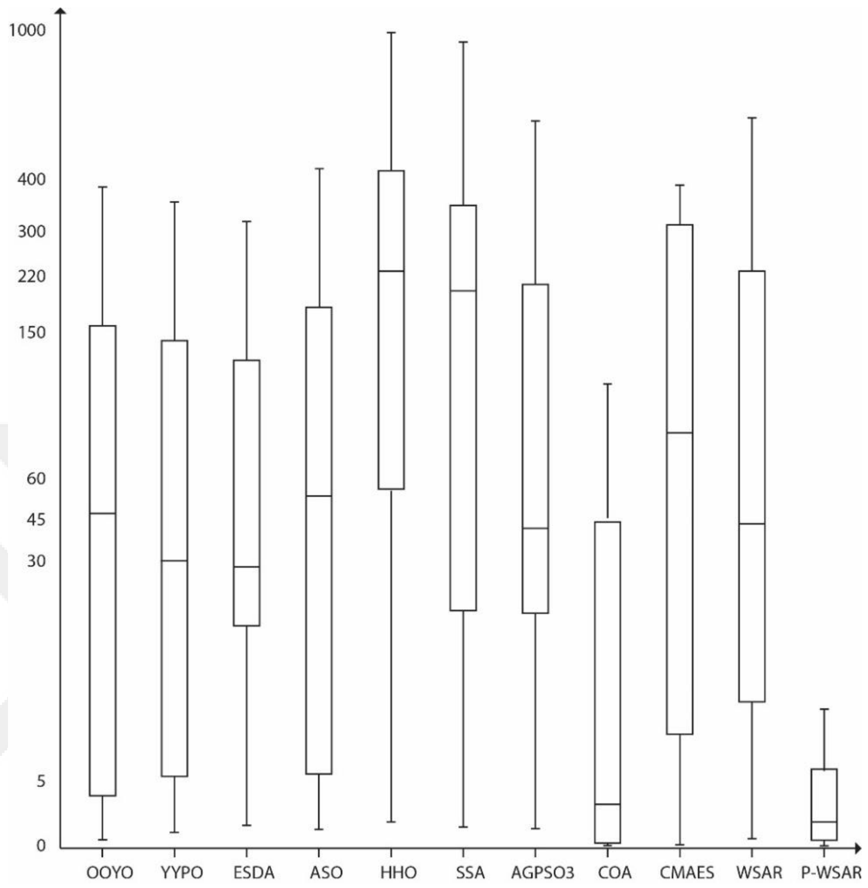


Figure 4.1 Boxplot of CEC 5D average results

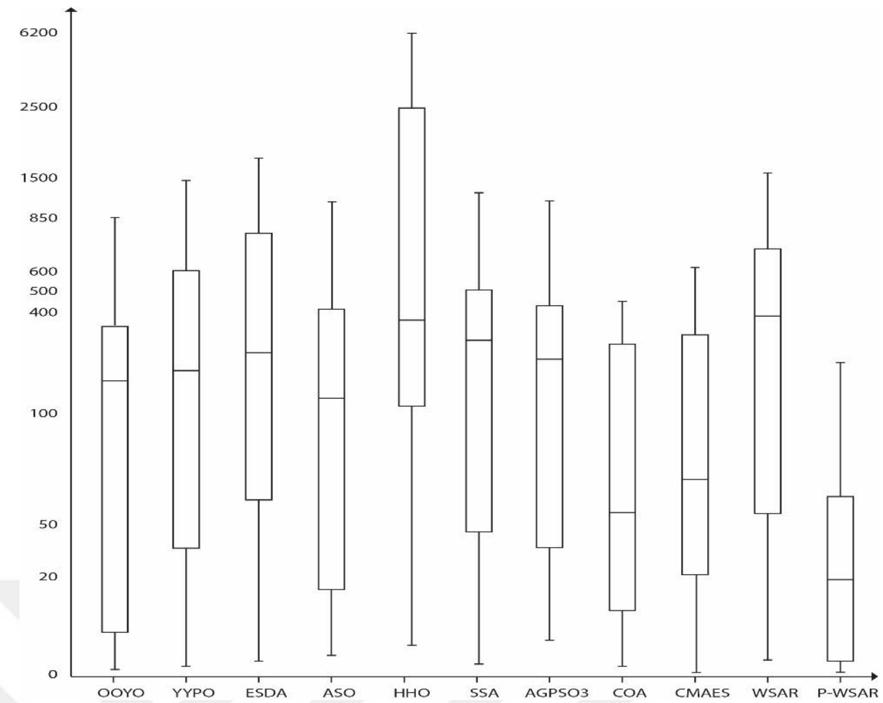


Figure 4.2 Boxplot of CEC 10D average results

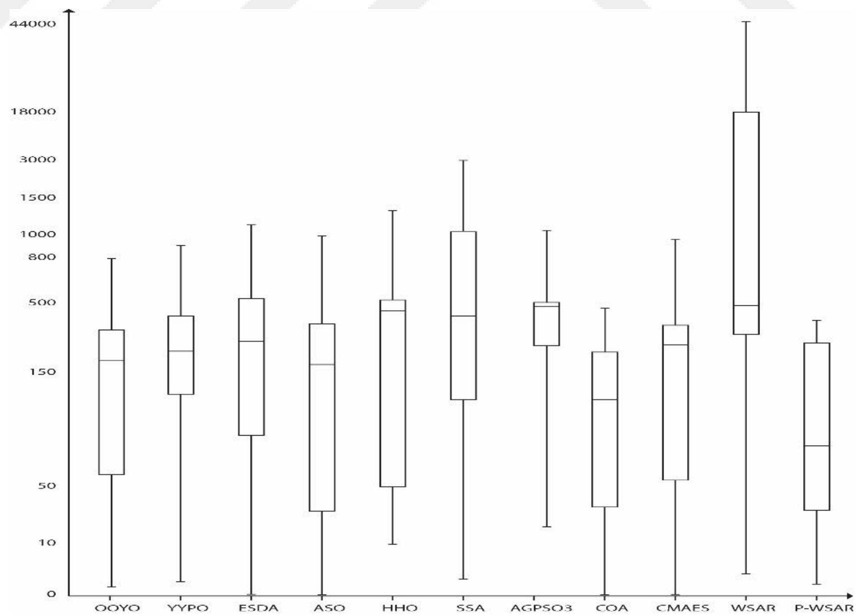


Figure 4.3 Boxplot of CEC 15D average results

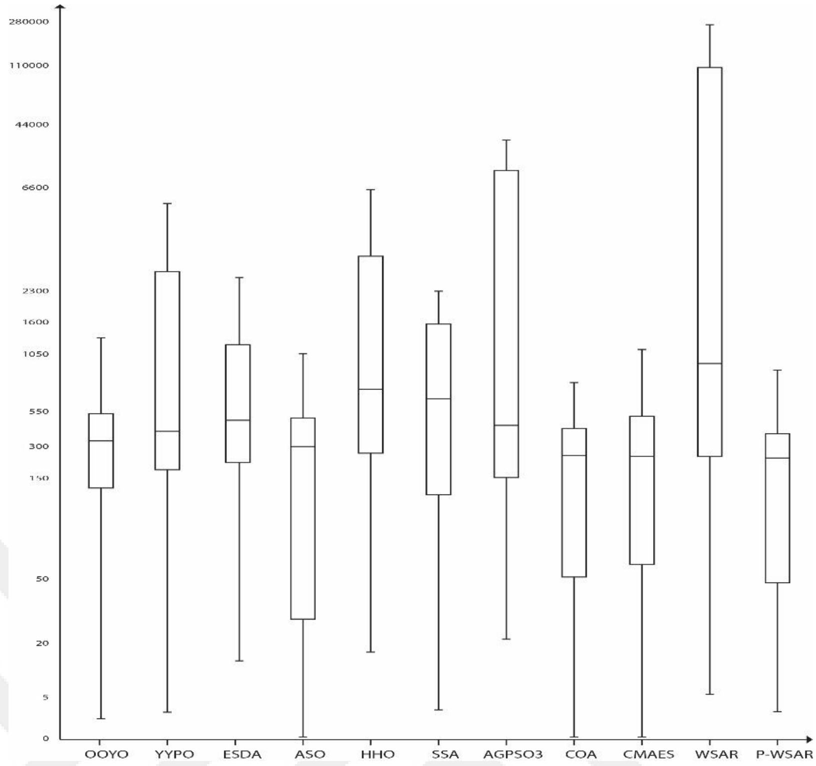


Figure 4.4 Boxplot of CEC 20D average results

Additional computational study is also performed in order to compare the performance of p-WSAR and WSAR algorithms on CEC2020 problems against each SSBMAs (that are used as search algorithms within p-WSAR) in respect of solution quality. Computational results are presented in Tables 4.10- 4.17.

Table 4.10 shows that p-WSAR can find close results to the global optimum on all instances in the 5D benchmark dataset. On the other hand, performance of the SSBMAs and WSAR is considerable behind the performance of the p-WSAR. Table 4.14 indicates that (from Friedman test) p-WSAR performs the best among other algorithms. In addition, based on the Wilcoxon signed-rank test, the p-WSAR performs better than and GD, GS, RS, SA, TA, and WSAR as the $p<0.05$.

Table 4.11 presents that p-WSAR is capable of finding the best solutions on CEC1, CEC2, CEC6, CEC7, CEC8, CEC9 and CEC10 instances in the 10D benchmark dataset. It also has the lowest mean value on all instances. Table 4.15

reveals that (based on the Friedman test results) p-WSAR performs the best among other algorithms. Moreover, based on the Wilcoxon signed-rank test, the p-WSAR performs better than and GD, GS, RS, SA, TA, and WSAR as the $p<0.05$.

Table 4.12 reveals that p-WSAR can find the best results on CEC1, CEC2, CEC5, CEC6, CEC7, CEC8, CEC9, and CEC10 instances the 15D benchmark dataset. Table 4.16 presents that p-WSAR performs the best among the other algorithms. In addition, the p-WSAR performs better than GD, GS, RS, SA, TA and WSAR as the $p<0.05$. Finally, Table 4.13 reveals that p-WSAR can find the best results on CEC1, CEC2, CEC5, CEC6, CEC7, CEC8, CEC9, and CEC10 instances the 20D benchmark dataset. According to the Friedman test results, Table 4.17 indicates that p-WSAR performs the best among other compared algorithms. Furthermore, based on the Wilcoxon signed-rank test, p-WSAR performs better than GD, GS, RS, SA, TA and WSAR as $p<0.1$. These results show that even though SSBMAs' individual performances are very poor, they can present considerable higher and competitive performance when they come together within a coalitional framework like p-WSAR.

Boxplots are also given for the results of SSBMAs, which are calculated based on the average performances in Figure 4.5-4.8. As it can be seen in the figures, p-WSAR's median is much lower than SSBMAs and PWSAR's box is the most compact one among others.

Table 4.10 5D case results of SSBMAs

Problems	Algorithm	GD	GS	RS	SA	TA	WSAR	p-WSAR
CEC1	Best	2,59E+03	9,87E+08	4,57E+06	4,71E+08	9,76E+02	3.59E+00	3.58E-01
	Median	1,48E+09	1,09E+10	3,26E+07	9,36E+09	3,84E+09	7.54E+03	3.59E-01
	Mean	2,27E+09	1,19E+10	3,23E+07	1,14E+10	2,97E+09	2.95E+04	3.61E-01
	Std. dev	1,94E+09	8,18E+09	1,70E+07	8,00E+09	1,87E+09	8.80E+04	3.13E-03
CEC2	Best	4,31E+02	1,33E+03	1,55E+02	5,53E+02	3,42E+02	2.71E+01	6.70E+00
	Median	7,38E+02	1,95E+03	3,09E+02	1,12E+03	9,04E+02	6.50E+01	6.70E+00
	Mean	8,44E+02	2,02E+03	2,89E+02	1,18E+03	9,06E+02	6.70E+01	6.70E+00
	Std. dev	3,03E+02	4,38E+02	7,31E+01	2,58E+02	4,17E+02	3.08E+01	4.10E-06
CEC3	Best	1,14E+02	1,11E+02	1,86E+01	7,24E+01	4,81E+01	5.23E+00	4.63E+00
	Median	3,14E+02	3,86E+02	2,88E+01	2,34E+02	3,22E+02	7.35E+00	4.63E+00
	Mean	3,39E+02	3,82E+02	2,92E+01	2,56E+02	3,35E+02	7.40E+00	4.63E+00
	Std. dev	1,33E+02	1,44E+02	5,51E+00	1,23E+02	1,33E+02	1.27E+00	1.40E-07
CEC4	Best	7,30E+01	1,48E+05	1,65E+00	4,92E+04	8,86E+00	3.95E-02	1.09E-01
	Median	5,96E+04	2,06E+07	3,68E+00	8,87E+06	4,10E+04	1.58E-01	1.09E-01
	Mean	3,78E+06	4,91E+08	3,66E+00	5,89E+07	2,41E+06	1.58E-01	1.14E-01
	Std. dev	4,49E+06	1,36E+09	7,34E-01	1,44E+08	3,76E+06	7.55E-02	3.21E-02
CEC5	Best	3,85E+01	3,60E+06	2,14E+02	4,43E+03	5,61E+00	1.09E+00	8.68E-08
	Median	1,04E+04	2,49E+09	5,54E+02	7,97E+08	9,81E+08	3.27E+00	1.16E-07
	Mean	4,80E+08	4,25E+09	6,48E+02	2,35E+09	5,78E+08	9.79E+00	1.71E-01
	Std. dev	5,23E+08	4,52E+09	3,71E+02	3,30E+09	5,15E+08	2.61E+01	6.20E-01

Table 4.10 Continues

CEC8	Best	3,48E+02	4,80E+01	2,57E+01	5,89E+02	5,53E+01	1.02E-04	1.07E-05
	Median	1,29E+03	2,42E+03	3,29E+01	1,58E+03	1,20E+03	2.49E-04	1.07E-05
	Mean	1,27E+03	2,25E+03	3,28E+01	1,59E+03	1,11E+03	2.11E+01	2.77E-05
	Std. dev	5,24E+02	7,81E+02	4,25E+00	4,92E+02	5,19E+02	4.07E+01	2.46E-05
CEC9	Best	1,06E+02	3,64E+02	1,16E+02	3,10E+02	1,36E+02	5.10E+01	7.92E-04
	Median	4,28E+02	1,34E+03	1,34E+02	9,68E+02	5,13E+02	1.00E+02	8.85E-04
	Mean	4,75E+02	1,50E+03	1,33E+02	1,33E+03	4,66E+02	1.87E+02	7.22E+00
	Std. dev	4,01E+02	9,81E+02	9,18E+00	1,01E+03	2,60E+02	1.13E+02	1.41E+01
CEC10	Best	3,00E+02	4,81E+02	3,27E+02	4,42E+02	2,49E+02	3.01E+02	6.20E-04
	Median	4,14E+02	2,88E+03	3,54E+02	1,87E+03	4,78E+02	3.47E+02	6.20E-04
	Mean	4,85E+02	4,33E+03	3,53E+02	2,53E+03	5,89E+02	3.46E+02	1.00E+01
	Std. dev	2,01E+02	4,07E+03	5,87E+00	2,06E+03	2,50E+02	8.48E+00	5.48E+01

Table 4.11 10D case results of SSBMAs

Problems	Algorithm	GD	GS	RS	SA	TA	WSAR	p-WSAR
CEC1	Best	2,98E+09	1,27E+10	8,18E+08	1,95E+10	3,89E+09	2,21E+06	1.71E+02
	Median	1,49E+10	6,60E+10	2,09E+09	6,21E+10	1,38E+10	3,80E+07	1.75E+02
	Mean	1,61E+10	7,64E+10	2,03E+09	6,18E+10	1,47E+10	3,93E+07	1.81E+02
	Std. dev	7,01E+09	3,61E+10	6,74E+08	2,53E+10	6,48E+09	1,86E+07	2.70E+01
CEC2	Best	7,04E+02	3,18E+03	8,15E+02	1,94E+03	1,12E+03	2,74E+02	2.18E+01
	Median	1,85E+03	4,09E+03	1,18E+03	2,65E+03	1,85E+03	7,76E+02	2.18E+01
	Mean	1,81E+03	4,04E+03	1,17E+03	2,61E+03	1,87E+03	7,16E+02	5.73E+01
	Std. dev	4,88E+02	4,73E+02	1,39E+02	3,69E+02	4,68E+02	1,63E+02	9.42E+01
CEC3	Best	3,18E+02	4,55E+02	1,07E+02	3,10E+02	3,90E+02	1,81E+01	2.38E+01
	Median	9,40E+02	9,57E+02	1,49E+02	7,60E+02	8,97E+02	2,74E+01	2.38E+01
	Mean	9,22E+02	9,75E+02	1,49E+02	7,74E+02	9,24E+02	2,82E+01	3.31E+01
	Std. dev	2,76E+02	2,76E+02	1,65E+01	2,24E+02	2,97E+02	5,81E+00	1.35E+01
CEC4	Best	9,22E+03	3,87E+06	2,48E+01	8,56E+05	1,15E+02	4,50E-01	3.33E+00
	Median	1,73E+06	1,93E+08	3,44E+02	6,28E+07	1,20E+06	9,01E-01	3.46E+00
	Mean	1,74E+06	3,88E+08	4,50E+02	1,88E+08	1,02E+06	1,10E+00	3.51E+00
	Std. dev	1,09E+06	4,23E+08	4,44E+02	3,23E+08	8,23E+05	6,01E-01	1.82E-01
CEC5	Best	5,92E+02	3,50E+07	4,29E+03	1,80E+06	3,88E+02	4,13E+02	5.31E+02
	Median	6,36E+05	1,56E+09	1,92E+04	1,83E+09	7,60E+05	1,60E+03	5.77E+02
	Mean	5,56E+06	2,56E+09	2,43E+04	3,60E+09	6,03E+06	4,49E+03	7.88E+02
	Std. dev	2,00E+07	3,88E+09	1,57E+04	4,74E+09	1,25E+07	5,73E+03	4.06E+02

Table 4.11 Continues

CEC6	Best	1,26E+02	1,22E+03	1,09E+02	8,57E+02	1,21E+02	5,28E+00	1.50E+00
	Median	7,90E+02	1,05E+04	1,94E+02	1,31E+04	8,44E+02	2,63E+01	1.50E+00
	Mean	1,90E+03	2,02E+04	1,94E+02	1,64E+04	1,86E+03	4,28E+01	1.59E+00
	Std. dev	2,10E+03	2,57E+04	3,82E+01	1,77E+04	2,04E+03	4,55E+01	2.51E-01
CEC7	Best	1,06E+03	2,21E+08	1,15E+03	1,37E+04	4,68E+02	6,70E+01	1.63E+02
	Median	3,38E+07	4,55E+09	3,86E+03	9,27E+08	8,24E+06	3,79E+02	2.28E+02
	Mean	1,15E+09	7,04E+09	4,92E+03	2,54E+09	6,16E+08	4,14E+02	2.73E+02
	Std. dev	1,36E+09	6,73E+09	3,67E+03	3,65E+09	1,09E+09	2,22E+02	1.03E+02
CEC8	Best	4,61E+02	2,23E+03	1,23E+02	1,32E+03	6,40E+02	1,54E+01	2.60E+01
	Median	2,30E+03	4,38E+03	1,85E+02	3,10E+03	2,35E+03	1,05E+02	2.60E+01
	Mean	2,11E+03	4,32E+03	1,86E+02	3,03E+03	2,38E+03	1,00E+02	3.27E+01
	Std. dev	6,61E+02	7,15E+02	2,77E+01	5,09E+02	6,94E+02	2,12E+01	9.03E+00
CEC9	Best	1,84E+02	6,47E+02	1,95E+02	5,33E+02	1,58E+02	3,42E+02	1.00E+02
	Median	7,15E+02	1,46E+03	2,69E+02	1,33E+03	6,37E+02	3,51E+02	1.00E+02
	Mean	7,27E+02	1,54E+03	2,65E+02	1,46E+03	6,67E+02	3,51E+02	1.00E+02
	Std. dev	2,43E+02	6,52E+02	2,67E+01	5,90E+02	2,30E+02	5,40E+00	1.97E+00
CEC10	Best	5,46E+02	2,75E+03	4,70E+02	2,04E+03	4,65E+02	4,01E+02	1.08E+02
	Median	1,03E+03	1,01E+04	5,30E+02	7,23E+03	8,10E+02	4,21E+02	1.40E+02
	Mean	1,17E+03	1,03E+04	5,29E+02	7,83E+03	9,48E+02	4,31E+02	2.24E+02
	Std. dev	5,18E+02	4,86E+03	3,11E+01	4,61E+03	4,86E+02	2,65E+01	1.37E+02

