New Approaches in Parallel Algorithm Portfolios for Metaheuristic Optimization

A Dissertation

submitted to the designated
by the General Assembly of Special Composition
of the Department of Computer Science and Engineering
Examination Committee

by

Dimitrios Souravlias

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

University of Ioannina

June 2017

Advisory Committee

- **Konstantinos E. Parsopoulos**, Associate Professor, Department of Computer Science & Engineering, University of Ioannina, Greece
- **Enrique Alba**, Professor, Department of Computer Science & Programming Languages, University of Málaga, Spain
- **Konstantina Skouri**, Associate Professor, Department of Mathematics, University of Ioannina, Greece

Examination Committee

- **Konstantinos E. Parsopoulos**, Associate Professor, Department of Computer Science & Engineering, University of Ioannina, Greece
- **Enrique Alba**, Professor, Department of Computer Science & Programming Languages, University of Málaga, Spain
- **Konstantina Skouri**, Associate Professor, Department of Mathematics, University of Ioannina, Greece
- **Isaac Lagaris**, Professor, Department of Computer Science & Engineering, University of Ioannina, Greece
- **Ilias Kotsireas**, Professor, Department of Physics & Computer Science, Wilfrid Laurier University, Canada
- **Vassilios Dimakopoulos**, Associate Professor, Department of Computer Science & Engineering, University of Ioannina, Greece
- **Dimitrios Papageorgiou**, Associate Professor, Department of Materials Science & Engineering, University of Ioannina, Greece

Dedication

To my family

Acknowledgements

First and foremost I would like to express my deepest gratitude to my advisor, Professor Konstantinos E. Parsopoulos, for his valuable guidance and continuous support from the beginning of my PhD studies. Our excellent collaboration has given me the opportunity to acquire important knowledge and skills that have helped me in my research endeavor. Our long discussions and his constant encouragement have greatly contributed to my scientific and personal development. I would feel grateful if I have managed to acquire even the minimum of the professionalism and academic ethics that distinguish him.

I would like to express my sincere thanks to the members of the Advisory Committee Professor Enrique Alba and Professor Konstantina Skouri. Professor Enrique Alba and his research team at the University of Málaga have been exceptional collaborators on solving difficult optimization problems met in the environment of smart cities. His advice and constructive comments have been of great importance. Professor Konstantina Skouri and I have collaborated on modeling and solving operations research problems. Her constructive guidance has been fundamental in my work in the field of operations research. Her advice has always been invaluable.

I am also very grateful to Professor Ilias Kotsireas with whom I have extensively collaborated on combinatorial optimization problems. Professor Ilias Kotsireas has always supported, motivated me, and assisted me to all my research needs. I would like to thank Professor Gerasimos Meletiou for our excellent collaboration on addressing challenging optimization problems in cryptography.

I would also like to thank the members of the Examination Committee, Professor Isaac Lagaris, Professor Vassilios Dimakopoulos, and Professor Dimitrios Papageorgiou for their insightful comments and critical view on my work.

Many thanks to my friends Dimitris Kitsakis, Vasilis Bourgos, Katerina Pandremmenou, Nikos Tziortziotis, Kostas Tziortziotis, Marina Drosou, Grigoris Tzortzis and

Kostas Stefanidis for their unconditional support during my postgraduate studies. Also, I would like to thank my friend Vasilis Tatsis for our long discussions and the pleasant atmosphere at the lab.

I would like to deeply thank my parents Nikos and Katerina and my brother Vasilis for their support. Their advice, encouragement, and endless love have always helped me move forward.

Finally, a special thanks goes to my life-partner Anastasia, who has always been by my side and patiently supported me at every step. Her continuous presence in my life has been a driving force within me.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

AC	Autocorrelation
AD	AvgDev Criterion
AES	Advanced Encryption Standard
AP	Algorithm Portfolio
ASY	Asychronous Particle Swarm Optimization
CHC Genetic	Cross-generational Elitist Selection, Heterogeneous recombination, and Cataclysmic Mutation
CWM	Circulant Weighing Matrix
DES	Data Encryption Standard
DBC	Directed Bee Colony
DE	Differential Evolution
DWA	Dynamic Weighted Aggregation
eDE	Enhanced Differential Evolution
ELS	Economic Lot Sizing
ELSR	Economic Lot Sizing with Remanufacturing
GA	Genetic Algorithm
G-CMA-ES	Restart Covariant Matrix Evolutionary Strategy
EvoPROpt	Evolutionary Path Relinking
HL	Humanitarian Logistics
ILS	Iterated Local Search
ITHS	Intelligent Tuned Harmony Search
L	Linear
LB	LocalBest Criterion
LS	Local Search
LWA	Linear Weighted Aggregation
NL	Nonlinear