Table 4.12 15D case results of SSBMAs

Problems	Algorithm	GD	GS	RS	SA	TA	WSAR	p-WSAR
CEC1	Best	2,09E+09	4,80E+10	4,73E+09	3,40E+10	3,28E+09	1,08E+08	8.88E+02
	Median	2,46E+10	8,44E+10	7,06E+09	9,69E+10	2,48E+10	4,02E+08	9.25E+02
	Mean	2,52E+10	9,62E+10	7,30E+09	9,25E+10	2,55E+10	4,05E+08	9.27E+02
	Std. dev	1,27E+10	3,61E+10	1,67E+09	3,87E+10	1,23E+10	1,45E+08	4.74E+01
CEC2	Best	2,00E+03	4,27E+03	2,13E+03	2,74E+03	1,46E+03	8,09E+02	5.89E+02
	Median	2,74E+03	6,21E+03	2,48E+03	4,17E+03	2,65E+03	1,77E+03	5.89E+02
	Mean	2,83E+03	6,01E+03	2,47E+03	4,11E+03	2,57E+03	1,81E+03	5.93E+02
	Std. dev	4,66E+02	9,12E+02	1,67E+02	5,53E+02	5,56E+02	3,73E+02	6.75E+00
CEC3	Best	5,93E+02	1,12E+03	2,96E+02	8,15E+02	5,77E+02	5,71E+01	1.16E+02
	Median	1,61E+03	1,85E+03	3,90E+02	1,33E+03	1,87E+03	8,06E+01	1.68E+02
	Mean	1,61E+03	1,97E+03	3,80E+02	1,39E+03	1,71E+03	8,04E+01	1.49E+02
	Std. dev	5,08E+02	6,06E+02	3,45E+01	3,42E+02	5,63E+02	1,41E+01	2.53E+01
CEC4	Best	2,86E+03	2,72E+06	1,19E+03	4,81E+05	1,32E+05	8,72E-01	5.56E+00
	Median	9,23E+05	1,80E+08	9,94E+03	5,39E+07	7,80E+05	2,68E+00	5.57E+00
	Mean	9,60E+05	3,93E+08	1,01E+04	1,76E+08	1,21E+06	2,79E+00	6.22E+00
	Std. dev	7,70E+05	5,72E+08	5,07E+03	2,49E+08	8,50E+05	9,69E-01	2.46E+00
CEC5	Best	1,18E+03	4,59E+08	1,97E+05	3,87E+06	7,26E+05	1,15E+04	9.88E+02
	Median	4,46E+08	4,85E+09	5,58E+05	1,91E+09	3,07E+08	3,64E+04	1.30E+03
	Mean	1,03E+09	9,12E+09	5,70E+05	5,64E+09	1,23E+09	6,41E+04	1.50E+03
	Std. dev	1,47E+09	9,78E+09	2,04E+05	7,17E+09	1,64E+09	6,63E+04	7.49E+02

Table 4.12 Continues

CEC6	Best	1,79E+02	1,62E+03	4,58E+02	1,48E+03	2,77E+02	7,79E+01	1.38E+00
	Median	1,20E+03	5,89E+03	6,42E+02	6,75E+03	1,07E+03	2,35E+02	1.38E+00
	Mean	1,13E+03	1,12E+04	6,31E+02	9,82E+03	1,10E+03	2,70E+02	1.38E+00
	Std. dev	4,64E+02	1,22E+04	1,00E+02	9,59E+03	3,95E+02	1,14E+02	2.80E-08
CEC7	Best	8,52E+02	1,63E+07	5,47E+04	8,74E+06	1,70E+04	1,74E+03	1.26E+03
	Median	6,31E+06	2,08E+09	1,79E+05	1,31E+09	9,66E+06	1,21E+04	1.56E+03
	Mean	1,83E+07	3,11E+09	1,81E+05	2,76E+09	1,82E+07	2,29E+04	2.00E+03
	Std. dev	3,96E+07	3,91E+09	7,89E+04	3,59E+09	2,36E+07	2,81E+04	1.13E+03
CEC8	Best	1,93E+03	5,02E+03	4,09E+02	3,85E+03	9,45E+02	1,22E+02	1.12E+02
	Median	3,34E+03	6,28E+03	8,84E+02	4,99E+03	3,27E+03	1,34E+02	1.12E+02
	Mean	3,41E+03	6,38E+03	8,38E+02	5,00E+03	3,25E+03	4,81E+02	1.12E+02
	Std. dev	9,22E+02	8,51E+02	2,26E+02	6,52E+02	9,74E+02	7,95E+02	1.04E-01
CEC9	Best	2,38E+02	1,09E+03	3,72E+02	6,92E+02	2,03E+02	4,00E+02	1.00E+02
	Median	1,18E+03	3,08E+03	5,89E+02	2,55E+03	1,15E+03	4,18E+02	1.00E+02
	Mean	1,18E+03	3,20E+03	5,76E+02	2,98E+03	1,12E+03	4,22E+02	1.01E+02
	Std. dev	3,17E+02	1,48E+03	6,55E+01	1,30E+03	2,48E+02	1,40E+01	2.99E-01
CEC10	Best	7,47E+02	8,18E+03	1,03E+03	5,36E+03	6,32E+02	4,41E+02	4.31E+02
	Median	2,57E+03	2,37E+04	1,28E+03	1,45E+04	2,24E+03	5,38E+02	4.32E+02
	Mean	2,50E+03	3,04E+04	1,28E+03	2,14E+04	1,99E+03	5,45E+02	4.61E+02
	Std. dev	6,66E+02	2,04E+04	1,38E+02	1,89E+04	8,47E+02	4,81E+01	7.22E+01

Table 4.13 20D case results of SSBMAs

Problems	Algorithm	GD	GS	RS	SA	TA	WSAR	p-WSAR
CEC1	Best	2,08E+09	6,53E+10	9,72E+09	3,57E+10	9,98E+09	6,87E+08	3.18E+03
	Median	2,09E+10	1,26E+11	1,55E+10	7,96E+10	2,57E+10	1,09E+09	3.38E+03
	Mean	2,17E+10	1,33E+11	1,53E+10	8,02E+10	2,52E+10	1,19E+09	5.54E+08
	Std. dev	9,64E+09	4,03E+10	1,96E+09	1,91E+10	7,42E+09	3,09E+08	1.13E+09
CEC2	Best	1,78E+03	5,65E+03	3,04E+03	3,85E+03	1,66E+03	1,29E+03	7.88E+02
	Median	3,28E+03	7,53E+03	3,69E+03	5,28E+03	3,65E+03	2,74E+03	7.89E+02
	Mean	3,29E+03	7,70E+03	3,64E+03	5,25E+03	3,62E+03	2,69E+03	9.43E+02
	Std. dev	6,84E+02	1,09E+03	2,32E+02	6,08E+02	8,07E+02	4,20E+02	2.00E+02
CEC3	Best	1,03E+03	1,27E+03	5,04E+02	7,31E+02	1,35E+03	1,01E+02	1.96E+02
	Median	2,01E+03	2,53E+03	6,24E+02	1,11E+03	2,20E+03	1,32E+02	1.97E+02
	Mean	2,07E+03	2,57E+03	6,20E+02	1,16E+03	2,24E+03	1,32E+02	2.01E+02
	Std. dev	5,35E+02	5,53E+02	4,90E+01	2,31E+02	5,27E+02	1,62E+01	1.44E+01
CEC4	Best	1,78E+03	7,31E+06	6,15E+03	1,82E+06	4,49E+03	3,37E+00	8.80E+00
	Median	1,40E+06	3,37E+08	3,99E+04	3,31E+07	7,98E+05	5,26E+00	8.82E+00
	Mean	1,04E+07	1,45E+09	4,22E+04	1,60E+08	7,84E+06	5,48E+00	1.15E+01
	Std. dev	1,52E+07	2,44E+09	2,16E+04	3,49E+08	1,31E+07	1,89E+00	1.36E+01
CEC5	Best	1,60E+04	8,75E+07	5,02E+05	8,73E+06	4,53E+03	1,72E+04	4.91E+03
	Median	1,49E+06	2,70E+09	1,45E+06	1,40E+08	2,17E+06	1,47E+05	5.10E+03
	Mean	5,39E+06	3,75E+09	1,48E+06	4,55E+08	7,05E+06	3,87E+05	6.34E+03
	Std. dev	1,02E+07	4,05E+09	5,34E+05	6,87E+08	8,63E+06	5,63E+05	4.66E+03

Table 4.13 Continues

CEC6	Best	2,88E+02	2,39E+03	7,40E+02	2,18E+03	5,76E+02	3,21E+01	1.28E+01
	Median	2,31E+03	1,11E+04	1,04E+03	6,42E+03	1,94E+03	5,39E+01	1.55E+02
	Mean	2,17E+03	1,79E+04	1,04E+03	9,66E+03	2,08E+03	1,13E+02	1.51E+02
	Std. dev	8,84E+02	1,75E+04	1,30E+02	8,49E+03	9,48E+02	9,45E+01	1.06E+02
CEC7	Best	3,16E+03	6,67E+07	3,75E+04	1,82E+06	2,02E+03	2,26E+04	7.00E+02
	Median	2,59E+06	4,00E+09	2,90E+05	4,21E+08	4,83E+06	9,90E+04	2.03E+03
	Mean	5,45E+07	5,84E+09	2,97E+05	1,71E+09	2,40E+07	1,48E+05	2.65E+03
	Std. dev	1,43E+08	5,00E+09	1,37E+05	2,81E+09	4,39E+07	1,49E+05	1.65E+03
CEC8	Best	3,29E+03	7,27E+03	1,10E+03	4,52E+03	5,56E+02	1,60E+02	1.04E+02
	Median	4,46E+03	8,70E+03	1,94E+03	5,84E+03	4,06E+03	2,36E+02	1.07E+02
	Mean	4,75E+03	8,77E+03	1,90E+03	5,76E+03	4,09E+03	1,39E+03	1.10E+02
	Std. dev	1,06E+03	9,10E+02	3,27E+02	6,22E+02	8,87E+02	1,58E+03	9.52E+00
CEC9	Best	2,08E+09	6,53E+10	9,72E+09	3,57E+10	9,98E+09	4,60E+02	1.07E+02
	Median	2,09E+10	1,26E+11	1,55E+10	7,96E+10	2,57E+10	4,95E+02	4.71E+02
	Mean	2,17E+10	1,33E+11	1,53E+10	8,02E+10	2,52E+10	4,92E+02	3.77E+02
	Std. dev	9,64E+09	4,03E+10	1,96E+09	1,91E+10	7,42E+09	1,56E+01	1.66E+02
CEC10	Best	1,78E+03	5,65E+03	3,04E+03	3,85E+03	1,66E+03	4,40E+02	4.15E+02
	Median	3,28E+03	7,53E+03	3,69E+03	5,28E+03	3,65E+03	4,60E+02	4.31E+02
	Mean	3,29E+03	7,70E+03	3,64E+03	5,25E+03	3,62E+03	4,73E+02	4.41E+02
	Std. dev	6,84E+02	1,09E+03	2,32E+02	6,08E+02	8,07E+02	3,12E+01	1.91E+01

Table 4.14 Friedman and Wilcoxon signed-rank test results on 5D case (SSBMAs)

Average rankings achieved by Friedman test		Wilcoxon signed-rank test between p-WSAR and other algorithms	
Algorithms	Sum of Ranks	p-WSAR vs.	p-value
GD	4.6250 (4)	GD	0.0078
GS	7.0000 (7)	GS	0.0078
RS	2.8750 (3)	RS	0.0078
SA	5.7500 (6)	SA	0.0078
TA	4.6250 (4)	TA	0.0078
WSAR	2.1250 (2)	WSAR	0.0078
p-WSAR	1.0000 (1)		

Table 4.15 Friedman and Wilcoxon signed-rank test results on 10D case (SSBMAs)

Average rankings achieved by Friedman test		Wilcoxon signed-rank test between p-WSAR and other algorithms	
Algorithms	Sum of Ranks	p-WSAR vs.	p-value
GD	4.7000 (5)	GD	0.0020
GS	6.9000 (7)	GS	0.0020
RS	2.9000 (3)	RS	0.0020
SA	5.9000 (6)	SA	0.0020
TA	4.5000 (4)	TA	0.0020
WSAR	2.1000 (2)	WSAR	0.0020
p-WSAR	1.0000 (1)		

Table 4.16 Friedman and Wilcoxon signed-rank test results on 15D case (SSBMAs)

Average rankings achieved by Friedman test		Wilcoxon signed-rank test between p-WSAR and other algorithms	
Algorithms	Sum of Ranks	p-WSAR vs.	p-value
GD	4.7000 (5)	GD	0.0020
GS	7.0000 (7)	GS	0.0020
RS	3.0000 (3)	RS	0.0020
SA	5.8000 (6)	SA	0.0020
TA	4.5000 (4)	TA	0.0020
WSAR	2.0000 (2)	WSAR	0.0020
p-WSAR	1.0000 (1)		

Table 4.17 Friedman and Wilcoxon signed-rank test results on 20D case (SSBMAs)

Average rankings achieved by Friedman test		Wilcoxon signed-rank test between p-WSAR and other algorithms	
Algorithms	Sum of Ranks	p-WSAR vs.	p-value
GD	4.4000 (5)	GD	0.0020
GS	7.0000 (7)	GS	0.0020
RS	3.2000 (3)	RS	0.0020
SA	5.8000 (6)	SA	0.0020
TA	4.6000 (4)	TA	0.0020
WSAR	2.0000 (2)	WSAR	0.0020
p-WSAR	1.0000 (1)		

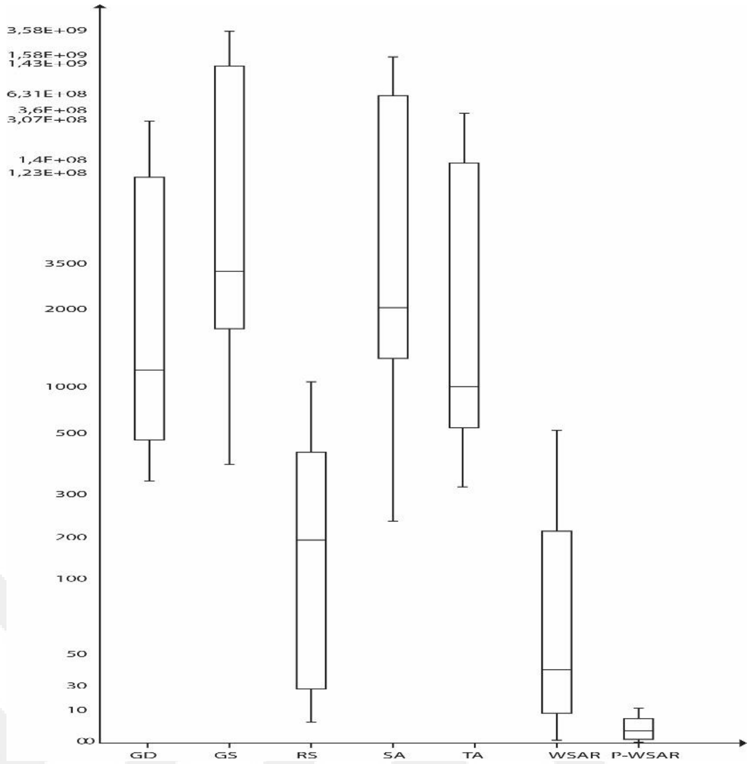


Figure 4.5 Boxplot of SSBMAs on CEC 5D average results

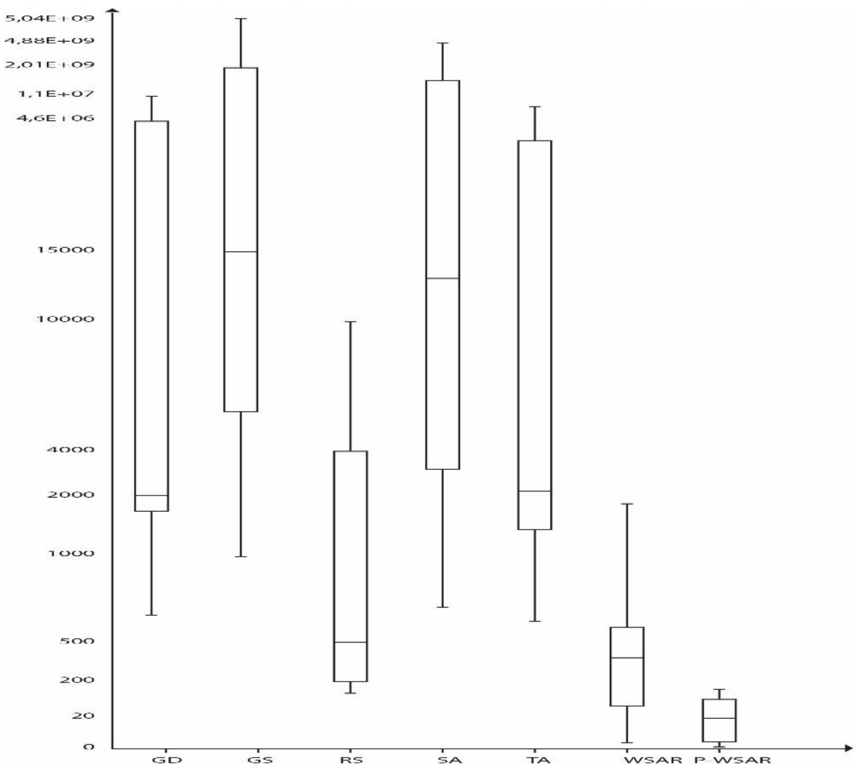


Figure 4.6 Boxplot of SSBMAs on CEC 10D average results

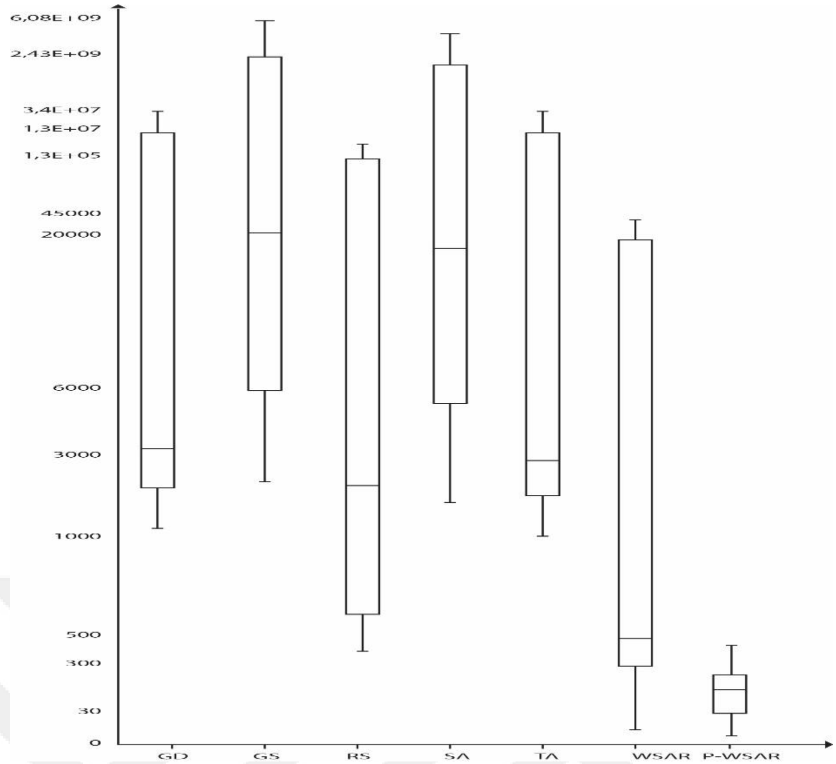


Figure 4.7 Boxplot of SSBMAs on CEC 15D average results

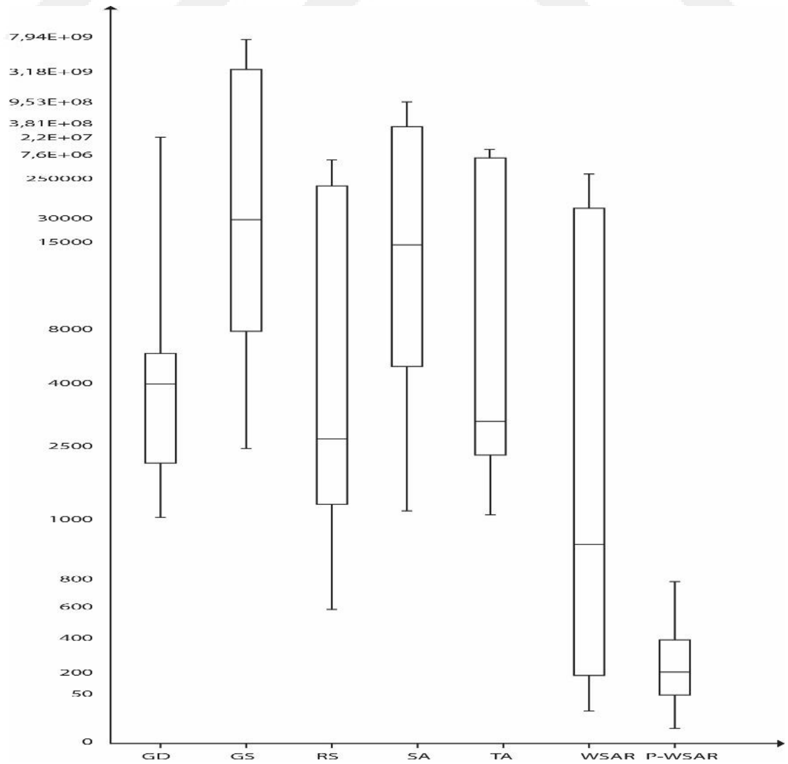


Figure 4.8 Boxplot of SSBMAs on CEC 20D average results

Performance of the p-WSAR algorithm is also compared with its serial version (where SSBMAs run in serial) in terms of solution time. Average computational time of each iteration is recorded for both versions (parallel and serial). The average computational times are presented in Table 4.18. It is clear that parallelization provides considerable advantage in computational times.