OCBA	Optimal Computing Budget Allocation
PAF	Periodic Autocorrelation Function
PFA	Pareto Front Approach
PR	Path Relinking
PSO	Particle Swarm Optimization
PSO-DLI	Particle Swarm Optimization with Deliberate Loss of Information
PSO-NBA	Particle Swarm Optimization with Neighborhood-based Budget Allocation
ROI	Return on Investment
SA	Simulated Annealing
S-box	Substitution box
SB	SumBest Criterion
SOBA	Single-objective Budget Allocation
SPSO2011	Standard Particle Swarm Optimization 2011
SPV	Smallest Position Value
MOBA	Multi-objective Budget Allocation
TL	Tabu List
TP	Test Problem
TS	Tabu Search
VNS	Variable Neighborhood Search
WHT	Walsh-Hadamard Transform

Abstract

Dimitrios Souravlias, Ph.D., Department of Computer Science and Engineering, University of Ioannina, Greece, June 2017.
New Approaches in Parallel Algorithm Portfolios for Metaheuristic Optimization.
Advisor: Konstantinos E. Parsopoulos, Associate Professor.

Optimization problems are ubiquitous in science and engineering. They emerge in various types and forms in almost all aspects of decision-making. The abundance and diversity of optimization problems have offered ample ground for the development of innovative solution methodologies. Numerous optimization methods have been proposed in the past decades, leading to an ongoing expansion of the available algorithmic artillery. Nowadays, there is a rich variety of optimization algorithms with diverse effectiveness and efficiency characteristics. However, both theoretical and experimental evidence suggest that the existence of a universal optimization algorithm capable of tackling all optimization problems equally well is highly improbable. This conjecture has offered strong motivation for the ongoing development of optimization algorithms with diverse characteristics that match specific properties of the confronted problems.

Typically, each optimization procedure requires a number of decisions taken by the practitioner. Perhaps the most important one is the selection of the applied optimization algorithm. This is a rather complex task and usually requires deep knowledge of the problem and experience from the practitioner's side. Whenever the available information on the problem is limited, preliminary experimentation is needed for the selection of the most promising algorithm among a group of candidates through a trial-and-error procedure. This phase is error-prone as well as time-consuming. In fact, it may require more time than the solution of the problem itself due to the computational intensity of the involved statistical methodologies. Moreover, it does not take directly into consideration the online dynamic of each algorithm, i.e., its performance

fluctuations during its execution. Indeed, some algorithms are exploration-intensive while other are exploitation-intensive. While the first are more useful at the beginning of the optimization procedure in order to detect promising regions of the search space, the latter are preferable at the end where more fine-grained search around the best solutions is desirable. An arbitrary selection of a single algorithm from either category seems to explicitly promote local or global search, thereby resulting in questionable performance.

Algorithm Portfolios were proposed as algorithmic models that harness a number of algorithms in a joint algorithmic scheme, aiming at the alleviation of the aforementioned deficiencies. Standard algorithm portfolios assume a number of algorithms that are executed either serially on a single processing unit or in parallel whenever multiple processing units are available. In the first case, the constituent algorithms of the portfolio are interchangeably executed, consuming a fixed portion of the available computational budget (function evaluations or time) in a round-robin manner. In the second case, the processing units and the computational budget are shared among the algorithms according to a prescribed plan, offering obvious advantages in terms of time-efficiency. The resources allocation plan of the portfolio is usually determined prior to its application, based on preliminary experiments or historical performance data of the algorithms. Thus, the problem of algorithm selection is replaced with the problem of appropriate resources allocation in the portfolio. Although the simplistic approach of assigning equal proportions of the available resources may provide adequate solutions in various problems, it does not take full advantage of the portfolio and its constituent algorithms. To this end, sophisticated resources allocation schemes can offer significant performance enhancement.