Table 4.18 Average computational times

Problem Dimension	Average Computational Time (in seconds)	
	p-WSAR(parallel)	p-WSAR(serial)
5D	0.8s	4.2s
10D	1.2s	6.5s
15D	1.9s	9.1s
20D	2.4s	11.8s

All solutions obtained from the proposed algorithm is shared in the following link: http://web.deu.edu.tr/baykasoglu/pwsarcec20results.rar

In this chapter, several metaheuristic algorithms are brought together to form a coalition under WSAR (Weighted Superposition Attraction-Repulsion Algorithm) in a parallel computing environment for solving complex unconstrained continuous optimization problems. The proposed methodology runs different single solution based metaheuristic algorithms (SSBMAs) in parallel and employs WSAR as the controller. The proposed approach is tested against the latest well-known and highly complex unconstrained continuous optimization problems (CEC2020). Results show that the proposed approach is capable of finding as effective solutions as the state of the art metaheuristic algorithms. Besides, the performance of the proposed algorithm is also compared with its constituents (SSBMAs and WSAR). The computational results prove the rationale of the proposed approach. SSBMAs and WSAR cannot produce effective solutions when they are run individually. However, when they come together (by forming a coalition), they can produce very effective results. On the other hand, thanks to parallel computing, algorithms could work simultaneously and runtime is considerable shortened.

CHAPTER 5

p-WSAR ALGORITHM ON BINARY OPTIMIZATION PROBLEMS

In the previous chapter, p-WSAR is proven to be successful in real-valued optimization problems. In this chapter, the performance of this algorithm will be analyzed in binary optimization problems, which is the main motivation of the present chapter. Accordingly, p-WSAR is redesigned for binary spaces. In this extension, p-WSAR does not require any functions for converting real values to binary, whereas they are commonly used in numerous approximation algorithms. Binary modification performance of this new generation metaheuristic is analyzed by using three well-known binary optimization problems, including uncapacitated facility location problem, 0-1 knapsack problem and a natural extension of it, the set union knapsack problem.

5.1 Used benchmarking problems

5.1.1 Uncapacitated facility location problem

Location problems have great importance for operations research society. They attract attention because they have numerous applications in real world (Laporte et al., 2015). UFLP (Cornuejols et al., 1990) is a well-known type of location problems, where each customer is served by exactly one facility and facilities are assumed to have unlimited capacities. Given a possible set of sites for establishing facilities and a set of customers at demand points, the aim here is to find the optimal locations for facilities to meet all demand such that the sum of facility opening costs and occurred shipment costs is minimized. UFLP can formally be formulated as in the following (Eqs. 5.1-5.5).

$$minimize \ \sum_{k\in K}\sum_{l\in L} c_{kl} z_{kl} + \sum_{k\in K} b_k y_k \tag{5.1}$$

s.t.

$$\sum_{k\in K} z_{kl} = 1, \qquad l \in L \tag{5.2}$$

$$z_{kl} \leq y_k, \qquad k \in K \text{ and } l \in L \tag{5.3}$$

$$z_{kl} \in \{0,1\}, \quad k \in K \text{ and } l \in L \tag{5.4}$$

$$y_k \in \{0,1\}, \qquad k \in K \tag{5.5}$$

where $K = \{1,2, \ldots, n\}$ is the possible locations of opening facilities set, $L = \{1,2, \ldots, m\}$ is the demand points set (customers), b_k is establishing a facility cost at the *k*th location, c_{kl} is the cost of shipment between the facility opened at the *k*th location and the *l*th customer, z_{kl} denotes whether the demand of the *l*th customer is supplied by the *k*th facility location and y_k is a binary variable that represents a facility location's status. Eq. 5.1 denotes the objective function that aims to minimize the total cost. Eq. 5.2 satisfies that each customer can exactly be supplied by only one facility. Eq. 5.3 provides that customers can only be supplied by the opened facilities. Finally, Eqs. 5.4-5.5 impose restrictions on decision variables. An illustration for this problem is presented in Figure 5.1.

UFLP is an NP-hard problem. Therefore, although there are reported exact solution techniques such as dual-based procedure (Erlenkotter, 1978), a Lagrange relaxation technique (Barcelo et. al., 1990) and a branch-and-bound method based on a dual ascent and adjustment procedure (Holmberg, 1990), by the emergence of metaheuristics algorithms in recent decades, this type of algorithms is more frequently employed in location problems. The reason is simple. UFLP shows an exponential growth as the dimensions of the problem are increased. Therefore, although exact solution methods guarantee optimality, they are unfortunately usable only to some extent. On the other hand, metaheuristic algorithms do not guarantee optimality, however, they are easy to implement to any size of a problem.

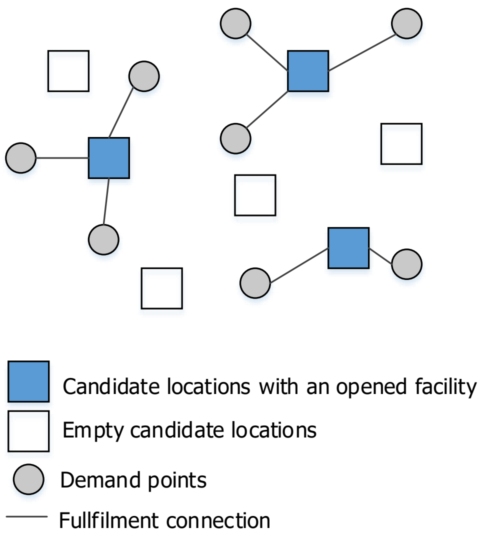


Figure 5.1. An illustration for UFLP

There are lots of publications that focus on location problems. Reporting them all is out of the scope of the present contribution. Therefore, only the most promising ones are given place here. In one of them, Jaramillo et al. (2002) employ GAs on several location problems. Tabu Search is another well-known method to solve UFLP (Al-Sultan and Al-Fawzan, 1999; Sun, 2006). Wang et al. (2008) presented a PSO based approach for UFLP. Literature also includes other PSO implementations for this problem (Sevkli and Guner, 2006; Guner and Sevkli, 2008). In some latest studies, an Kiran (2015) proposed an Artificial Bee Colony (ABC) algorithm and Tsuya et al. (2017) employ FA to solve UFLP. de Armas (2017) take both deterministic and stochastic versions of UFLP into consideration.

5.1.2 0-1 Knapsack problem

0-1 KP (Kellerer et al., 2004) is the second benchmark problem that is used in this study. Because it has many real life applications such as project funding selection, cutting stock, budget management, cargo loading etc., this problem also attracts attention of the researchers. The aim here is to maximize the knapsack profit such that the knapsack capacity is not exceeded. A mathematical model for this problem can be formulated as in Eqs. 5.6-5.8.

$$maximize \sum_{r \in R} p_r x_r \tag{5.6}$$

s.t.

$$\sum_{r \in R} w_r x_r \leq C \tag{5.7}$$

$$x_r \in \{0,1\}, \qquad r \in R \tag{5.8}$$

where $R = \{1,2, \dots, z\}$ is the item set, p_r denotes the profit of item r, w_r is the rth item's resource consumption in the knapsack, C is the knapsack capacity, x_r is a binary variable that denotes the rth item is in the knapsack or not ($x_r = 1$ if the rth item is assigned, $x_r = 0$, otherwise). In this regard, Eq. 5.6 is the objective function that aims to maximize the knapsack's profit. Eq. 5.7 is the capacity of knapsack that should not be exceeded. Finally, Eq. 5.8 defines the domain set of decision variables.

0-1 KP is an NP-hard problem (Kellerer et al., 2004). Although there exist some recently developed exact solution methodologies (Della Croce et al., 2017a; Della Croce et al., 2017b), heuristic approaches are more commonly preferred here due to the similar reasons to the UFLP's. Shi (2006) proposes an improved modification of ACO. A schema-guiding evolutionary algorithm is used by Liu and Liu (2009). In the same year, a binary modification PSO is employed by Li and Li (2009). There are also several new bio-inspired algorithms lately proposed to solve 0-1 KP (Bhattacharjee and Sarmah, 2014; Feng et al., 2017). Zhou et al. (2016) use a complex-valued encoding within a BA. Finally, the same procedure is adopted by Zhou et al. (2017) within another approximation algorithm, namely, wind driven optimization algorithm. Bhattacharjee and Sarmah (2017) report a survey focusing on swarm-based algorithm for knapsack problems.

5.1.3 Set union knapsack problem

SUKP (Goldschmidt et al., 1994; Kellerer et al., 2004; Arulselvan, 2014) is an extension of 0-1 KP. It includes a set of elements $U = \{1, \dots, n\}$ and a set of items $\aleph = \{1, \dots, m\}$. $\aleph$ $(i = 1, \dots, m)$ denotes a subset of elements, represented by S_i, with a non-negative profit denoted by $p: \aleph \rightarrow \mathbb{R}^+$. Each of the elements has non-negative weight $w: U \rightarrow \mathbb{R}^+$ in SUKP. For an arbitrary subset $A \subseteq \aleph$, total weight of the union of subset A is defined as $W(A) = \sum_{e \in \cup_{i \in A} S_i} w_e$ and profit of A is represented by $P(A) = \sum_{i \in A} p_i$. The objective is to find a combination of the subset of items $\aleph^* \subseteq \aleph$ that maximizes the total profit without exceeding the knapsack. The mentioned problem is more formally given by Eqs. 5.9-5.10, where w_e is the eth element's weight in the union set of the selected items, p_i is the profit of ith item in the subset A and C is the knapsack capacity. For a better understanding, a motivating example picture of SUKP is also depicted in Figure 5.2, where 3 items and 3 elements are used for illustration.

$$maximize\ P(A) = \sum_{i \in A} p_i \tag{5.9}$$

s.t.

$$W(A) = \sum_{e \in \cup_{i \in A} S_i} w_e \leq C, \quad A \subseteq \aleph \tag{5.10}$$

SKUP has many practical applications including financial decision making, flexible manufacturing, database partitioning and data stream compression (He et al., 2018). Although there is a number of SKUP-related publication in the literature, this problem deserves further research. In one of them, Goldschmidt et al. (1994) first propose a dynamic programming procedure for SUKP. Lister et al. (2010) address dynamic character caching as SKUP. An important note for this problem is presented by Arulselvan (2014). In addition, Riondato and Vandin (2014) utilized SUKP to get an upper bound for their problem. Taylor (2016) proposed various approximation schemes for SUKP. Su and Zhou (2016) modified SUKP to define it in a reduced form. Diallo et al. (2017) utilized SUKP to solve model virtual machine selection problem in cloud computing. Finally, He et al. (2018) present a recent and detailed information about this problem.

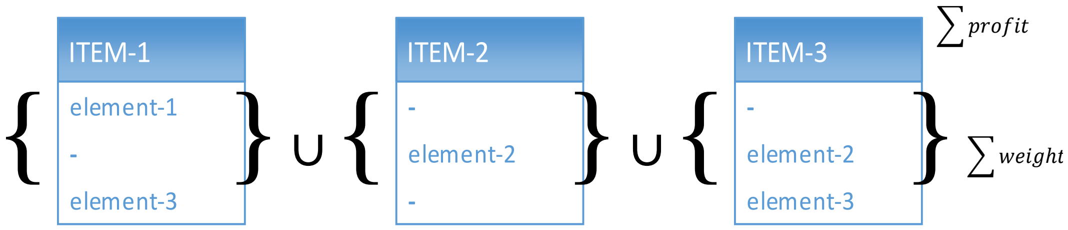


Figure 5.2 An illustration for set union knapsack problem

5.2 Modifications of p-WSAR for binary problems

As mentioned in Chapter 3, an attractive superposition and a repulsive superposition are generated in p-WSAR However, in a binary space, a repulsive superposition is not practicable. Therefore, only attractive superposition, clarified in the following subsection, is utilized.

5.2.1 Generating a binary superposition

First, solutions returned from SSBMAs (agents) are sorted in regard to their fitness values (1st rank is the fittest) and *weights* are evaluated. Next, uniform random numbers *rand*() ∈ [0,1] are generated for each dimension. These random numbers are utilized as threshold values to choose the candidate agents who will be selected using a roulette wheel. Agents with weights equal to or greater than these thresholds are thus subjected to a roulette wheel method. Finally, the bit of the winner agent is chosen as the corresponding bit of the superposition vector. This procedure is followed for each of the dimensions. For a better understanding, an example with different *weights* for different τ values is presented in Table 5.1.

Table 5.1 A sample population for p-WSAR in binary spaces

						weight (1,*i*)=$i^{-\tau}$				
ranks	*dimension*					$\tau = 0.1$	$\tau = 0.3$	$\tau = 0.5$	$\tau = 0.7$	$\tau = 0.9$
1	0	1	**1**	0	**1**	1.000	1.000	1.000	1.000	1.000
2	**1**	1	0	**0**	1	0.933	0.812	0.707	0.615	0.535
3	0	**1**	0	1	1	0.895	0.719	0.577	0.463	0.372
4	1	1	1	0	0	0.870	0.659	0.500	0.378	0.287
5	1	0	0	1	1	0.851	0.617	0.447	0.324	0.234
6	0	0	1	1	0	0.835	0.584	0.408	0.285	0.199
7	1	0	1	0	1	0.823	0.557	0.377	0.256	0.173
8	1	1	0	1	0	0.812	0.535	0.353	0.233	0.153
random numbers	0.6	0.3	0.35	0.4	0.9					
superposition	1	1	1	0	1					

Let's assume that τ is fixed to 0.7 and generated random numbers for each of the corresponding dimensions are 0.6, 0.3, 0.35, 0.4 and 0.9, respectively. Thus, for the first dimension, the first two agents undergo a roulette wheel, because, their weights are greater than 0.6. The bit of the winner agent (underlined and highlighted in bold) of this roulette wheel is transferred to the superposition vector (which is currently empty). The same procedure is performed for the remaining dimensions. In this respect as highlighted in Table 5.1, the first five, the first four and the first three agents go through a roulette wheel, respectively for the rest of the dimensions except for the last dimension. The roulette wheel approach requires at least two candidates that's why the last bit of the superposition vector is directly taken from the first agent since it is the only candidate.

It should be noted that the first agent is always a candidate for the roulette wheel procedure. It is also clear that transfer methods like s-shaped or v-shaped functions are not necessarily required here. Thus, one can simply use a binary population. As another issue, in parallel with p-WSAR in continuous spaces, behavior of p-WSAR in binary spaces can also be controlled via different values of τ. It should be stressed that, while lower values of τ encourages diversity, on the contrary, greater values of τ yield to a more elitist behavior.

5.2.2 Solution representation and generating neighborhood

A solution is represented by binary variables in pWSAR for binary spaces. It is clear for the UFLP that the 1s in a solution string represent the opened facilities at the corresponding locations. Therefore, the length of a solution string for UFLP is equal to the number of possible locations at which a facility can be established. Similarly, length of a solution string for 0-1 KP and SUKP is equal to the number of items that can be assigned to the knapsack. In parallel, 1s in a solution string for 0-1 KP represent the included items in the knapsack.

In pWSAR for continuous spaces, agents move according to Eq 3.1 and Eq. 3.2. The parameters ss and φ control the speed of the movements. However, since these parameters and equations are not suitable for binary spaces, they are modified through uniform-based crossover and bit flipping procedures.

It should be recalled that agents either move towards the coordinates of the superposition or they perform random walks. Let us first consider the random walks here. In order to detect promising regions quickly, it is clear that a more diversified search is required at the beginning of the search. Additionally, it is known that an approximation algorithm should be encouraged to perform more augmented search through the end of the run to achieve more accurate and better results. Therefore, a step length that exponentially decreases (Baykasoglu, 2012; Baykasoğlu and Ozsoydan, 2015) given in Eq. 3.3 is extended here for binary spaces. In this extension, the parameter $sl^t \in [0,1]$ is decreased for the next iteration by using the formula given in Eq. 5.11. The calculated value is then multiplied by the length of the solution string (dimension), and the result is adjusted up to find the total number of bits to flip., which shown by int_sl^{t+1} (Eq. 5.12). Finally, randomly selected int_sl^{t+1} bits of an agent are flipped.

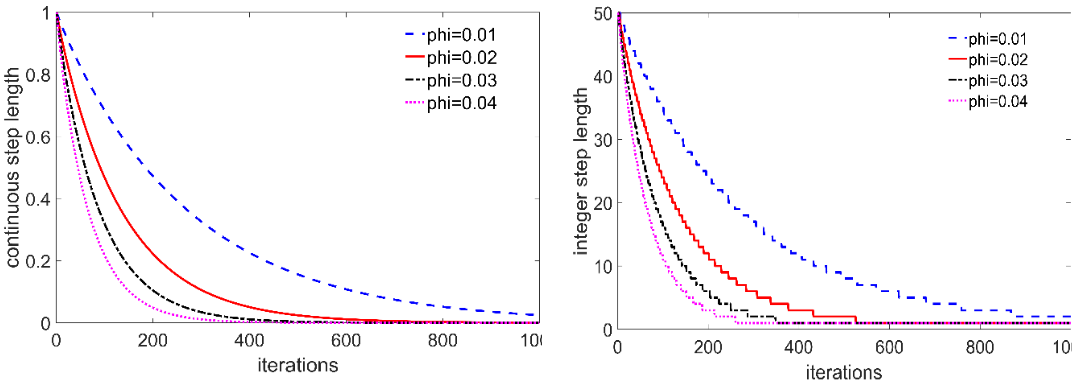
$$sl^{t+1} = sl^t - e^{-t/(t+1)} * \varphi * sl^t \tag{5.11}$$

$$int\_sl^{t+1} = \lceil sl^{t+1} \times D \rceil \tag{5.12}$$

Differently from p-WSAR in continuous spaces, it assumed here that initial step length is bounded by the interval $sl^1 \in [0,1]$. For example, if sl^1 is set to 0.50, an agent performing a random walk flips 50% of its bits at the first iteration. An illustration for sl^1=1.00 and for *D*=50 using different values of φ is presented in Figure 5.3. As one can see from this figure, greater values of φ give rise to steeper decrements in the continuous step length (Figure 5.3.a). Accordingly, randomly walking agents flip only a few bits throughout the second half of the search. On the other hand, φ=0.01 yields to a late intensification.

While moving an agent towards the superposition, a uniform-based crossover is employed in p-WSAR in binary spaces. In this procedure, first, uniform-based crossover rate (p_{uxo}) is defined. This parameter represents the probability of taking a gene from the superposition vector. Accordingly, random numbers *rand*()$\in [0,1]$ are drawn for each dimension separately. If the generated random number for the corresponding bit is smaller than or equal to p_{uxo}, that dimension takes the value of the superposition's. Otherwise, it remains stable. Thus, greater values of p_{uxo} induce faster movements towards the superposition.

In the present study, p_{uxo} is assumed to remain stationary until the end of the run. A pseudo code of movement of the agents in p-WSAR for binary spaces, which is prepared for a minimization problem, is presented in Figure 5.4. Finally, putting things together, a pseudo code for p-WSAR for binary spaces is given in Figure 5.5.



(a) (b)

Figure 5.3 Exponentially decreasing a) continuous step length and b) corresponding integer step length for different values of φ

if $f(\overrightarrow{AS}) \leq f(i)$

  for (j=1 *to* D) ***do***

    if rand() $\leq p_{uxo}$

      $\vec{x}(i, j)= AS(1,j)$

    end if

  end for

else if $f(i) < f(\overrightarrow{AS})$

  if rand() $< e^{f(i)-f(\overrightarrow{AS})}$

    for (j=1 *to* D) ***do***

      if rand() $\leq p_{uxo}$

        $\vec{x}(i,j)= \overrightarrow{AS}(1,j)$

      end if

    end for

  else

    for (j=1 *to* int_sl^t) ***do***

      select a random locus ω

      flip the value of $\vec{x}(i,\omega)$

    end for

  end if

end if

end for

Figure 5.4 Movement of the population

```
set the input parameters of SSBMAs
generate an initial population & evaluate fitness values
while (itr<maxiter) do
        run SSBMAs in parallel
        rank the solutions (agents) according to their fitness values
        determine the weights and the attractive superposition (Table 5.1)
        evaluate fitness of the binary superposition
        move SSBMAs (Figure 5.4)
        evaluate fitness of the SSBMAs
        update the best found solution
        modify the ith step length  (Eqs. 5.11-5.12)
        Randomly generate a new set of parameters for SSBMAs
        Randomly match solutions with SSBMAs

end
print the best solution
```

Figure 5.5 Pseudo code of p-WSAR for binary spaces

5.2.3 Constraint handling mechanism

The first binary problem solved via p-WSAR in the present study is indeed obtained by excluding the capacity constraints from facility location problems. In other words, demands of the all customers can be fulfilled by only a single opened facility in UFLP. It is also mentioned in the previous subsection that the number of the opened facilities is controlled by solution representations in p-WSAR. Therefore, this problem can be considered as an unconstrained binary optimization problem. Additionally, as mentioned by Kiran (2015), demand of a customer is always entirely fulfilled by the nearest opened facility. On the other hand, 0-1 KP as the second problem, has a knapsack capacity constraint. Thus, the second problem solved here is a constrained binary optimization problem. In this regard, the penalizing function of Zou et al. (2011) (Eq. 5.13) is adopted here for 0-1 KP.

$$minimize(-\sum_{r \in R} p_r x_r + \omega \times \max(0, g)) \quad (5.13)$$

In this method, first, maximization function of 0-1 KP (Eq. 5.6) is transformed into a minimization function by multiplying it by -1. Next, the penalty term $\omega \times \max(0, g)$ is appended to the objective function, where $g = \sum_{r \in R} w_r x_r - C$ and ω is a scalar that is fixed to 10^{20}.

For SUKP, the repairing method presented in He et al. (2018) is employed. In this method, first, a quick sorting algorithm is used to sort the items in regard to their profit/weight ratios, taking the frequencies of elements separately for each item into account. Next, violating item(s) is(are) found by using this sorted list. After discarding them, a posterior stage in run to detect any fitting item(s) to the remaining capacity. This procedure is given in detail by He et al. (2018).

5.3 Computational Results

5.3.1 Results for UFLP

In order to test the performance of the pWSAR in UFLP, benchmarking instances, whose optimum solution values are already known, are adopted from OR-Lib (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/). The attributes of the mentioned problems are presented in Table 5.2.

Computational results for UFLP are given in Table 5.3. Columns of this table (*best*, *worst*, *std. dev.*, *hit*) represent the best found solution, the worst found solution, standard deviance of the found solutions over 30 runs and the number of the runs for which the optimum is found, respectively. These results of p-WSAR are compared to the previously published results of the algorithms based on bWSA, PSO and ABC.

As one can see from Table 5.3, p-WSAR is capable of finding optimum in all instances. Additionally, the highlighted results in Table 5.3 represent the instances, where *hit* numbers are greater than or equal to the *hit* numbers of other algorithms. In the respect, PWSAR clearly outperforms the other algorithms.

Table 5.2 Attributes of the used UFLP instances

problem ID	*size (facilities × customers)*	*optimum solution value*
#cap71	16×50	932615.750
#cap72	16×50	977799.400
#cap73	16×50	1010641.450
#cap74	16×50	1034976.975
#cap101	25×50	796648.437
#cap102	25×50	854704.200
#cap103	25×50	893782.112
#cap104	25×50	928941.750
#cap131	50×50	793439.562
#cap132	50×50	851495.325
#cap133	50×50	893076.712
#cap134	50×50	928941.750

Table 5.3 Computational results for UFLP

problem ID	CPSO (Sevkli and Guner, 2006)				$\mathrm{ABC_{bin}}$ (Kiran, 2015)				bWSA				p-WSAR			
	best	*worst*	*std. dev*	*hit*	*best*	*worst*	*std. dev*	*hit*	*best*	*worst*	*std. dev*	*hit*	*best*	*worst*	*std. dev*	*hit*
cap71	932615.75	934199.14	562.23	25	932615.75	932615.75	0.00	30	932615.750	932615.750	0.000	30	**932615.750**	***932615.750***	**0.000**	***30***
cap72	977799.40	983713.81	1324.30	25	977799.40	977799.40	0.00	30	977799.400	977799.400	0.000	30	**977799.400**	***977799.400***	***0.000***	***30***
cap73	1010641.45	1012643.69	702.13	22	1010641.45	1010641.45	0.00	30	1010641.450	1010641.450	0.000	30	**1010641.450**	***1010641.450***	***0.000***	***30***
cap74	1034976.98	1045342.23	2124.54	0	1034976.98	1034976.98	0.00	30	1034976.975	1034976.975	0.000	30	**1034976.975**	***1034976.975***	***0.000***	***30***
cap101	796648.44	802457.23	1480.72	0	796648.44	796648.44	0.00	30	796648.437	797508.725	380.434	22	**796648.437**	**796648.437**	***0.000***	***30***
cap102	854704.20	857380.85	1015.64	10	854704.20	854704.20	0.00	30	854704.200	854704.200	0.000	30	**854704.200**	***854704.200***	***0.000***	***30***
cap103	893782.11	899424.91	1695.79	0	893782.11	894008.14	85.67	25	893782.112	895027.188	470.951	10	**893782.112**	**893782.112**	***0.000***	***30***
cap104	928941.75	944394.83	3842.64	18	928941.75	928941.75	0.00	30	928941.750	928941.750	0.000	30	**928941.750**	***928941.750***	***0.000***	***30***
cap131	795291.86	804549.64	2429.54	0	793439.56	794910.64	1065.73	6	793439.562	798449.038	1025.786	6	**793439.562**	***793439.562***	***0.000***	***30***
cap132	851495.33	865667.16	4297.07	0	851495.33	851636.70	213.28	14	851495.325	852257.975	251.654	23	**851495.325**	***851495.325***	***0.000***	***30***
cap133	893076.71	909908.70	4210.93	0	893076.71	895407.93	561.34	5	893076.712	894801.163	501.912	7	**893076.712**	***893076.712***	***0.000***	***30***
cap134	928941.75	951803.25	6619.05	7	928941.75	928941.75	0.00	30	928941.750	934586.975	1016.144	26	**928941.750**	***928941.750***	***0.000***	***30***

5.3.2 Results for 0-1 KP

In order to analyze the performance of pWSAR in 0-1 KP, commonly used standard benchmarks (Table 5.5) are used. All printed results are evaluated over 30 independent runs.

The results of pWSAR for the 10 standard commonly used benchmarks are presented in Table 5.4, whose columns (f, *opt*., *best*, *mean*, *worst*,*std*. *dev*., gap*%*, *hit*) denote the instance index, optimum value, the best found solution, mean over 30 runs, the worst found solution, standard deviance, percentage gap (Eq. 5.14) and the number of runs, for which the optimum is found, respectively.

$$gap\% = \left({}^{(z^* - opt)}\!/_{opt} \right) \times 100 \tag{5.14}$$

Table 5.4 Standard instances for 0-1 KP (Zou et. al., 2011; Bhattacharjee and Sarmah, 2014).

f	*opt.*	*best*	*mean*	*worst*	*std. dev.*	*gap%*	*hit*
f_1	295	295	295	295	0.000	0.000	30
f_2	1024	1024	1024	1024	0.000	0.000	30
f_3	35	35	35	35	0.000	0.000	30
f_4	23	23	23	23	0.000	0.000	30
f_5	481.0694	481.0694	481.0694	481.0694	0.000	0.000	30
f_6	52	52	52	52	0.000	0.000	30
f_7	107	107	106.53	105	0.000	0.000	30
f_8	9767	9767	9767	9767	0.000	0.000	30
f_9	130	130	130	130	0.000	0.000	30
f_{10}	1025	1025	1025	1025	0.000	0.000	30

As one can see from Table 5.4, p-WSAR is capable of finding optimum solution for each of the instances f_{1-10}. Moreover, p-WSAR finds optimum solutions in all runs.

Additionally, performance of the p-WSAR is also compared with the performance of SSBMAs and bWSA based on Friedman and Wilcoxon signed-rank test results. According to the test results, (from Friedman test) p-WSAR performs the best among other algorithms. In addition, based on the Wilcoxon signed-rank test, the difference between p-WSAR and bWSA is found negligible as $p > 0.05$. Besides p-WSAR is performed slightly better than GD, GS, RS, SA, TA as the $p < 0.05$.

Table 5.5 Friedman and Wilcoxon signed-rank test results on KP

Average rankings achieved by Friedman test		Wilcoxon signed-rank test between p-WSAR and other algorithms	
Algorithms	Sum of Ranks	p-WSAR vs.	p-value
GD	4.900	GD	0.0020
GS	5.700	GS	0.0020
RS	5.900	RS	0.0020
SA	3.800	SA	0.0039
TA	4.600	TA	0.0020
bWSA	1.650	bWSA	0.5000
p-WSAR	1.450		

5.3.3 Results for SUKP

In order the test the performance of p-WSAR for SUKP, the instances introduced by He et al. (2018) are used in the present work. The authors name the instances in the form of $m\_n\_\alpha\_\beta$, where m is the number of items, n is the number of elements, α is the density of an element and β is a parameter to define the knapsack capacity.

Three groups of instances are provided He et al. (2018), where it is assumed that $m > n, m = n, m < n$ for the first, for the second and for the third group of instances, respectively. Differently from the previous problems, *NS* is fixed to 100 for SUKP. The rest of the parameters are used exactly the same. All results, which are presented in Tables 5.6-5.8, are evaluated over 100 independent runs.

In these tables, the *Instance* columns represent the attributes of the instance. The columns *A-SUKP* give the results obtained by the method proposed by Arulselvan (2014). Another column *BABC* is devoted to the results of the binary *ABC* of He et al. (2018), inspired from the study of Ozturk et al. (2015). A similar algorithm is denoted by ABC_{bin} that is adopted from Kiran (2015). The other columns *GA* and *binDE* are devoted to the results of GA and a binary modification of DE. All these results are adopted from He et al. (2018). The bWSA results are adopted from Baykasoğlu et al. (2020). Finally, the columns *p-WSAR* are devoted to the results of the proposed approach here.

As one can see from these tables p-WSAR outperforms the rest of the algorithms in a great majority of the instances. It is apparent that there are some ties particularly in smaller-scaled problems. However, it is further clear that the performance of p-WSAR is better than the compared algorithms particularly in large-scaled problems. This indeed shows that the proposed approach mostly obtains higher quality of solutions in SUKP regardless of the problem size.

Table 5.6 The first type instances results for SUKP

Instance	*Results*	*A-SUKP*	*GA*	*BABC*	*ABC_{bin}*	*binDE*	*bWSA*	*p-WSAR*
100_85_0.10_0.75	*best*	12459	**13044**	13251	13044	13044	**13044**	**13044**
	mean	12459	12956.4	13028.5	12818.5	12991	12915.67	12917.23
	std. dev.	0.00	130.66	92.63	153.06	75.95	185.45	164.65
	worst	-	-	-	-	-	12244	12332
100_85_0.15_0.85	*best*	11119	12066	12238	12238	**12274**	12238	12274
	mean	11119	11546	12155	12049.3	12123.9	11527.41	12456.30
	std. dev.	0.00	214.94	53.29	96.11	67.61	332.27	335.75
	worst	-	-	-	-	-	10408	10605
200_185_0.10_0.75	*best*	11292	13064	13241	12946	13241	**13250**	**13250**
	mean	11292	12492.5	13064.4	11861.5	12940.7	12657.65	132.20
	std. dev.	0.00	320.03	99.57	324.65	205.70	319.58	325.62
	worst	-	-	-	-	-	11951	12032
200_185_0.15_0.85	*best*	12262	13671	13829	13671	13671	**13858**	**13858**
	mean	12262	12802.9	13359.2	12537	13110	12585.35	13565.84
	std. dev.	0.00	291.66	234.99	289.53	269.69	302.66	304.75
	worst	-	-	-	-	-	11836	11948
300_285_0.10_0.75	*best*	8941	10553	10428	9751	10420	**10991**	**10991**
	mean	8941	9980.87	9994.76	9339.3	9899.24	10366.21	10412.58
	std. dev.	0.00	142.97	154.03	158.15	153.18	257.10	276.32
	worst	-	-	-	-	-	9802	9912

Table 5.6 Continues

Instance	*Results*	*A-SUKP*	*GA*	*BABC*	*ABC_{bin}*	*binDE*	*bWSA*	*p-WSAR*
300_285_0.15_0.85	*best*	9432	11016	12012	10913	11661	**12093**	**12093**
	mean	9432	10349.8	10902.9	9957.85	10499.4	10901.59	11235.25
	std. dev.	0.00	215.13	449.45	276.90	403.95	508.79	565.41
	worst	-	-	-	-	-	9912	9958
400_385_0.10_0.75	*best*	9076	10083	10766	9674	10576	**11321**	**11321**
	mean	9076	9641.85	10065.2	9187.76	9681.46	10785.74	11235.15
	std. dev.	0.00	168.94	241.45	167.08	275.05	361.45	389.36
	worst	-	-	-	-	-	9798	9865
400_385_0.15_0.85	*best*	8514	9831	9649	8978	9649	**10435**	**10435**
	mean	8514	9326.77	9135.98	8539.95	9020.87	9587.72	9629.35
	std. dev.	0.00	192.20	151.90	161.83	150.99	360.29	372.90
	worst	-	-	-	-	-	8695	8713
500_485_0.10_0.75	*best*	9864	11031	10 784	10340	10586	**11540**	**11540**
	mean	9864	10567.9	10452.2	9910.32	10363.8	10921.58	11091.20
	std. dev.	0.00	123.15	114.35	120.82	93.39	351.69	352.85
	worst	-	-	-	-	-	10293	11282
500_485_0.15_0.85	*best*	8299	9472	9090	8759	9191	**9681**	**9681**
	mean	8299	8692.67	8857.89	8365.04	8783.99	9013.09	9102.17
	std. dev.	0.00	180.12	94.55	114.10	131.05	204.85	212.58
	worst	-	-	-	-	-	8479	8512

Table 5.7 The second type instances results for SUKP

Instance	*Results*	*A-SUKP*	*GA*	*BABC*	*ABC_{bin}*	*binDE*	*bWSA*	*p-WSAR*
100_100_0.10_0.75	*best*	13634	**14044**	13860	13860	13814	**14044**	**14044**
	mean	13634	13806	13734.9	13547.2	13675.9	13492.71	13503,08
	std. dev.	0.00	144.91	70.76	119.11	119.53	325.34	332.23
	worst	-	-	-	-	-	12625	12703
100_100_0.15_0.85	*best*	11325	13145	**13508**	13498	13407	13407	13612
	mean	11325	12234.8	13352.4	13103.1	13212.8	12487.88	12692.33
	std. dev.	0.00	388.66	155.14	343.46	287.45	718.23	770.68
	worst	-	-	-	-	-	10987	11032
200_200_0.10_0.75	*best*	10328	11656	11846	11191	11535	**12271**	**12271**
	mean	10328	10888.7	11194.3	10424.1	10969.4	11430.23	11507.54
	std. dev.	0.00	237.85	249.58	197.88	302.52	403.33	459.22
	worst	-	-	-	-	-	10622	10713
200_200_0.15_0.85	*best*	9784	11792	11521	11287	11469	**11804**	**11804**
	mean	9784	10827.5	10945	10345.9	10717.1	11062.06	11154.12
	std. dev.	0.00	334.43	255.14	273.47	341.08	423.90	440.09
	worst	-	-	-	-	-	10042	10165
300_300_0.10_0.75	*best*	10208	12055	12186	11494	12304	**12644**	**12644**
	mean	10208	11755.1	11945.8	10922.3	11864.4	12227.56	12356.12
	std. dev.	0.00	144.45	127.80	182.63	160.42	308.11	312.08
	worst	-	-	-	-	-	11365	11425

Table 5.7 Continues

Instance	*Results*	*A-SUKP*	*GA*	*BABC*	*ABC$_{bin}$*	*binDE*	*bWSA*	*p-WSAR*
300_300_0.15_0.85	*best*	9183	10666	10382	9633	10382	**11113**	**11113**
	mean	9183	10099.2	9859.69	9186.87	9710.37	10216.71	11002.65
	std. dev.	0.00	337.42	177.02	147.78	208.48	351.12	378.32
	worst	-	-	-	-	-	9520	9588
400_400_0.10_0.75	*best*	9751	10570	10626	10160	10462	**11199**	**11199**
	mean	9751	10112.4	10101.1	9549.04	9975.8	10624.79	10705.62
	std. dev.	0.00	157.89	196.99	141.27	185.57	266.46	281.54
	worst	-	-	-	-	-	9818	9906
400_400_0.15_0.85	*best*	8497	9235	9541	9033	9388	**10915**	**10915**
	mean	8497	8793.76	9032.95	8365.62	8768.42	9580.64	9635.26
	std. dev.	0.00	169.52	194.18	153.40	212.24	411.83	451.78
	worst	-	-	-	-	-	8717	8725
500_500_0.10_0.75	*best*	9615	10460	10755	10071	10546	**10827**	**10827**
	mean	9615	10185.4	10328.5	9738.17	10227.7	10482.80	10559.90
	std. dev.	0.00	114.19	91.615	111.63	103.32	165.62	176.96
	worst	-	-	-	-	-	10147	10193
500_500_0.15_0.85	*best*	7883	9496	9318	9262	9312	**10082**	**10082**
	mean	7883	8882.88	9180.74	8617.91	9096.13	9478.71	95123.38
	std. dev.	0.00	158.21	84.91	141.32	145.45	262.44	275.78
	worst	-	-	-	-	-	8705	8917

Table 5.8 The third type instances results for SUKP

Instance	*Results*	*A-SUKP*	*GA*	*BABC*	*ABC$_{bin}$*	*binDE*	*bWSA*	*p-WSAR*
85_100_0.10_0.75	*best*	10231	11454	11664	11206	11352	**11947**	**11947**
	mean	10231	11092.7	11182.7	10879.5	11075	11233.16	11665.29
	std. dev.	0.00	171.22	183.57	163.62	119.42	216.67	222.45
	worst	-	-	-	-	-	10627	11003
85_100_0.15_0.85	*best*	10483	12124	**12369**	12006	**12369**	**12369**	**12369**
	mean	10483	11326.3	12081.6	11485.3	11875.9	11342.70	11633.35
	std. dev.	0.00	417.00	193.79	248.33	336.94	474.76	485.19
	worst	-	-	-	-	-	9774	9846
185_200_0.10_0.75	*best*	11508	12841	13047	12308	13024	**13505**	**13505**
	mean	11508	12236.6	12522.8	11667.9	12277.5	12689.09	13001.05
	std. dev.	0.00	198.18	201.35	177.14	234.24	336.51	389.63
	worst	-	-	-	-	-	11820	11963
185_200_0.15_0.85	*best*	8621	**10920**	10602	10376	10547	10831	10920
	mean	8621	10351.5	10150.6	9684.33	10085.4	10228.07	10546.13
	std. dev.	0.00	208.08	152.91	184.84	160.60	286.92	292.66
	worst	-	-	-	-	-	9467	9554
285_300_0.10_0.75	*best*	9961	10994	11158	10269	11152	**11538**	**11538**
	mean	9961	10640.1	10775.9	9957.09	10661.3	11105.09	11212.28
	std. dev.	0.00	126.84	116.80	141.48	149.84	197.78	200.57
	worst	-	-	-	-	-	10600	10719

Table 5.8 Continues

Instance	*Results*	*A-SUKP*	*GA*	*BABC*	*ABC_{bin}*	*binDE*	*bWSA*	*p-WSAR*
285_300_0.15_0.85	*best*	9618	11093	10528	10051	10528	**11377**	**11377**
	mean	9618	10190.3	9897.92	9424.15	9832.32	10452.03	10525.32
	std. dev.	0.00	249.76	186.53	197.14	232.72	416.76	423.65
	worst	-	-	-	-	-	9519	9695
385_400_0.10_0.75	*best*	8672	9799	10085	9235	9883	**10414**	**10414**
	mean	8672	9432.82	9537.5	8904.94	9314.57	9778.03	9808.62
	std. dev.	0.00	163.84	184.62	111.85	191.59	221.49	259.56
	worst	-	-	-	-	-	9378	9662
385_400_0.15_0.85	*best*	8064	9173	9456	8932	9352	**10077**	**10077**
	mean	8064	8703.66	9090.03	8407.06	8846.99	9203.52	9342.21
	std. dev.	0.00	154.15	156.69	148.52	210.91	303.12	312.06
	worst	-	-	-	-	-	8600	8716
485_500_0.10_0.75	*best*	9559	10311	10823	10357	10728	**10835**	**10835**
	mean	9559	9993.16	10483.4	9615.37	10159.4	10607.21	10662.30
	std. dev.	0.00	117.73	228.34	151.41	198.49	191.86	198.47
	worst	-	-	-	-	-	10024	10102
485_500_0.15_0.85	*best*	8157	9329	9333	8799	9218	**9603**	**9603**
	mean	8157	8849.46	9085.57	8347.82	8919.64	9141.94	9203.63
	std. dev.	0.00	141.84	115.62	122.65	168.90	180.42	191.22
	worst	-	-	-	-	-	8562	8609

5.4 Concluding remarks

This chapter analyzes the performance of the p-WSAR in binary optimization problems. In this respect, p-WSAR is modified for binary spaces. A specialized roulette wheel procedure is used in this modification. Additionally, uniform-based crossover and random bit flips are employed while generating a new solution.

As secondarily, a systematically controlled step length mechanism to put control on the search speed is adopted in p-WSAR. Step length, which is indeed the number of bits to be flipped here, is exponentially decreased throughout generations. Thus, while p-WSAR performs a more diversified search at the initialization stage, it is encouraged to perform more intense search around up-and- coming regions through end of the run.

The performance of p-WSAR is investigated on some of the famous binary optimization problems, including uncapacitated facility location problem, 0-1 knapsack problem and set union knapsack problem, which have numerous application in real-life. Promising results and statistical tests point out the effectiveness of p-WSAR.

CHAPTER 6
p-WSAR ALGORITHM ON COMBINATORIAL OPTIMIZATION PROBLEMS

Performance of the p-WSAR was extensively investigaed with many continuous and binary optimization test problems in the previous chapters. The combinatoric version of p-WSAR developed in this chapter. The determination of the superposition mechanism is modified to handle combinatorial problems without needing a repair operator. A new crossover mechanism is incorporated into p-WSAR algorithm for improving its diversification capability. Performance of p-WSAR is tested on two well-known combinatorial optimization problems, namely the Resource Constrained Project Scheduling Problem (RCPSP) and the Permutation Flow Shop Scheduling Problem (PFSP). The results of the extensive computational study point out the efficiency of the p-WSAR.