The main goals of the present thesis are the justification for the use of metaheuristic algorithm portfolios in demanding optimization problems of various types, and the development of new parallel algorithm portfolio models with adaptive resources allocation. In the first part of the thesis (Chapters 1- 3) motivation for the use of algorithm portfolios is provided. The usefulness of appropriate computational budget allocation in contemporary metaheuristics is identified, and simplistic parallel algorithm portfolios are demonstrated on challenging problems such as the design of cryptographically strong S-boxes, and the traffic light scheduling in smart cities environments.

In the second part of the thesis, two new parallel algorithm portfolio models

with sophisticated resources allocation mechanisms are proposed. The first model is an algorithm portfolio with trading-based budget allocation, which introduces a market-based environment where the constituent algorithms of the portfolio can trade their solutions for additional running time. The model is highly autonomous and allows the algorithms to individually interact whenever specific conditions (e.g., search stagnation) are met. It is demonstrated on three challenging problems, namely the detection of circulant weighing matrices in combinatorics, the lot-sizing planning in production systems with returns and remanufacturing, and the transportation of commodities in humanitarian logistics.

The second proposed model is a forecasting-based parallel algorithm portfolio where time series forecasting is employed to predict the performance of the portfolio's constituent algorithms. The predictions are used to assign computational resources to the constituent algorithms accordingly. The model is demonstrated again on the detection of circulant weighing matrices in combinatorics, offering valuable insight regarding its parameterization.

Εκτεταμενη Περιληψη

Δημήτριος Σουραβλιάς, Δ.Δ., Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Ιούνιος 2017.
Νέες Προσεγγίσεις σε Παράλληλα Χαρτοφυλάκια Αλγορίθμων για Μεταευρετική Βελτιστοποίηση.
Επιβλέπων: Κωνσταντίνος Ε. Παρσόπουλος, Αναπληρωτής Καθηγητής.

Τα προβλήματα βελτιστοποίησης είναι πανταχού παρόντα στην επιστήμη και στη μηχανική. Εμφανίζονται σε διάφορους τύπους και μορφές σε όλες σχεδόν τις διαδικασίες λήψης αποφάσεων. Η αφθονία και η ποικιλομορφία των προβλημάτων βελτιστοποίησης έχουν δώσει πρόσφορο έδαφος για την ανάπτυξη καινοτόμων μεθόδων και τεχνικών επίλυσης. Διάφορες μέθοδοι βελτιστοποίησης έχουν προταθεί τις τελευταίες δεκαετίες, καταγράφοντας διαρκή αύξηση των διαθέσιμων αλγορίθμων. Αυτό έχει ως αποτέλεσμα την ύπαρξη πλούσιας ποικιλίας αλγορίθμων με ετερογενή χαρακτηριστικά ως προς την αποτελεσματικότητα και την αποδοτικότητά τους. Επιπλέον, τόσο θεωρητικές όσο και πειραματικές μελέτες καταλήγουν στο συμπέρασμα ότι η ύπαρξη ενός καθολικού αλγορίθμου βελτιστοποίησης ικανού να αντιμετωπίσει εξίσου καλά όλα τα δυνατά προβλήματα βελτιστοποίησης είναι απίθανη. Αυτό έχει προσφέρει ισχυρά κίνητρα για τη συνεχή ανάπτυξη αλγορίθμων βελτιστοποίησης με ποικίλα χαρακτηριστικά, τα οποία εκμεταλλεύονται συγκεκριμένες ιδιότητες των εκάστοτε προβλημάτων.

Γενικά, κάθε διαδικασία βελτιστοποίησης απαιτεί τη λήψη μιας σειράς αποφάσεων. Ίσως η πιο σημαντική απόφαση είναι η επιλογή του αλγορίθμου βελτιστοποίησης που θα χρησιμοποιηθεί για επίλυση. Πρόκειται για μια δύσκολη επιλογή που συνήθως απαιτεί βαθιά γνώση του προβλήματος αλλά και εμπειρία. Σε περίπτωση που οι διαθέσιμες πληροφορίες για το πρόβλημα είναι περιορισμένες, συνήθως απαιτείται προκαταρκτική πειραματική μελέτη για την επιλογή του καταλληλότερου αλγορίθμου από ένα σύνολο υποψηφίων, μέσω μιας διαδικασίας δοκιμής-σφάλματος.