6.1 Used Bench-mark problems

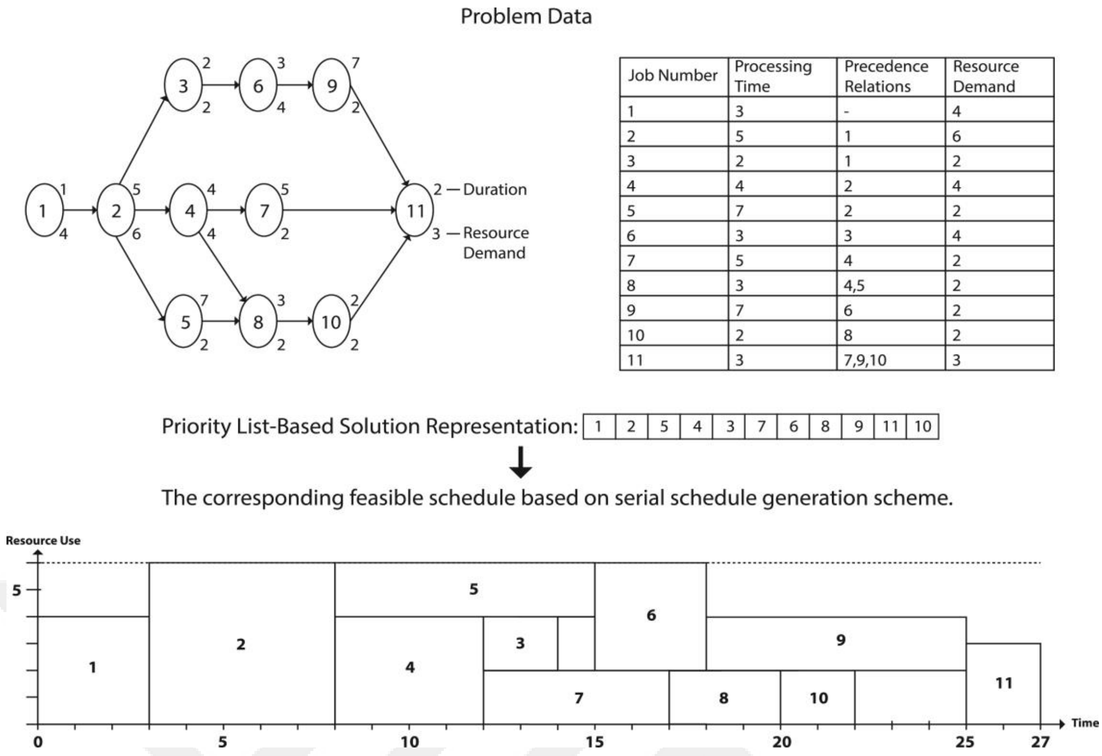
6.1.1 Resource Constrained Project Scheduling problem (RCPSP)

The RCPSP can be formally stated as follows: A single project contains n activities where each activity needs to be processed in order to complete the project. The activities are interconnected with two types of constraints. Precedence constraints mean that activity j cannot be started before all its immediate predecessors are completed. Carrying out activities requires resources with limited capacities. Overall there are a set of R resources. Activity j requires r_{jh} units of resource $h \epsilon R$ at every time instant of its non-empty duration p_j while it's processing. Resource h has a limited capacity of R_h at any point in time. The parameters p_j, r_{jh} and R_h are considered to be non-negative and deterministic. The objective of the RCPSP is to find precedence and resource feasible completion times for all activities such that the makespan of the project is minimized (Yang and Deb, 2010). RCPSPs are known as NP-hard therefore no efficient algorithm exists for solving them. Actually, the RCPSP is often considered as one of the most intractable problems in operations research, it is in fact a very popular benchmark for validating the newest optimization techniques, and virtually all discrete search algorithms have to be tested against this benchmark (Yang and Deb, 2010). RCPSP can be mathematically stated as follows (Miandoab, 2013):

$$\left\{\begin{array}{ll} \min t_j & \\ s.t. & \\ t_j - t_i \geq P_i & \forall_j \in S_i \\ \sum_{j\in A_f} G_{rj} \leq G_r^M & r \in R \\ t_j \geq 0 & j = 1, \dots, J \end{array}\right\} \quad (6.1)$$

Where, t_j is an integer finish time of activity j; A_f is the set of processing activities in period f. $A_f = \{j | t_j - p_j + 1 \leq f \leq t_j, \quad j = 1, \dots, J\}$; Si contains the set of activities that have been previously scheduled; $G^M{}_r$ is the maximum available amount of resource r in each period. The first constraint controls precedence and the second one controls resource limits.

In order solve the present RCPSP by making use of the p-WSAR we employed "priority list, PL" representation (Kim and Ralph D. Ellis, 2010) for solution encoding and the "serial scheduling generation scheme, SSGS" (Kim and Ralph D. Ellis, 2010) for solution decoding and fitness determination. PL puts activities in a priority order. During decoding, activities are selected from the PL starting from the first activity in the PL. The main purpose of the p-WSAR is actually to determine the best possible order of activities in the PL, which will hopefully yield the best possible schedule. Neighbourhood generation (move) mechanisms, which will be mentioned in the next section, are applied to the PL along with the random walk mechanism to generate neighbor solutions (new PLs). SSGS is activity oriented and it is able to generate active schedules. The activity increment approach of SSGS schedules each activity one at a time and as soon as possible within the precedence and resource constraints. To that purpose, SSGS scans the PL and selects at each stage the next activity from the PL in order to schedule it at its first possible starting time without violating precedence and resource constraints. An example RCPSP is presented in Figure 6.1 along with PL based solution encoding and the corresponding feasible schedule which is decoded by SSGS.



Job Number	Processing Time	Precedence Relations	Resource Demand
1	3	-	4
2	5	1	6
3	2	1	2
4	4	2	4
5	7	2	2
6	3	3	4
7	5	4	2
8	3	4,5	2
9	7	6	2
10	2	8	2
11	3	7,9,10	3

Figure 6.1 An example RCPSP with solution encoding and decoding

6.1.2 Permutation Flow Shop Scheduling Problem (PFSP)

There are a set of *m* machines and a group of *n* jobs in the PFSP. Every job composes a set of *m* operations, which need to be completed on different machines. Each of the *n* jobs has the same ordering of machines for its process sequence. Each machine can only perform one operation at a time and all machines execute jobs sequentially according to a predefined permutation schedule. It is assumed that machine failures do not occur, so all of the machines are ready for the processing of operations during the production stage. Pre-emption of operations is also not allowed. The objective is to find a schedule that minimizes the total completion time of the jobs without violating the assumptions above. (Anandaraman, 2011) stated the maximum completion time for the permutation schedule as follows (Anandaraman, 2011):

$$\left\{\begin{matrix} C(1,1) = T(1,1) \\ C(1,i) = C(1,i-1) + T(1,i) \\ C(r,1) = C(r-1,1) + T(r,1) \\ C(r,i) = \max\big(C(r,i-1), C(r-1,i)\big) + T(r,i) \end{matrix}\right\} \tag{6.2}$$

Where; $T(r,i)$ represents the execution time of the rth operation of the ith job on machine M_r. $C(r,i)$ refers to maximum completion time of the ith job on the machine M_r. When m is equal to number of machines and k is equal to total number of jobs. $C(m,k)$ refers to maximum completion time of jobs which is makespan.

In the PFSP, a permutation type n-dimensional real-number vector can be used to determine the process sequence of the jobs. After determining the order of the jobs, the makespan can be computed through the "completion time matrix approach", which was also utilized by Onwubolu and Davendra (Onwubolu and Davendra, 2006). Let us use a 5-machine and 5-job problem for demonstrating the working principle of the completion time matrix. The processing times of jobs are given in Table 6.1. Assume that a job permutation is obtained from the following solution vector: {3,5,2,1,4}.

Table 6.1 Processing times for a 5 x 5 example problem

Jobs				
1	2	3	4	5
5	7	4	3	6
6	5	7	6	7
7	8	3	8	5
8	6	5	5	8
4	4	8	7	3

The completion time matrix is depicted below. According to the completion matrix, the last entry of the completion matrix is the makespan (which is 58).

$$[C] = \begin{bmatrix} 4 & 10 & 17 & 22 & 25 \\ 11 & 18 & 23 & 29 & 35 \\ 14 & 23 & 31 & 38 & 46 \\ 19 & 31 & 37 & 46 & 51 \\ 27 & 34 & 41 & 50 & 58 \end{bmatrix}$$

6.2 Modifications of p-WSAR for combinatorial problems

p-WSAR necessarily differs from the continuous version of p-WSAR in some characteristics. The determination of the superpositions and step length need to be different from the continuous p-WSAR. Despite these differences, the general flow of the algorithm is exactly the same. p-WSAR contains three different/modified mechanisms which are named as "*determination of superpositions (attractive and repulsive)*" and "*determining SSBMAs' next moves*".

Determination of superposition: The main difference between continuous p-WSAR and p-WSAR in combinatorial problems is in the determination of the superposition. First of all, an empty vector is defined (*which is going to be the superposition at the end*). Then a random number is generated (*for each position of the SSBMA's solution vector "which contains a permutation of unique integer numbers" in turn*) and the weight of every agent is compared with this random number. If the SSBMAs weight is greater than the random number, then the corresponding element of that SSBMA is a candidate for the empty position. This comparison builds up the candidate list by comparing all of the SSBMAs' weights with the generated random number. Then the elements of the candidate list go through a roulette wheel approach and the winning element is determined. The winning candidate element is marked as selected (*and removed from the SSBMAs' solution vectors in the swarm temporarily*) to not to select the same element for the remaining empty positions in the superposition vector. This recursive procedure lasts until the all of the elements of the superposition vector are determined. In some of the iterations, the candidate list can be empty, in this case a element is selected among the unselected elements randomly. The algorithm for determination of the superposition is shown in Figure 6.2. An example of determination of a superposition is illustrated in Figure 6.3.

```
tar =zeros(1,D)
weight = zeros(1,NS)
isselected=zeros(1,DD)
FOR (i=1 to NS) DO
  weight(1,i) = i^((-1)^τ)
END FOR
FOR (i= 1 to D) DO
    Candidatelist=[];
     Counter=0;
      sprob=rand();
          FOR (j=1 to NS) DO
               IF weight(1,j) > sprob
                   IF isselected (1,x(j,i))==0
                        Counter=counter+1;
                        Candidatelist(1,Counter)=x(j,i);
                        Candidatelist(2,Counter)=weight(1,j);
                     END IF
                   END IF
             END FOR
  IF Candidatelist =! [ ]
    Tar(1,i)= Candidatelist(1,ROULETTE(Candidatelist (2,: ) )
   ELSE
     Nbr=RAND()*DD;
    WHILE isselected(1,Say) == 1
        Nbr=ROUND(RAND()*DD);
     END WHILE
      tar(1,i)=Nbr;
  END IF
END FOR
```

Figure 6.2 Determination of the superposition in p-WSAR in combinatorial problems

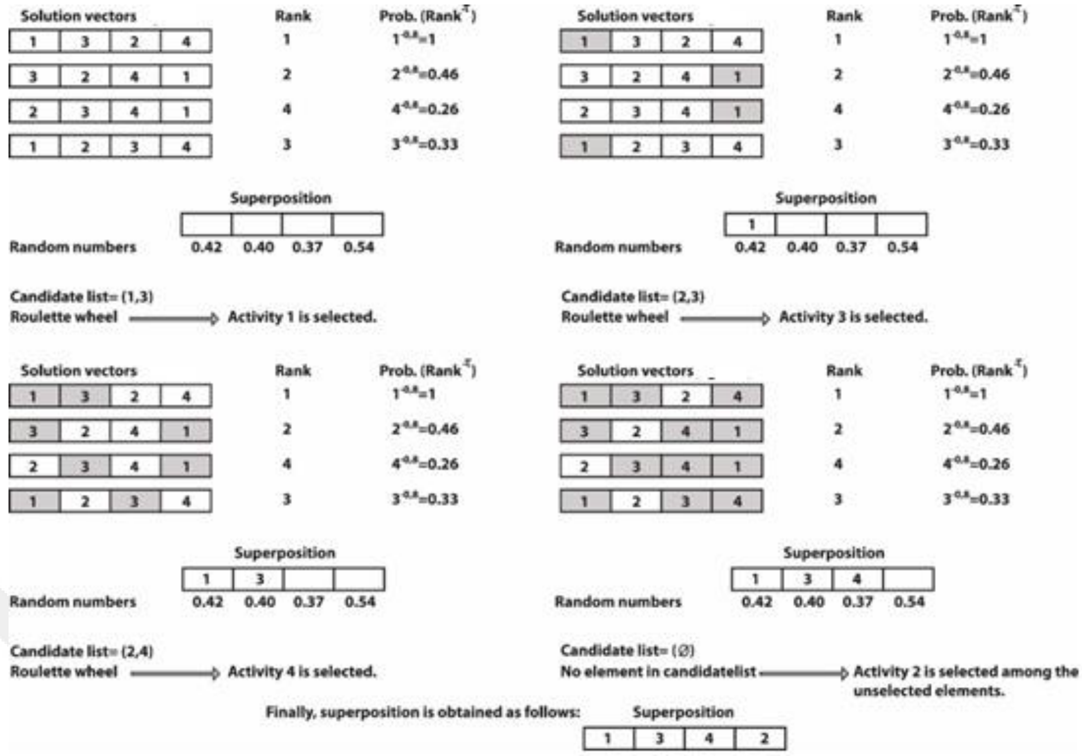


Figure 6.3 Determination of the superposition for permutation type vectors

Determining agents' next moves: In order to determine SSBMAs' next moves, every SSBMAs' fitness value and the fitness value of the attractive superposition is compared. If the fitness value of the attractive superposition is better than the SSBMA's fitness value, then the SSBMA applies the linear order-crossover (LOX) with the attractive superposition and try to escape from repulsive superposition (more information about LOX crossover is given later). Otherwise, one of the pre-defined move mechanisms (which are mentioned later) is randomly selected and employed to determine the SSBMA's new position. This procedure is illustrated in Figure 6.4. A critical issue, which also has a significant effect on the p-WSAR's performance, is the determination of step length in LOX and move mechanisms (*determination of number of swaps, inserts, length of inversion etc.*). A famous random walk mechanism, which is known as the Lévy flight, is incorporated into pWSAR in order to adjust step length. More details about random walk mechanism are given in Appendix.

```
FOR (i = 1 to NS) DO
      IF f(i) <= f(AS)
        LOCAL=RAND()*7;
        X(i)=NEIGHBOR(X(i),LOCAL)
      ELSE
        X(i)=LOX(X(i),AS);
     END IF
END FOR
```

Figure 6.4 Determination of the next moves of the SSBMAs in p-WSAR in combinatorial problems

Crossover mechanisms: LOX crossover is selected as the crossover mechanism in p-WSAR because of its ability to copy and transfer segment. The LOX mechanism was proposed by Falkenauer and Bouffoix (Falkenauer and Bouffouix, 1991) as a minor modification of the order crossover (OX) operator. The approach is quite similar, with the distinction that chromosomes (solution vectors/agents) are not considered cyclic. This implies that after copying the segment between the two cut points from the first parent into the offspring, the missing elements fill the unfilled genes in the offspring while maintaining the relative order of the second parent scanning from left to right. (Falkenauer and Bouffouix, 1991). A LOX crossover is shown in Figure 6.5.

Move mechanisms: Seven very famous neighborhood generation (move) mechanisms (variety of block swaps, insertions, ejection chains and inversion etc.) are employed in p-WSAR. The step length (i.e. number of single, two and three block swaps; length of inversion and ejection chain; and number and length of single and block insertions) is determined by making use of a famous random walk mechanism, which is known as Lévy flight in the literature. Detailed information and illustration of these move mechanisms and an example of calculation of one step length will be mentioned in Appendix.

Attractive superposition	3	1	4	6	7	5	2	9	8
Repulsive superposition	5	7	2	4	1	6	3	8	9
SSBMA solution vector	2	4	6	7	1	5	3	9	8
Offspring	2	1	4	6	7	5	9	3	8

Figure 6.5 LOX crossover

Random walk mechanism: As mentioned above the determination of the length of the segment for crossover and for some of the move mechanisms (also the number of swaps and insertions for the others) are important for diversification during the search, which also has critical importance in the algorithm's overall performance. Using a random walk strategy may prevent being trapped into local optima by its non-stationary structure. In this study, the Lévy flight procedure is employed as a random walk mechanism. Lévy flight is a random-walk procedure in which steps are delineated in terms of the step lengths having a certain probability distribution with the directions of the steps being isotropic and random. Lévy flights (Shlesinger et al., 1995), which implements a similar path to fruit flies or paths punctuated by a sudden 90^0 turn. Lévy flights is utilized in many swarm intelligence algorithms as discussed in Yang (Yang, 2010). The step length, sl_{ik} , of each SSBMA *i* at each iteration *k* is determined by using Eq. 1. In order to prevent exceeding limits (*length* of the agent's solution vector), a modulus operator is utilized in Eq.6.3. Moreover, the absolute value is taken in order to prevent negative step lengths as Eq. 6.4 can generate outputs in the interval [-1, 1].

$$sl_{ik} = mod(abs(sl_{i(k-1)} + Lévy), length) \tag{6.3}$$

The *Lévy* term in Eq. 6.3 is obtained as in Eq. 6.4, where an output (sl_{k+1}) is an input (sl_k) for the next number generation.

$$\left\{ \begin{array}{c} \omega = rnd(\,) \times 2\pi \\ \rho = rnd(\,)^{\left(\frac{-1}{\eta}\right)} \\ sl_{k+1} = sl_k + \rho \times \cos\omega \end{array} \right\} \tag{6.4}$$

A simplified overall procedure of the proposed p-WSAR is shown in Figure 6.6.

```
PROCEDURE p-WSAR
Randomly generate initial SSBMA population,
Sort the population according to fitness values
WHILE iteration<MaxIter
        Determine superpositions
        Calculate f(as)
             FOR each SSBMA in the population
                     if SSBMA's fitness<f(as)
                     Apply one of the neighborhood mechanisms
             ELSE
                     LOX- Crossover with AS
             END IF
             END FOR
             Sort population based on the fitness values
             Set SSBMAs parameters
END WHILE
RETURN best fitness
END PROCEDURE
```

Figure 6.6 A simplified overall procedure of p-WSAR

6.3 Experimental Results

6.3.1 Experimental results of RCPSP

The computational performance of p-WSAR is tested on the RCPSPs, which are selected from a publically available project scheduling library, ("Single Mode Data Sets," n.d.). This library, consisting of RCPSPs data sets ranging from 30 to 120 jobs in size, is generated by Kolisch and Sprecher (Kolisch and Sprecher, 1997). One can see the study (Kolisch and Sprecher, 1997) for further information about these datasets. All 480 problems for the J30 and J60 sets and all 600 problems of the J120 set are solved. A parameter, average deviation (Avg_Dev) is used to determine the quality of the solutions. For J30, the optimal value of each problem instance is known. The average deviation between solutions is obtained through the proposed algorithm and the optimal values are measured according to a formula which is shown in Equation 6.5. Besides, in the J60 and J120 problem sets, recent upper bounds and lower bounds are available. The lower bounds that are delineated by using the critical path make-span in the resource relaxation of the problem through the critical path method (CPM) by (“Single Mode Data Sets,” n.d.) are used for performance comparison with other approaches. The critical-path lower bound (Cplb) value is used to indirectly measure the optimal value. The average value of Cplb is represented in Equation 6.6. The effectiveness of the proposed algorithm is compared with 23 different metaheuristic algorithms from the literature. In order to achieve an unbiased comparison, the algorithm implemented according to 1000 and 5000 iterations in 20 independent runs which are commonly used in the literature. P-WSAR is implemented in MATLAB 2022a and all the experiments are performed on a PC with 3.20 GHz CPU and 16 GB RAM running under a Windows 10 operating system.

$$Avg_{Dev} = \frac{\sum_{instances}(\frac{Obtained-Optimal}{Optimal} X\ 100\%)}{instances} \tag{6.5}$$

$$Avg_{Dev_{cplb}} = \frac{\sum_{instances}(\frac{Obtained-Cplb}{Cplb} X\ 100\%)}{instances} \tag{6.6}$$

The percentage deviations from the optimal makespan for problem set J30 are depicted in Table 6.2. The p-WSAR algorithm is able to solve with an average deviation of 0.39 % and 0.22 % for 1000 and 5000 schedules from the optimal solutions respectively. The proposed algorithm is able to provide comparable and acceptable solutions.

For the J60 and J120 problem sets, the optimal solutions are not known, therefore results are reported concerning the average percentage deviation from the CPM based lower bound. A summary of the results for the J60 problem set is shown in Table 6.3. For the J60 problem set, the proposed algorithm surpasses all the other heuristics with an average deviation of 4.31 % and 3.07 % for 1000 and 5000 schedules from the lower bounds. A summary of the results for the J120 problem set is illustrated in Table 6.4. For the J120 problem set, the proposed algorithm is much better than all of the other algorithms with an average deviation of 15.88 % and 14.98 % for 1000 and 5000 schedules from the lower bounds.

Table 6.2 Average deviations from optimal solutions for the J30 problem set

		J30	
Algorithm	Reference	Number of Schedules	
		1000	5000
p-WSAR	Present work	0.39	0.22
cWSA	Baykasoğlu and Şenol (2019)	0.66	0.28
COA	Elsayed et al.(Elsayed et al., 2017)	0.04	0.00
Art. Imm. Alg.	Mobini et al. (Mobini et al., 2011)	0.05	0.03
GAPS	Mendes et al.(Mendes et al., 2009)	0.06	0.02
GA-TS-Path relinking	Kochetov and Stolyar (Kochetov and Stolyar,	0.10	0.04
Neurogenetic (FBI)	Agarwal et al.(Agarwal et al., 2011)	0.13	0.10
ACOSS	Chen et al. (Chen et al., 2010)	0.14	0.06
MAOA	Zheng and Wang (Zheng and Wang, 2015)	0.17	0.06
PSO-HH	Koulinas et al. (Koulinas et al., 2014)	0.26	0.04
SS-EM	Debels et al.(Debels et al., 2006)	0.27	0.11
Hybrid GA	Valls et al. (Valls et al., 2008)	0.27	0.06
JPSO	Chen (Chen, 2011)	0.29	0.14
Sampling-LFT,FBI	Tormos and Lova (Tormos and Lova, 2003)	0.30	0.16
Sampling-random,	Tormos and Lova (Tormos and Lova, 2001)	0.30	0.17
GA-FBI	Goncalves et al. (Gonçalves et al., 2011)	0.32	0.02
GA	Alcaraz and Maroto (Alcaraz and Maroto,	0.33	0.12
GA-FBI	Valls et al.(Valls et al., 2005)	0.34	0.20
SFL	Fang and Wang (Fang and Wang, 2012)	0.36	0.21
HEDA	Wang and Fang (Wang and Fang, 2012)	0.38	0.14
GA	Hartmann (Hartmann, 2002)	0.38	0.22
SA	Bouleinmen and Lecocq (Bouleimen and	0.38	0.23
TS	Klein (Klein, 2000)	0.42	0.17
BA-FBI	Ziarati et al. (Ziarati et al., 2011)	0.42	0.19
TS	Nonobe and Ibaraki (Nonobe and Ibaraki,	0.46	0.16
Sampling-random,	Valls et al. (Valls et al., 2008)	0.46	0.28
GA-activity list	Hartmann (Hartmann, 1998)	0.54	0.25
GANS	Proon and Jin (Proon and Jin, 2011)	1.83	1.27

Table 6.3 Average deviations from CPM based lower bound for the J60 problem set

Algorithm	Reference	J60 Number of Schedules 1000	5000
p-WSAR	Present work	4.31	3.07
cWSA	Baykasoğlu and Şenol (2019)	4.58	3.12
COA	Elsayed et al.(Elsayed et al., 2017)	11.13	10.77
Art. Imm. Alg.	Mobini et al. (Mobini et al., 2011)	11.17	10.80
GANS	Proon and Jin (Proon and Jin, 2011)	11.35	10.53
SFL	Fang and Wang (Fang and Wang, 2012)	11.44	10.87
Neurogenetic (FBI)	Agarwal et al.(Agarwal et al., 2011)	11.51	11.29
Hybrid GA	Valls et al. (Valls et al., 2008)	11.56	11.10
MAOA	Zheng and Wang (Zheng and Wang, 2015)	11.67	10.84
GA-TS-Path relinking	Kochetov and Stolyar (Kochetov and Stolyar,	11.71	11.17
GAPS	Mendes et al.(Mendes et al., 2009)	11.72	11.04
SS-EM	Debels et al (Debels et al., 2006)	11.73	11.10
PSO-HH	Koulinas et al. (Koulinas et al., 2014)	11.74	11.13
ACOSS	Chen et al. (Chen et al., 2010)	11.75	10.98
HEDA	Wang and Fang (Wang and Fang, 2012)	11.97	11.43
JPSO	Chen (Chen, 2011)	12.03	11.43
Sampling-random,	Tormos and Lova (Tormos and Lova, 2001)	12.14	11.82
Sampling-LFT,FBI	Tormos and Lova (Tormos and Lova, 2003)	12.18	11.87
GA-FBI	Valls et al (Valls et al., 2008)	12.21	11.27
GA	Hartmann (Hartmann, 2002)	12.21	11.70
BA-FBI	Ziarati et al. (Ziarati et al., n.d.)	12.55	12.04
GA-activity list	Hartmann (Hartmann, 1998)	12.68	11.89
SA	Bouleinmen and Lecocq (Bouleimen and	12.75	11.90
TS	Klein (Klein, 2000)	12.77	12.03
TS	Nonobe and Ibaraki (Nonobe and Ibaraki,	12.97	12.18
GA-FBI	Goncalves et al. (Gonçalves et al., 2011)	NA	11.56
Sampling-random,	Valls et al. (Valls et al., 2008)	NA	NA
GA	Alcaraz and Maroto (Alcaraz and Maroto,	NA	NA

Table 6.4 Average deviations from CPM based lower bound for the J120 problem set

Algorithm	Reference	J120 Number of Schedules	
		1000	5000
p-WSAR	Present work	15.88	14.97
cWSA	Baykasoğlu and Şenol (2019)	16.11	15.32
GANS	Proon and Jin (Proon and Jin, 2011)	33.45	31.51
MAOA	Zheng and Wang (Zheng and Wang, 2015)	33.87	32.64
Art. Imm. Alg.	Mobini et al. (Mobini et al., 2011)	34.01	32.57
COA	Elsayed et al.(Elsayed et al., 2017)	34.04	32.90
Hybrid GA	Valls et al. (Valls et al., 2008)	34.07	33.10
Neurogenetic (FBI)	Agarwal et al. (Agarwal et al., 2011)	34.65	34.15
GA-TS-Path relinking	Kochetov and Stolyar (Kochetov and Stolyar,	34.74	32.59
SFL	Fang and Wang (Fang and Wang, 2012)	34.83	33.20
ACOSS	Chen et al. (Chen et al., 2010)	35.19	32.48
PSO-HH	Koulinas et al. (Koulinas et al., 2014)	35.20	32.59
SS-EM	Debels et al.(Debels et al., 2006)	35.22	33.36
GA-FBI	Valls et al. (Valls et al., 2008)	35.39	32.54
Sampling-random,	Valls et al. (Valls et al., 2008)	35.39	33.24
HEDA	Wang and Fang (Wang and Fang, 2012)	35.44	33.61
JPSO	Chen (Chen, 2011)	35.71	33.88
GAPS	Mendes et al. (Mendes et al., 2009)	35.87	33.03
Sampling-random,	Tormos and Lova (Tormos and Lova, 2001)	36.24	35.56
Sampling-LFT,FBI	Tormos and Lova (Tormos and Lova, 2003)	36.49	35.81
GA	Hartmann (Hartmann, 2002)	37.19	35.39
BA-FBI	Ziarati et al. (Ziarati et al., 2011)	37.72	36.76
GA	Alcaraz and Maroto (Alcaraz and Maroto,	39.36	36.57
GA-activity list	Hartmann (Hartmann, 1998)	39.37	36.74
TS	Nonobe and Ibaraki (Nonobe and Ibaraki,	40.86	37.88
SA	Bouleinmen and Lecocq (Bouleimen and	42.81	37.68
GA-FBI	Goncalves et al. (Gonçalves et al., 2011)	NA	35.94
TS	Klein (Klein, 2000)	NA	NA

6.3.2 Experimental results of PFSP

The p-WSAR's performance in PFSP is evaluated using the (Taillard, 1990) benchmark instances, which are divided into 12 groups of problems. 5 of these problems are selected to test p-WSAR’s performance against some state of the art algorithms and WSAR. These problems’ size (PS: (J*M) and well known solutions (WKS) is given in Table 6.5. The best, the worst and the average performance of 30 runs of each algorithm is recorded. In all of the instances, p-WSAR is able to find better solutions than other algorithms.

Table 6.5 Comparison of p-WSAR with some state of the art algorithms and WSAR

Problems	Algorithm	TLBO (Baykasoğlu et. Al, 2014)	HPSO (Lin et al., 2010)	NPSO (Lian et al., 2008)	WSAR	p-WSAR
ta001 PS:(20*5) WKS:1278	Best	1278	1278	1278	1278	1278
	Worst	1297	1278	1297	1297	1278
	Average	1287.2	1278	1279.9	1278.6	1278
ta011 PS:(20*10) WKS:1582	Best	1586	1582	1582	1586	1582
	Worst	1618	1596	1639	1618	1582
	Average	1606	1587.3	1605.8	1592.2	1582
ta031 PS:(50*5) WKS:2724	Best	2724	2724	2724	2724	2724
	Worst	2741	2724	2729	2729	2724
	Average	2729.4	2724	2725	2724.6	2724
ta051 PS:(50*20) WKS:3771	Best	3986	3923	3938	3969	3902
	Worst	4095	3963	3989	4063	3923
	Average	4029.7	3944.6	3964.3	4015.9	3916
ta061 PS:(100*5) WKS:5493	Best	5493	5493	5493	5493	5493
	Worst	5527	5493	5495	5495	5493
	Average	5499.4	5493	5493.2	5493.2	5493

In addition, the performance of p-WSAR is statistically compared with the other al-gorithms through nonparametric statistical tests by using average values. Table 6.6 indi-cates that (based on the Friedman test results) p-WSAR surpasses the other algorithms. Furthermore, according to the Wilcoxon signed-rank test, the difference between p-WSAR and HPSO is found negligible as the $p > 0.1$. Besides p-WSAR is performed slightly better than TLBO, NPSO, WSAR as $p<0.1$.

Table 6.6 Non-parametric test results on Taillard Instances

Friedman test average rankings		Wilcoxon signed-rank test between p-WSAR and state of the art algorithms	
Algorithms	Sum of Ranks	p-WSAR vs.	p-value
TLBO	5.0 (5)	TLBO	0.0625
HPSO	1.7 (2)	HPSO	0.5
NPSO	3.7 (4)	NPSO	0.0625
WSAR	3.3 (3)	WSAR	0.0625
p-WSAR	1.3 (1)		

Another computational study is organized to test the performance of p-WSAR with its constituents (SSBMAs) in terms of solution quality. The results are presented in Table 6.7, and Table 6.8. According to the computational results, p-WSAR' performance is far beyond its constituents (SSBMAs). Besides, in respect of the non-parametric statistical tests, p-WSAR is able to produce more effective results than its constituents. Also, there is statistically significant difference between the performance of the p-WSAR and its constituents since p-value is < 0.1.

Table 6.7 Comparison of p-WSAR with SSBMAs

Problems	Algorithm	SA	RS	GD	TA	GS	p-WSAR
ta001	Best	1286	1294	1278	1278	1284	1278
PS:(20*5)	Worst	1297	1302	1297	1284	1292	1278
WKS:1278	Average	1292.2	1296.5	1279.9	1280.6	1287.8	1278
ta011	Best	1606	1616	1596	1592	1608	1582
PS:(20*10)	Worst	1620	1650	1616	1618	1642	1582
WKS:1582	Average	1610	1632.4	1610.7	1608	1624	1582
ta031	Best	2804	2942	2806	2864	2916	2724
PS:(50*5)	Worst	2908	3026	2846	2938	3002	2724
WKS:2724	Average	2856	2978	2824.6	2886	2984	2724
ta051	Best	4206	4807	4402	4622	4424	3902
PS:(50*20)	Worst	4240	6240	4803	5162	6024	3923
WKS:3771	Average	4222.4	5465.8	4627	4838.6	5146	3916
ta061	Best	6122	8640	6248	6125	7426	5493
PS:(100*5)	Worst	6378	9026	6414	6642	8424	5493
WKS:5493	Average	6564.3	8924.7	6344.9	6348.4	8012.6	5493

Table 6.8 Non-parametric test results on Taillard Instances p-WSAR vs. SSBMAs

Friedman test average rankings		Wilcoxon signed-rank test between p-WSAR and state of the art algorithms	
Algorithms	Sum of Ranks	p-WSAR vs.	p-value
SA	3.4 (4)	SA	0.0625
RS	5.8 (6)	RS	0.0625
GD	2.6 (2)	GD	0.0625
TA	3.2 (3)	TA	0.0625
GS	5.0 (5)	GS	0.0625
p-WSAR	1.0 (1)		

6.4 Conclusions

In this chapter, we proposed a combinatorial version of p-WSAR. The p-WSAR algorithm incorporates a novel procedure for determining superposition for permutation type agents (solution vectors) without needing to use an indirect representation mechanism like random keys. This new technique can also be incorporated into other metaheuristic algorithms. The performance of the p-WSAR is tested on several resource constrained project scheduling problems and flow shop scheduling problems from the literature. The results of an extensive computational study show that p-WSAR is able to provide very competitive results in comparison to existing algorithms. We aim to apply p-WSAR to several other static and dynamic combinatorial optimization problems with real life applications as future research.

CHAPTER 7
CONCLUSION

The No Free Lunch Theorem claims that there is no statistical difference between the performances of optimization algorithms if we try to solve all optimization problems. This study is a research attempt that combines different metaheuristic algorithms in order to provide a satisfactory performance on solving the optimization problems by benefiting their diverse characteristics. In addition, parallel execution of metaheuristic algorithms is very effective in solving large-scale optimization problems where sequential methods cannot provide solutions in reasonable computational times.

Based on this motivation, several metaheuristic algorithms are brought together to form a coalition in a parallel computing environment for solving optimization problems in this thesis. We proposed the p-WSAR algorithm that composes of five single solutions based metaheuristic algorithms, namely, Random Search (RS) (Rogers, 1972), Threshold Accepting (TA) (Dueck & Scheuer, 1990), Great Deluge (GD) (Dueck, 1993), Greedy Search (GS) (Feo & Resende, 1995), Simulated Annealing (SA) (Kirkpatrick et al., 1983) and the WSAR algorithm of Baykasoğlu (2020, 2021). The proposed methodology runs different single solution based metaheuristic algorithms (SSBMAs) in parallel and employs WSAR as the controller. While SSBMAs are responsible for the exploration of the search space, WSAR algorithm serves as a controller and guides the whole searching process.

In Chapter 4, the proposed approach is tested against the latest well-known and highly complex unconstrained continuous optimization problems (CEC2020). Results show that the proposed approach is capable of finding as effective solutions as the state of the art metaheuristic algorithms. Besides, the performance of the proposed algorithm is also compared with its constituents (SSBMAs and WSAR).

The Chapter 5 analyzes the performance of the p-WSAR in binary optimization problems. In this respect, p-WSAR is modified for binary spaces. A specialized roulette wheel procedure is used in this modification. Additionally, uniform-based crossover and random bit flips are employed while generating a new solution. As secondarily, a systematically controlled step length mechanism to put control on the search speed is adopted in p-WSAR. Step length, which is indeed the number of bits to be flipped here, is exponentially decreased throughout generations. Thus, while p-WSAR performs a more diversified search at the initialization stage, it is encouraged for intensification around the found promising regions towards end of a run. The performance of p-WSAR is analyzed on some of the well-known binary optimization problems, including uncapacitated facility location problem, 0-1 knapsack problem and set union knapsack problem, which have numerous application in real-life. Promising results and statistical tests point out the effectiveness of p-WSAR.

In Chapter 6, we proposed a combinatorial version of p-WSAR. The p-WSAR algorithm incorporates procedure for determining superposition for permutation type agents (solution vectors) without needing to use an indirect representation mechanism like random keys. The performance of the p-WSAR is tested on several resource constrained project scheduling problems and flow shop scheduling problems from the literature. The results of an extensive computational study show that p-WSAR is able to provide very competitive results in comparison to existing algorithms.

The computational results prove the rationale of the proposed approach. SSBMAs and WSAR cannot produce effective solutions when they are run individually. However, when they come together (by forming a coalition), they can produce very effective results. On the other hand, thanks to parallel computing, algorithms could work simultaneously and runtime is considerable shortened.

In this study MATLAB parallel programming tool box is utilized in a single computer. The number of cores run SSBMAs in parallel. As a future research direction, different computers can get together on a network for collaboration while solving optimization problems. By this way, it is predicted to shorten solution time much more.

REFERENCES

Ab Wahab, M.N., Nefti-Meziani, S., & Atyabi, A., (2015). A comprehensive review of swarm optimization algorithms. *PLoS One 10*, 1–36. https://doi.org/10.1371/journal.pone.0122827

Abdelhafez, A., Alba, E., & Luque, G. (2019). Performance analysis of synchronous and asynchronous distributed genetic algorithms on multiprocessors. *Swarm and Evolutionary Computation, 49*, 147–157. https://doi.org/10.1016/j.swevo.2019.06.003

Abed-alguni, B. H., & Alawad, N. A. (2021). Distributed Grey Wolf Optimizer for scheduling of workflow applications in cloud environments. *Applied Soft Computing, 102*, 107113. https://doi.org/10.1016/j.asoc.2021.107113

Agarwal, A., Colak, S., & Erenguc, S. (2011). A Neurogenetic approach for the resource-constrained project scheduling problem. *Computers & Operations Research 38*, 44–50. https://doi.org/10.1016/j.cor.2010.01.007

Alba, E. (2005). Parallel Metaheuristics: A New Class of Algorithms. *In Parallel Metaheuristics: A New Class of Algorithms.* https://doi.org/10.1002/0471739383

Alba, E., & Troya, J. M. (2002). Improving flexibility and efficiency by adding parallelism to genetic algorithms. *Statistics and Computing, 12(2)*, 91–114. https://doi.org/10.1023/A:1014803900897

Alcaraz, J., & Maroto, C. (2001). A Robust Genetic Algorithm for Resource Allocation in Project Scheduling. *Annals of Opererations Research, 102*, 83–109. https://doi.org/10.1023/A:1010949931021

Al-Sultan, K. S., & Al-Fawzan, M. A. (1999). A tabu search approach to the uncapacitated facility location problem. *Annals of Operations Research, 86*, 91-103. doi: 10.1023/A:1018956213524

Anandaraman, C. (2011). An improved sheep flock heredity algorithm for job shop scheduling and flow shop scheduling problems. *International Journal of Industrial Engineering Computations, 2,* 749–764. https://doi.org/10.5267/j.ijiec.2011.06.007

Ang, K. M., Lim, W. H., Isa, N. A. M., Tiang, S. S., & Wong, C. H. (2020). A constrained multi-swarm particle swarm optimization without velocity for constrained optimization problems. *Expert Systems with Applications, 140*, 112882. https://doi.org/10.1016/j.eswa.2019.112882

Arulselvan, A. (2014). A note on the set union knapsack problem. *Discrete Applied Mathematics, 169,* 214-218. doi: 10.1016/j.dam.2013.12.015

Ashlock, D. (2006). *Evolutionary computation for modeling and optimization*. New York: Springer. https://doi.org/10.1007/0-387-31909-3

Barcelo, J., Hallefjord, Å., Fernandez, E., & Jörnsten, K. (1990). Lagrangean relaxation and constraint generation procedures for capacitated plant location problems with single sourcing. *OR Spectrum, 12*(2), 79-88. doi: 10.1007/BF01784983

Barr, R. S., & Hickman, B. L. (1993). Feature Article—Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Experts' Opinions. *ORSA Journal on Computing, 5*(1), 2–18. https://doi.org/10.1287/ijoc.5.1.2

Baykasoglu, A. (2001). MOAPPS 1.0: Aggregate production planning using the multiple-objective tabu search. *International Journal of Production Research, 39*(16), 3685–3702. https://doi.org/10.1080/00207540110061607

Baykasoglu, A. (2012). Design optimization with chaos embedded great deluge algorithm. *Applied Soft Computing, 12*(3), 1055-1067. doi: 10.1016/j.asoc.2011.11.018

Baykasoğlu, A. (2021). Optimising cutting conditions for minimising cutting time in multi-pass milling via weighted superposition attraction-repulsion (WSAR) algorithm. *International Journal of Production Research, 59*(15), 4633-4648. https://doi.org/10.1080/00207543.2020.1767313

Baykasoğlu, A., & Akpinar, Ş. (2015). Weighted superposition attraction (WSA): A swarm intelligence algorithm for optimization problems-Part 2: constrained optimization. *Applied Soft Computing, 37*, 396-415. doi: 10.1016/j.asoc.2015.08.052

Baykasoğlu, A., & Akpinar, Ş. (2017). Weighted Superposition Attraction (WSA): A swarm intelligence algorithm for optimization problems-Part 1: Unconstrained optimization. *Applied Soft Computing, 56,* 520-540. doi: 10.1016/j.asoc.2015.10.036

Baykasoglu, A. , & Baykasoglu, C. (2020), Optimal design of truss structures using weighted superposition attraction algorithm. *Engineering with Computers, 36*(3), 965-979.

Baykasoglu, A. , & Baykasoglu, C. (2021). Weighted superposition attraction-repulsion (WSAR) algorithm for truss optimization with multiple frequency constraints. *Structures, 30*, 253-264.

Baykasoglu, A. , & Durmusoglu, Z.D. (2014). A hybrid MCDM for private primary school assessment using DEMATEL based on ANP and fuzzy cognitive map. *International Journal of Computational Intelligence Systems, 7*(4), 615-635.

Baykasoglu, A., Hamzadayi, A., & Akpinar, S. (2019). Single Seekers Society (SSS): bringing together heuristic optimization algorithms for solving complex problems. *Knowledge-Based Systems, 165*, 53-76.

Baykasoğlu, A., Hamzadayi, A., & Köse, S. Y. (2014). Testing the performance of teaching–learning based optimization (TLBO) algorithm on combinatorial problems: Flow shop and job shop scheduling cases. *Information Sciences, 276*, 204-218. https://doi.org/10.1016/j.ins.2014.02.056

Baykasoğlu, A., & Ozsoydan, F. B. (2015). Adaptive firefly algorithm with chaos for mechanical design optimization problems. *Applied Soft Computing, 36*, 152-164. doi: 10.1016/j.asoc.2015.06.056

Baykasoğlu, A., Ozsoydan, F. B., & Senol, M. E. (2020). Weighted superposition attraction algorithm for binary optimization problems. *Operational Research*, *20*(4), 2555-2581. doi:10.1007/s12351-018-0427-9

Baykasoğlu, A., & Şenol, M. E. (2016, December). *Oppositon-based weighted superposition attraction algorithm for travelling salesman problems.* In LMSCM 2016 XIV. Internatıonal Logıstıcs And Supply Chaın Congress, *314*. Izmir, Turkey.

Baykasoğlu, A., & Şenol, M. E. (2019). Weighted superposition attraction algorithm for combinatorial optimization. *Expert Systems with Applications, 138*. https://doi.org/10.1016/j.eswa.2019.07.009

Baykasoğlu, A., & Şenol M.E. (2016). *Combinatorial optimization via weighted superposition attraction.* International Conference on Operations Research of the GOR, OR 2016, Hamburg.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, *6*(2), 154-160.https://doi.org/10.1287/ijoc.6.2.154

Bhattacharjee, K. K., & Sarmah, S. P. (2014). Shuffled frog leaping algorithm and its application to 0/1 knapsack problem. *Applied Soft Computing, 19,* 252-263. doi: 10.1016/j.asoc.2014.02.010

Bhattacharjee, K. K., & Sarmah, S. P. (2017). Modified swarm intelligence based techniques for the knapsack problem. *Applied Intelligence, 46*(1), 158-179. doi: 10.1007/s10489-016-0822-y

Bouleimen, K., & Lecocq, H., (2003). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal Of Operational Research*, *149*(2), 268-281. https://doi.org/10.1016/S0377-2217(02)00761-0

Chen, R.M. (2011). Particle swarm optimization with justification and designed mechanisms for resource-constrained project scheduling problem. *Expert Syst. Appl. 38*, 7102–7111. https://doi.org/10.1016/j.eswa.2010.12.059

Chen, W., Shi, Y. J., Teng, H. F., Lan, X. P., & Hu, L. C. (2010). An efficient hybrid algorithm for resource-constrained project scheduling. *Information Sciences*, *180*(6), 1031-1039.

Chi, Y., & Cai, G. (2010, March). *Particle swarm optimization with opposition-based disturbance.* In 2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics (CAR 2010), 2, 223-226. IEEE.

Cornuejols, M.L, Nemhauser, G.L., Wolsey, L.A. *The uncapacitated facility location problem,* in: R.L. Francis, P. Mirchandani (Eds.), Discrete Location Theory, Wiley Interscience, New York, 1990.

de Armas, J., Juan, A. A., Marquès, J. M., & Pedroso, J. P. (2017). Solving the deterministic and stochastic uncapacitated facility location problem: from a heuristic to a simheuristic. *Journal of the Operational Research Society*, *68*(10), 1161-1176. doi: 10.1057/s41274-016-0155-6

de Sousa Junior, W. T., Montevechi, J. A. B., Miranda, R. de C., de Oliveira, M. L. M., & Campos, A. T. (2020). Shop floor simulation optimization using machine learning to improve parallel metaheuristics. *Expert Systems with Applications, 150*, 113272. https://doi.org/10.1016/j.eswa.2020.113272

Debels, D., De Reyck, B., Leus, R., & Vanhoucke, M. (2006). A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. . *European Journal Of Operational Research 169*, 638–653. https://doi.org/10.1016/j.ejor.2004.08.020

Della Croce, F., Salassa, F., & Scatamacchia, R. (2017). A new exact approach for the 0-1 Collapsing Knapsack Problem. *European Journal of Operational Research, 260*(1), 56-69.b doi: 10.1016/j.ejor.2016.12.009

Della Croce, F., Salassa, F., & Scatamacchia, R. (2017). An exact approach for the 0-1 knapsack problem with setups. *Computers & Operations Research, 80*, 61-67. doi: 10.1016/j.cor.2016.11.015

Diallo, M. H., August, M., Hallman, R., Kline, M., Slayback, S. M., & Graves, C. (2017). AutoMigrate: a framework for developing intelligent, self-managing cloud services with maximum availability. *Cluster Computing, 20*(3), 1995-2012. doi: 10.1007/s10586-017-0900-x

Dokeroglu, T., Pehlivan, S., & Avenoglu, B. (2020). Robust parallel hybrid artificial bee colony algorithms for the multi-dimensional numerical optimization. *Journal of Supercomputing, 76*(9), 7026–7046. https://doi.org/10.1007/s11227-019-03127-7

Dorigo, M, Maniezzo, V, Colorni, A. (1991). Po*sitive feedback as a search strategy.* Politecnico di Milano, Italy, (Technical Report No: 91-016).

Maniezzo, V., Dorigo, M., & Colorni, A. (1996). The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26*(1), 29-41.

Dueck, G. (1993). New optimization heuristics; The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics, 104*(1), 86–92. https://doi.org/10.1006/jcph.1993.1010

Dueck, G., & Scheuer, T. (1990). Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics, 90*(1), 161–175. https://doi.org/10.1016/0021-9991(90)90201-B

Elsayed, S., Sarker, R., Ray, T., & Coello, C. C. (2017). Consolidated optimization algorithm for resource-constrained project scheduling problems. *Information Sciences*, *418,* 346-362. https://doi.org/10.1016/j.ins.2017.08.023

Ergezer, M., & Simon, D. (2011, June). *Oppositional biogeography-based optimization for combinatorial problems.* In 2011 IEEE Congress of Evolutionary Computation (CEC) (pp. 1496-1503). IEEE. https://doi.org/10.1109/CEC.2011.5949792

Erlenkotter, D. (1978). A dual-based procedure for uncapacitated facility location. *Operations Research, 26(*6), 992-1009. doi: 10.1287/opre.26.6.992

Falkenauer, E., & Bouffouix, S. (1991, April). A genetic algorithm for job shop. In *ICRA* , 824-829.

Fang, C., & Wang, L. (2012). An effective shuffled frog-leaping algorithm for resource-constrained project scheduling problem. *Computers & Opererations Research 39,* 890–901. https://doi.org/10.1016/j.cor.2011.07.010

Feng, Y., Wang, G. G., Deb, S., Lu, M., & Zhao, X. J. (2017). Solving 0-1 knapsack problem by a novel binary monarch butterfly optimization. *Neural computing and applications*, *28*(7), 1619-1634. doi: 10.1007/s00521-015-2135-1

Feo, T. A., & Resende, M. G. C. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, *6*(2), 109–133. https://doi.org/10.1007/BF01096763

Gmys, J., Carneiro, T., Melab, N., Talbi, E. G., & Tuyttens, D. (2020). A comparative study of high-productivity high-performance programming languages for parallel metaheuristics. *Swarm and Evolutionary Computation, 57*, 100720. https://doi.org/10.1016/j.swevo.2020.100720

Goldschmidt, O., Nehme, D., & Yu, G. (1994). Note: On the set-union knapsack problem. *Naval Research Logistics (NRL), 41*(6), 833-842.

Gonçalves, J.F., Resende, M.G.C., & Mendes, J.J.M. (2011). A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. *Journal of Heuristics 17*, 467–486. https://doi.org/10.1007/s10732-010-9142-2

Gonçalves-e-Silva, K., Aloise, D., & Xavier-de-Souza, S. (2018). Parallel synchronous and asynchronous coupled simulated annealing. *Journal of Supercomputing, 74*(6), 2841–2869. https://doi.org/10.1007/s11227-018-2327-4

Gudder, S.P. (1970). A Superposition Principle in Physics. *Journal of Mathematical Physics 11*, 1037–1040. https://doi.org/10.1063/1.1665193

Guner, A. R., & Sevkli, M. (2008). A discrete particle swarm optimization algorithm for uncapacitated facility location problem. *Journal of Artificial Evolution and Applications.* doi: 10.1155/2008/861512

Hansen, N., & Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *In Evolutionary computation 9*(2), 159–195. MIT Press. https://doi.org/10.1162/106365601750190398

Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics (NRL), 45*(7), 733-750.

Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics (NRL) 49*, 433–448. https://doi.org/10.1002/nav.10029

He, Y., Xie, H., Wong, T. L., & Wang, X. (2018). A novel binary artificial bee colony algorithm for the set-union knapsack problem. *Future Generation Computer Systems 78*, 77-86. doi: 10.1016/j.future.2017.05.044

Heidari, A. A., Mirjalili, S., Faris, H., Aljarah, I., Mafarja, M., & Chen, H. (2019). Harris hawks optimization: Algorithm and applications. *Future Generation Computer Systems 97*, 849–872. https://doi.org/10.1016/j.future.2019.02.028

Herrán, A., Colmenar, J. M., Martí, R., & Duarte, A. (2020). A parallel variable neighborhood search approach for the obnoxious p -median problem. *International Transactions in Operational Research, 27*(1), 336–360. https://doi.org/10.1111/itor.12510

Holland, J.H. *Adaptation in Natural and Artificial Systems,* The University of Michigan Press, Ann Arbor, MI, 1975.

Holmberg, K. (1999). Exact solution methods for uncapacitated location problems with convex transportation costs. *European Journal of Operational Research, 114*(1), 127-140. doi: 10.1016/S0377-2217(98)00039-3

Jaramillo, J. H., Bhadury, J., & Batta, R. (2002). On the use of genetic algorithms to solve location problems. *Computers & Operations Research, 29*(6), 761-779. doi: 10.1016/S0305-0548(01)00021-1

Karaboga, D. (2005). *An idea based on honey bee swarm for numerical optimization 200,* 1-10. Technical report-tr06, Erciyes University, Engineering Faculty, Computer Engineering Department.

Kellerer, H., Pferschy, U., & Pisinger, D., *Knapsack Problems*, Springer, Berlin, 2004.

Kennedy, J., & Eberhart, R., (1995). *Particle swarm optimization.* In proceedings of IEEE International Conference on Neural Networks, 4, 1942-1948, Perth, WA, Australia. doi: 10.1109/ICNN.1995.488968

Kim, J. L., & Ellis Jr, R. D. (2010). Comparing schedule generation schemes in resource-constrained project scheduling using elitist genetic algorithm. *Journal of construction engineering and management*, *136*(2), 160-169.

Kim, Y. H., Yoon, Y., & Geem, Z. W. (2019). A comparison study of harmony search and genetic algorithm for the max-cut problem. *Swarm and Evolutionary Computation, 44,* 130–135. https://doi.org/10.1016/j.swevo.2018.01.004

Kiran, M. S. (2015). The continuous artificial bee colony algorithm for binary optimization. *Applied Soft Computing, 33,* 15-23. doi: 10.1016/j.asoc.2015.04.0

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). *Optimization by simulated annealing. Science, 220*(4598), 671–680. https://doi.org/10.1126/science.220.4598.671

Kizys, R., Juan, A. A., Sawik, B., & Calvet, L. (2019). A biased-randomized iterated local search algorithm for rich portfolio optimization. *Applied Sciences (Switzerland), 9*(17). https://doi.org/10.3390/app9173509

Klein, R. (2000). Project scheduling with time-varying resource constraints. *International Journal of Production Research*, *38*(16), 3937-3952. https://doi.org/10.1080/00207540050176094

Kochetov, Y., & Stolyar, A. (2003). *Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem.* In Proceedings of the 3rd international workshop of computer science and information technologies 132. https://doi.org/10.1080/15235882.2000.10162765

Kolisch, R., & Sprecher, A. (1997). PSPLIB-a project scheduling problem library: OR software-ORSEP operations research software exchange program. *European journal of operational research*, *96*(1), 205-216. https://doi.org/10.1016/S0377-2217(96)00170-1

Koulinas, G., Kotsikas, L., & Anagnostopoulos, K. (2014). A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem. *Information Sciences*, *277*, 680-693. https://doi.org/10.1016/j.ins.2014.02.155

Laporte, G., Nickel, S., & da Gama, F. S. *Location science*, Berlin: Springer, 2015.

Laporte, G., Nobert, Y., & Taillefer, S. (1987). A branch-and-bound algorithm for the asymmetrical distance-constrained vehicle routing problem. *Mathematical Modelling, 9*(12), 857–868. https://doi.org/10.1016/0270-0255(87)90004-2

Leite, N., Melício, F., & Rosa, A. C. (2019). A fast simulated annealing algorithm for the examination timetabling problem. *Expert Systems with Applications, 122*, 137–151. https://doi.org/10.1016/j.eswa.2018.12.048

Li, Y., Soleimani, H., & Zohal, M. (2019). An improved ant colony optimization algorithm for the multi-depot green vehicle routing problem with multiple objectives. *Journal of Cleaner Production, 227*, 1161–1172. https://doi.org/10.1016/j.jclepro.2019.03.185

Li, Z.K. & Li, N., (2009). *A novel multi-mutation binary particle swarm optimization for 0/1 knapsack problem*. In Proceedings of IEEE Control and Decision Conference, 3042-3047, Guilin, China. doi: 10.1109/CCDC.2009.5192838

Lian, Z., Gu, X., & Jiao, B. (2008). A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan. *Chaos, Solitons and Fractals 35*, 851–861. https://doi.org/10.1016/j.chaos.2006.05.082

Lin, F. T. (2008). Solving the knapsack problem with imprecise weight coefficients using genetic algorithms. *European Journal of Operational Research, 185*(1), 133-145. doi: 10.1016/j.ejor.2006.12.046

Lin, S.Y., Horng, S.J., Kao, T.W., Huang, D.K., Fahn, C.S., Lai, J.L., Chen, R.J., & Kuo, I.H. (2010). An efficient bi-objective personnel assignment algorithm based on a hybrid particle swarm optimization model. *Expert Systems with Applications 37*, 7825–7830. https://doi.org/10.1016/j.eswa.2010.04.056

Lister, W., Laycock, R. G., & Day, A. M. (2010,December). *A dynamic cache for real-time crowd rendering*. In Computer Graphics Forum. Oxford, UK.

Liu, Y., Liu, & C., (2009). *A schema-guiding evolutionary algorithm for 0-1 knapsack problem.* In Proceedings of IEEE Computer Science and Information Technology - Spring Conference, 160-164, Singapore, Singapore. doi: 10.1109/IACSIT-SC.2009.31

Lu, C., Gao, L., Pan, Q., Li, X., & Zheng, J. (2019). A multi-objective cellular grey wolf optimizer for hybrid flowshop scheduling problem considering noise pollution. *Applied Soft Computing Journal*, *75*, 728–749. https://doi.org/10.1016/j.asoc.2018.11.043

Malisia, A. R. (2007). *Investigating the application of opposition-based ideas to ant algorithms* [Unpublished Master's thesis] University of Waterloo.

Martin, S., Ouelhadj, D., Beullens, P., Ozcan, E., Juan, A.A., & Burke, E.K. (2016). A multi-agent based cooperative approach to scheduling and routing. *European Journal of Operational Research*, *254*(1), 169-178.

Meignan, D., Koukam, A., & Creput, J.C. (2010). Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, *16*(6), 859-879.

Memeti, S., Pllana, S., Binotto, A., Kołodziej, J., & Brandic, I. (2019). Using meta-heuristics and machine learning for software optimization of parallel computing systems: a systematic literature review. *Computing*, *101*(8), 893–936. https://doi.org/10.1007/s00607-018-0614-9

Mendes, J. J., Gonçalves, J. F., & Resende, M. G. (2009). A random key based genetic algorithm for the resource constrained project scheduling problem. *Computers & operations research*, *36*(1), 92-109. https://doi.org/10.1016/j.cor.2007.07.001

Miandoab, M. H. (2013). A new discrete Cuckoo Search for Resource Constrained Project Scheduling Problem (rcPSP). *International Journal of Advanced Studies in Humanities and Social Science*, *1*(11), 2368-2378.

Mirjalili, S., Gandomi, A. H., Mirjalili, S. Z., Saremi, S., Faris, H., & Mirjalili, S. M. (2017). Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems. *Advances in Engineering Software*, *114*, 163–191. https://doi.org/10.1016/j.advengsoft.2017.07.002

Mirjalili, S., Lewis, A., & Sadiq, A. S. (2014). Autonomous Particles Groups for Particle Swarm Optimization. *Arabian Journal for Science and Engineering*, *39*(6), 4683–4697. https://doi.org/10.1007/s13369-014-1156-x

Mobini, M., Mobini, Z., & Rabbani, M. (2011). An Artificial Immune Algorithm for the project scheduling problem under resource constraints. *Applied Soft Computing Journal 11*, 1975–1982. https://doi.org/10.1016/j.asoc.2010.06.013

Munguía, L. M., Ahmed, S., Bader, D. A., Nemhauser, G. L., & Shao, Y. (2018). Alternating criteria search: A parallel large neighborhood search algorithm for mixed integer programs. *Computational Optimization and Applications, 69*(1), 1–24. https://doi.org/10.1007/s10589-017-9934-5

Nonobe, K., & Ibaraki, T. (2002). Formulation and tabu search algorithm for the resource constrained project scheduling problem. In *Essays and surveys in metaheuristics*, 557-588. Springer, Boston, MA. https://doi.org/10.1007/978-1-4615-1507-4_25

Onwubolu, G., & Davendra, D. (2006). Scheduling flow shops using differential evolution algorithm. *European Journal of Operational research*, *171*(2), 674-692. https://doi.org/10.1016/j.ejor.2004.08.043

Ozturk, C., Hancer, E., & Karaboga, D. (2015). A novel binary artificial bee colony algorithm based on genetic operators. *Information Sciences, 297*, 154-170. doi: 10.1016/j.ins.2014.10.060

Özbakır, L., & Turna, F. (2017). Clustering performance comparison of new generation meta-heuristic algorithms. *Knowledge-Based Systems 130* (2017) 1-16. doi: 10.1016/j.knosys.2017.05.023

Paliszkiewicz, J. (2007). Knowledge Management: An Integratıve Vıew And Empırıcal Examınatıon. Cybernetics and Systems: *An International Journal, 38*(8), 825–836. https://doi.org/10.1080/01969720701601148

Palomo-Romero, J. M., Salas-Morera, L., & García-Hernández, L. (2017). An island model genetic algorithm for unequal area facility layout problems. *Expert Systems with Applications, 68*, 151–162. https://doi.org/10.1016/j.eswa.2016.10.004

Panigrahi, B.K., Shi, Y., Lim, M.-H., 2011. *Handbook of swarm intelligence : concepts, principles and applications*. Springer-Verlag.

Pierezan, J., & Dos Santos Coelho, L. (2018). *Coyote Optimization Algorithm: A New Metaheuristic for Global Optimization Problems*. 2018 IEEE Congress on Evolutionary Computation, CEC 2018 - Proceedings. https://doi.org/10.1109/CEC.2018.8477769

Pisinger, D., & Saidi, A. (2017). Tolerance analysis for 0-1 knapsack problems. *European Journal of Operational Research, 258*(3), 866-876. doi: 10.1016/j.ejor.2016.10.054

Precup, R.-E., David, R.-C., Roman, R.-C., Petriu, E. M., & Szedlak-Stinean, A.-I. (2021). Slime Mould Algorithm-Based Tuning of Cost-Effective Fuzzy Controllers for Servo Systems. *International Journal of Computational Intelligence Systems*, *14*(1), 1042–1052. https://doi.org/10.2991/IJCIS.D.210309.001

Preitl, Z., Precup, R. E., Tar, J. K., & Takács, M. (2006). *Use of multi-parametric quadratic programming in fuzzy control systems.* Acta Polytechnica Hungarica, 3(3), 29–43.

Proon, S., & Jin, M. (2011). A genetic algorithm with neighborhood search for the resource-constrained project scheduling problem. *Naval Research Logistics (NRL)*, *58*(2), 73-82. https://doi.org/10.1002/nav.20439

Rahnamayan, S., Tizhoosh, H. R., & Salama, M. M. (2008). Opposition-based differential evolution. *IEEE Transactions on Evolutionary computation*, *12*(1), 64-79.

Rajabioun, R. (2011). Cuckoo optimization algorithm. *Applied soft computing, 11*(8), 5508-5518. doi: 10.1016/j.asoc.2011.05.008

Ribas, I., Companys, R., & Tort-Martorell, X. (2011). An iterated greedy algorithm for the flowshop scheduling problem with blocking. *Omega 39,* 293–301. https://doi.org/10.1016/j.omega.2010.07.007

Riondato, M., & Vandin, F. (2014). *Finding the true frequent item sets.* In proceedings of the 2014 SIAM International Conference on Data Mining, 497-505. Society for Industrial and Applied Mathematics. doi: 10.1137/1.9781611973440.57

Rios, E., Ochi, L. S., Boeres, C., Coelho, V. N., Coelho, I. M., & Farias, R. (2018). Exploring parallel multi-GPU local search strategies in a metaheuristic framework. *Journal of Parallel and Distributed Computing, 111*, 39–55. https://doi.org/10.1016/j.jpdc.2017.06.011

Rogers, D. (1972). Random Search and Insect Population Models. *The Journal of Animal Ecology, 41*(2), 369. https://doi.org/10.2307/3474

Ruiz, P., Dorronsoro, B., Torre, J.C.D.L., & Burguillo, J.C. (2018). *Finding the most influential parameters of coalitions in a PSO-CO algorithm.* International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, June, Springer, Cham, pp. 284-296.

Sang, H.Y. (2010). *A discrete differential evolution algorithm for lot-streaming flow shop scheduling problems*, in: Proceedings - 2010 6th International Conference on Natural Computation, ICNC 2010. Elsevier, pp. 10–13. https://doi.org/10.1109/ICNC.2010.5583339

Sevkli, M., & Guner, A. R. (2006, September). *A continuous particle swarm optimization algorithm for uncapacitated facility location problem*. In International workshop on ant colony optimization and swarm intelligence, 316-323. Springer, Berlin, Heidelberg. doi: 10.1007/11839088_28

Shadravan, S., Naji, H. R., & Bardsiri, V. K. (2019). The Sailfish Optimizer: A novel nature-inspired metaheuristic algorithm for solving constrained engineering optimization problems. *Engineering Applications of Artificial Intelligence, 80,* 20–34. https://doi.org/10.1016/j.engappai.2019.01.001

Shi, H. (2006). *Solution to 0/1 knapsack problem based on improved ant colony algorithm.* In proceedings of IEEE International Conference on Information Acquisition, 1062-1066, Weihai, China. doi: 10.1109/ICIA.2006.305887

Shlesinger, M. F., Zaslavsky, G. M., & Frisch, U. (1995). Lévy flights and related topics in physics. adsabs.harvard.edu.

Single Mode Data Sets [WWW Document], n.d. URL http://www.om-db.wi.tum.de/psplib/getdata.cgi?mode=sm (accessed 05.03.22).

Storn, R., & Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization, 11(*4), 341-359. doi: 10.1023/A:1008202821328

Su, L., & Zhou, Y. (2016). *Tolerating Correlated Failures In Massively Parallel Stream Processing Engines.* In proceedings of IEEE conference In Data Engineering (ICDE), 517-528, Helsinki, Finland. doi: 10.1109/ICDE.2016.7498267

Sun, M. (2006). Solving the uncapacitated facility location problem using tabu search. *Computers & Operations Research*, *33*(9), 2563-2589. doi: 10.1016/j.cor.2005.07.014

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European journal of Operational research*, *47*(1), 65-74. https://doi.org/10.1016/0377-2217(90)90090-X

Taylor, R. (2016). Approximations of the densest k-subhypergraph and set union knapsack problems. *arXiv preprint arXiv:1610.04935*.

Tizhoosh, H.R. (2005). *Opposition-Based Learning: A New Scheme for Machine Intelligence.* Int. Conf. Comput. Intell. Model. Control Autom. Int. Conf. Intell. Agents, Web Technol. Internet Commer. 1, 695–701. https://doi.org/10.1109/CIMCA.2005.1631345

Tormos, P., & Lova, A. (2001). A Competitive Heuristic Solution Technique for Resource-Constrained Project Scheduling. *Annals of Operations Research 102,* 65–81. https://doi.org/10.1023/A:1010997814183

Tormos, P., & Lova, A. (2003). An efficient multi-pass heuristic for project scheduling with constrained resources. *International Journal of Production Research*, *41*(5), 1071-1086. https://doi.org/10.1080/0020754021000033904

Tsuya, K., Takaya, M., & Yamamura, A. (2017). Application of the firefly algorithm to the uncapacitated facility location problem. *Journal of Intelligent & Fuzzy Systems*, *32*(4), 3201-3208. doi: 10.3233/JIFS-169263

Valls, V., Ballestín, F., & Quintanilla, S. (2005). Justification and RCPSP: A technique that pays. *European Journal of Operational Research*, *165*(2), 375-386. https://doi.org/10.1016/j.ejor.2004.04.008

Valls, V., Ballestin, F., & Quintanilla, S. (2008). A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European journal of operational research*, *185*(2), 495-508. https://doi.org/10.1093/brain/awq328

Ventresca, M., & Tizhoosh, H.R. (2007). *Simulated Annealing With Opposite Neighbors.* Proc. 2007 IEEE Symp. Found. Comput. Intell. FOCI 2007 186–192. https://doi.org/10.1109/FOCI.2007.372167

Wang, C., Mu, D., Zhao, F., & Sutherland, J. W. (2015). A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup-delivery and time windows. *Computers and Industrial Engineering*, *83*, 111–122. https://doi.org/10.1016/j.cie.2015.02.005

Wang, D., Wu, C. H., Ip, A., Wang, D., & Yan, Y. (2008). *Parallel multi-population particle swarm optimization algorithm for the uncapacitated facility location problem using openMP*. In Proceedings of IEEE World Congress on Computational Intelligence, 1214-1218, Hong Kong, China. doi: 10.1109/CEC.2008.4630951

Wang, L., & Fang, C. (2012). A hybrid estimation of distribution algorithm for solving the resource-constrained project scheduling problem. *Expert Systems with Applications* 39, 2451–2460. https://doi.org/10.1016/j.eswa.2011.08.095

Wang, W. C., Xu, L., Chau, K. W., Zhao, Y., & Xu, D. M. (2021). An orthogonal opposition-based-learning Yin–Yang-pair optimization algorithm for engineering optimization. *Engineering with Computers*, 1-35. https://doi.org/10.1007/s00366-020-01248-9

Wang, Y., Jia, Z. hong, & Li, K. (2020). A multi-objective co-evolutionary algorithm of scheduling on parallel non-identical batch machines. *Expert Systems with Applications, 167*, 114145. https://doi.org/10.1016/j.eswa.2020.114145

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82. https://doi.org/10.1109/4235.585893

Yang, X.S. (2010). *A new metaheuristic bat-inspired algorithm.* In: J. González, D. Pelta,C. Cruz, G. Terrazas, N. Krasnogor (Eds.), Nature Inspired Cooperative Strategies for Optimization, Springer, Berlin/Heidelberg, 2010, pp. 65-74. doi: 10.1007/978-3-642-12538-6_6

Yang, X.S., 2010. *Firefly algorithm, Levy flights and global optimization.* Res. Dev. Intell. Syst. XXVI Inc. Appl. Innov. Intell. Syst. XVII 209–218. https://doi.org/10.1007/978-1-84882-983-1-15

Yang, X. S., & Deb, S. (2010). Engineering optimisation by cuckoo search. *International Journal*, *1*(4), 330-343. https://doi.org/10.1504/IJMMNO.2010.035430

Yang, X. S. (2010). *Nature-inspired metaheuristic algorithms*. Luniver press.

Yilmaz, H., & Yilmaz, M. (2020). A mathematical model and tabu search algorithm for multi-manned assembly line balancing problems with assignment restrictions. *Engineering Optimization*, *52*(5), 856–874. https://doi.org/10.1080/0305215X.2019.1618288

Young, H. D., Freedman, R. A., & Bhathal, R. (2010). *University physics: Australian edition*. Pearson Higher Education AU.

Yue, C. T., Price, K. V., Suganthan, P. N., Liang, J. J., Ali, M. Z., Qu, B. Y., Awad, N. H., & Biswas, P. P. (2019). *Problem Definitions and Evaluation Criteria for the CEC 2020 Special Session and Competition on Single Objective Bound Constrained Numerical Optimization.* Zhengzhou University, Zhengzhou China And Nanyang Technological University, Singapore, November. https://www.ntu.edu.sg/home/epnsugan/index_files/CEC2020/CEC2020-2.htm

Zapata, H., Perozo, N., Angulo, W., & Contreras, J. (2020). A Hybrid Swarm Algorithm for Collective Construction of 3D Structures Ontological Learning Schemes for Self-Organizing and Emergent Systems View project a Multiagent Architecture for Self-Organizing and Emergent Systems (MASOES) View project A Hybrid Swarm Algorithm for Collective Construction of 3D Structures. *International Journal of Artificial Intelligence, 18*(1), 1–18. https://www.researchgate.net/publication/340679872

Zhao, W., Wang, L., & Zhang, Z. (2019). Atom search optimization and its application to solve a hydrogeologic parameter estimation problem. *Knowledge-Based Systems, 163*, 283–304. https://doi.org/10.1016/j.knosys.2018.08.030

Zheng, X.L., & Wang, L. (2015). A multi-agent optimization algorithm for resource constrained project scheduling problem. *Expert System with Application*, *42*, 6039–6049. https://doi.org/10.1016/j.eswa.2015.04.009

Zhou, Y., Bao, Z., Luo, Q., & Zhang, S. (2017). A complex-valued encoding wind driven optimization for the 0-1 knapsack problem. *Applied Intelligence, 46*(3), 684-702. doi: 10.1007/s10489-016-0855-2

Zhou, Y., Li, L., & Ma, M. (2016). A complex-valued encoding bat algorithm for solving 0-1 knapsack problem. *Neural Processing Letters, 44*(2), 407-430. doi: 10.1007/s11063-015-9465-y

Ziarati, K., Akbari, R., & Zeighami, V. (2011). On the performance of bee algorithms for resource-constrained project scheduling problem. *Applied Soft Computing*, *11*(4), 3720-3733.

Zou, D., Gao, L., Li, S., & Wu, J. (2011). Solving 0-1 knapsack problem by a novel global harmony search algorithm. *Applied Soft Computing*, *11*(2), 1556-1564. doi: 10.1016/j.asoc.2010.07.019

APPENDICIES

Pseudo Codes of SSBMAs

```
PROCEDURE RS
Counter=0
WHILE Counter<convergence iteration limit
        Generate new solution randomly
        Calculate new solution' fitness
              IF new solution fitness<best fitness
                      BestSolution=NewSolution;
                      Counter=0;
              ELSE
                      Counter=Counter+1
              END IF
END WHILE
END PROCEDURE
```

Figure A1. Random Search algorithm

```
PROCEDURE TA
Counter=0
WHILE Counter<convergence iteration limit
        Generate new solution via neighborhood mechanism
        Calculate the new solution' fitness
        Gap= (new solution fitness-solution fitness)
              IF Gap<Threshold // Threshold represents acceptance limit of bad solutions
                      Solution=NewSolution;

              END IF
              IF new solution fitness<best fitness
                          BestSolution=NewSolution;
                   Counter=0;
              ELSE
                   Counter=Counter+1
              END IF
Threshold=Threshold*DecreaseRate;
END WHILE
END PROCEDURE
```

Figure A2. Threshold Accepting algorithm

```
PROCEDURE GD
Counter=0
WHILE Counter<convergence iteration limit
        Generate new solution by neighborhood mechanism
        Calculate the new solution fitness
              IF new solution fitness<best fitness
                  BestSolution=NewSolution;
                  Counter=0
              ELSE IF new solution fitness <Level // Level represents acceptance limit of bad solutions
                  Solution=BestSolution;
                  Counter=Counter+1
              END IF
Level= Level*DecreaseRate;
END WHILE
END PROCEDURE
```

Figure A3. Great Deluge algorithm

```
PROCEDURE GS
Counter=0
WHILE Counter<convergence iteration limit
        Generate new solution by neighborhood mechanism
        Calculate the new solution fitness
              IF new solution fitness<solution fitness
                  Solution=NewSolution;
              END IF
              IF solution fitness<best fitness
                  Solution=BestSolution;
                  Counter=0
              ELSE
                  Counter=Counter+1
              END IF
END WHILE
END PROCEDURE
```

Figure A4. Greedy Search algorithm

```
PROCEDURE SA
Counter=0
WHILE Counter<convergence iteration limit
        Generate new solution by neighborhood mechanism
        Calculate new solution fitness
              IF new solution fitness<solution fitness
                   Solution=NewSolution;
              ELSE
              Delta= (new solution fitness – solution fitness)/solution fitness
              P=e^(-Delta/T) // T represents Temperature
                    IF Rand () <=P // P is a random number between 0-1.
                    Solution=NewSolution;
                    END IF
              END IF
              IF solution fitness<best fitness
                  BestSolution=Solution;
                  Counter=0
              ELSE
                  Counter=Counter+1
              END IF
         UPDATE T
END WHILE
END PROCEDURE
```

Figure A5. Simulated Annealing algorithm

Move Mechanisms

i. **Single Swap:** Single swap operator chooses two positions in the current solution vector and exchanges the elements of these two positions. The all other positions preserve their elements. An illustration of this operator is depicted in Figure 14. Assume that the randomly chosen positions are 3 and 7.

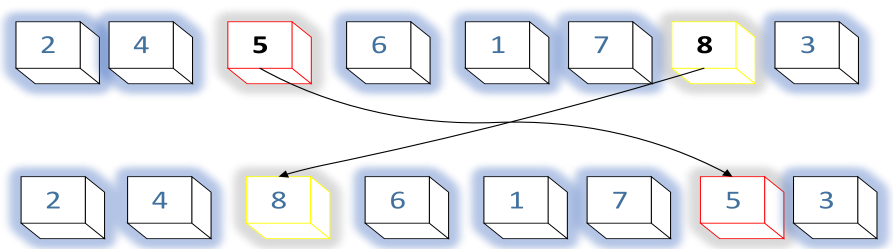


Figure A.6. Single Swap Move

ii. **Two-Block Swap:** Two-block swap operator chooses two different blocks with a length of two in the current solution vector and exchanges the elements of these two blocks. An illustration of this operator is depicted in Figure 15. Assume that the randomly chosen positions are 2 and 5 respectively.

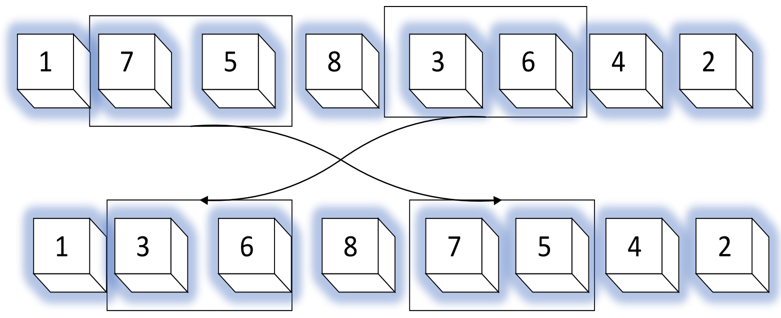


Figure A.7 Two-Block Swap Move

iii. **Three-Block Swap:** Three-block swap operator chooses two different blocks with a length of three in the current solution vector and exchanges the elements of these two blocks. An illustration of this operator is depicted in Figure 16. Assume that the randomly chosen positions are 2 and 6 respectively.

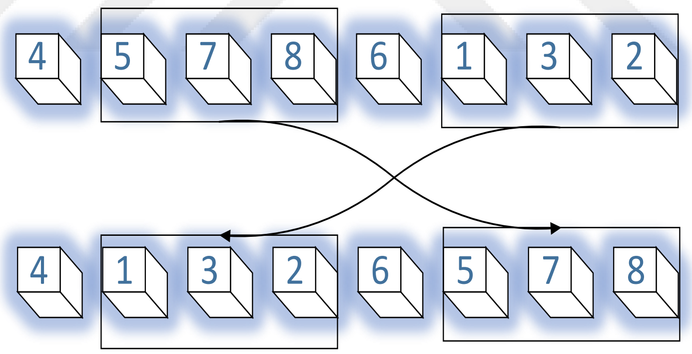


Figure A.8 Three-Block Swap Move

iv. **Inversion:** Inversion operator chooses a block of elements in the current solution vector with a length of step size and reverses the order of the elements in this block. An illustration of this operator is depicted in Figure 17. Assume that the randomly chosen position is 4 and the step length is 3.

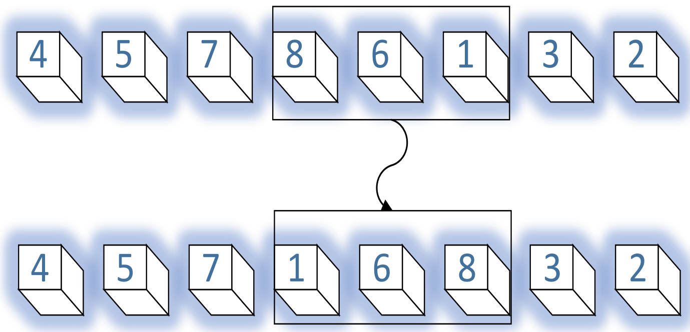


Figure A.9 Three-Block Swap Move

v. **Ejection Chain:** Ejection chain operator is the combination of the three different operators. It includes single swap, single shifting and double shifting operators respectively. An illustration of this operator is depicted in Figure 18. Assume that the randomly chosen positions are 3 and 7 for the single swap, 2 and 5 for single shifting and (4, 6) and (3, 5) for double shifting.

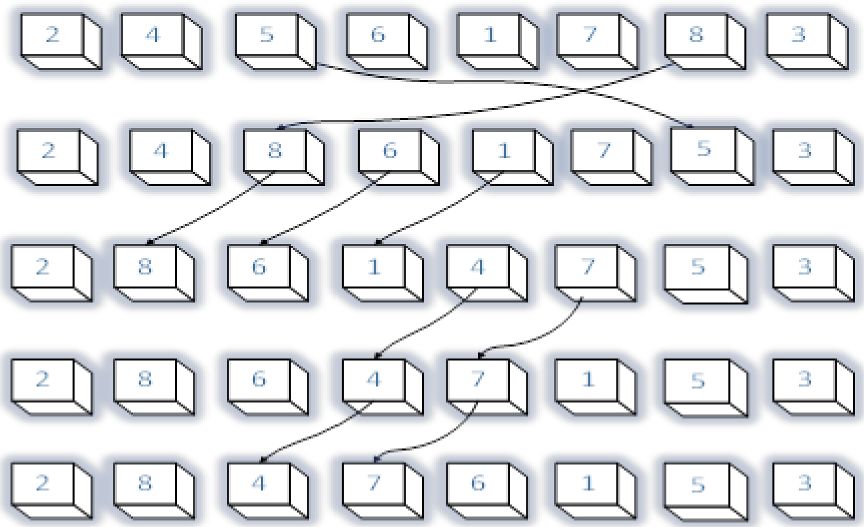


Figure A.10 Ejection Chain Move

vi. **Single Insertion:** Single Insertion operator chooses a position from the current solution vector and inserts the element of this position to a randomly selected position. An illustration of this operator is depicted in Figure 19. Assume that randomly chosen positions are 3 and 7.

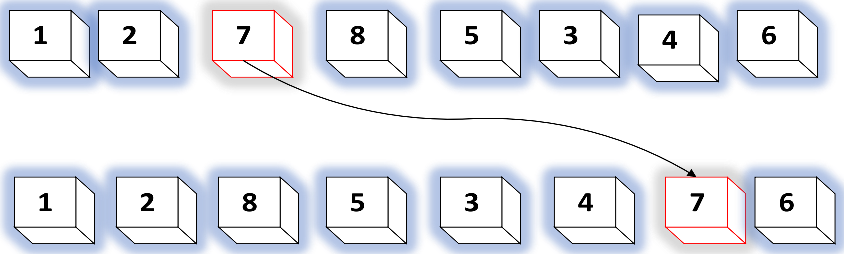


Figure A.11 Single Insertion Move

vii. **Block Insertion:** Block insertion operator chooses a block with a length of step size from the current solution vector and inserts the element of this block to a randomly selected position. An illustration of this operator is depicted in Figure 20. Assume that step length is 2 and randomly chosen positions are 2 and 6.

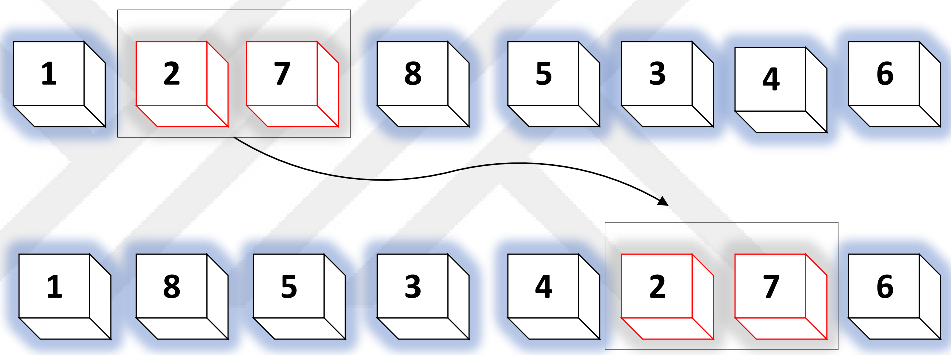


Figure A.12 Block Insertion Move

Step Size Calculation

$$sl_{ik} = mod(abs(sl_{i(k-1)} + Lévy), length) \tag{A.1}$$

The *Lévy* term in Eq. 1 is obtained as in Eq. 2, where an output (sl_{k+1}) is an input (sl_k) for the next number generation.

$$\begin{Bmatrix} \omega = rnd(\,) \times 2\pi \\ \rho = rnd(\,)^{\left(\frac{-1}{\eta}\right)} \\ sl_{k+1} = sl_k + \rho \times \cos\omega \end{Bmatrix} \tag{A.2}$$

Assume that

$sl_k = 1$, η= 1.5, random number 1 = 0.32; random number 2= 0.78.

$\omega = rnd(\,) \times 2\pi$

$\omega = 0.32 \; \times 2\pi \; = 2.0106$

$\rho = rnd(\,)^{\left(\frac{-1}{\eta}\right)}$

$\rho = (0.78)^{\frac{-1}{1.5}}$ =1.1801

$sl_{k+1} = sl_k + \rho \times \cos\omega$

$sl_{k+1} = 1 + 1.1801 \; x \cos(2.0106)$ =0.4976