Αυτή η διαδικασία είναι επιρρεπής σε λάθη καθώς και χρονοβόρα. Στην πράξη μπορεί να απαιτεί περισσότερο χρόνο ακόμη κι από τη διαδικασία επίλυσης του ίδιου του προβλήματος, λόγω του υπολογιστικού φόρτου των χρησιμοποιούμενων στατιστικών μεθόδων. Επιπλέον, δε λαμβάνει άμεσα υπόψη την δυναμική του κάθε αλγορίθμου σε πραγματικό χρόνο, δηλαδή τις διακυμάνσεις της απόδοσης κατά την εκτέλεσή του. Για παράδειγμα, ορισμένοι αλγόριθμοι είναι καταλληλότεροι για ευρύτερη εξερεύνηση του χώρου αναζήτησης, ενώ άλλοι λειτουργούν πιο αποτελεσματικά για την τοπική βελτίωση των καλύτερων λύσεων. Ενώ οι πρώτοι είναι συνήθως πιο χρήσιμοι στην αρχή της βελτιστοποίησης για την ανίχνευση καλών περιοχών του χώρου αναζήτησης, οι τελευταίοι είναι πιο χρήσιμοι στο τέλος όπου είναι επιθυμητή η πιο λεπτομερής αναζήτηση γύρω από τις καλύτερες λύσεις. Η αυθαίρετη επιλογή ενός και μόνο αλγορίθμου που ανήκει σε κάποια από τις δύο αυτές κατηγορίες μπορεί να μην έχει την αναμενόμενη απόδοση στο πρόβλημα που θα εφαρμοστεί.

Για τους παραπάνω λόγους, *χαρτοφυλάκια αλγορίθμων* αποτελούμενα από μεταευρετικούς αλγορίθμους έχουν προταθεί στη βιβλιογραφία για την αντιμετώπιση δύσκολων προβλημάτων βελτιστοποίησης. Τα χαρτοφυλάκια αλγορίθμων αποτελούν σχήματα που ενσωματώνουν διαφορετικούς αλγορίθμους ή διαφορετικές εκδοχές του ίδιου αλγορίθμου, οι οποίες εκτελούνται σειριακά (σε μία μονάδα επεξεργασίας) ή παράλληλα (όταν περισσότερες μονάδες επεξεργασίας είναι διαθέσιμες). Στην πρώτη περίπτωση, οι αλγόριθμοι του χαρτοφυλακίου εναλλάσσουν την εκτέλεσή τους, καταναλώνοντας ο καθένας εκ περιτροπής ένα κλάσμα των διαθέσιμων υπολογιστικών πόρων (συναρτησιακοί υπολογισμοί ή χρόνος). Στη δεύτερη περίπτωση, οι μονάδες επεξεργασίας και οι υπολογιστικοί πόροι μοιράζονται μεταξύ των αλγορίθμων σύμφωνα με ένα προκαθορισμένο πλάνο, προσφέροντας προφανή πλεονεκτήματα ως προς το χρόνο εκτέλεσης. Το πλάνο κατανομής πόρων καθορίζεται συνήθως πριν από την εφαρμογή του χαρτοφυλακίου, διαμέσου προκαταρκτικής πειραματικής μελέτης ή ιστορικών δεδομένων απόδοσης των αλγορίθμων. Έτσι, το πρόβλημα επιλογής αλγορίθμου αντικαθίσταται από το πρόβλημα κατάλληλης κατανομής υπολογιστικών πόρων στο χαρτοφυλάκιο. Αν και η απλοϊκή προσέγγιση της ισόποσης κατανομής πόρων στους αλγορίθμους μπορεί να προσφέρει επαρκείς λύσεις σε διάφορα προβλήματα, δεν εκμεταλλεύεται πλήρως το χαρτοφυλάκιο και τους αλγόριθμους που το αποτελούν. Η πραγματικού χρόνου κατανομή υπολογιστικών πόρων στους αλγορίθμους μπορεί να οδηγήσει σε σημαντική βελτίωση τόσο της αποδοτικότητας όσο και της αποτελεσματικότητας.

Οι κύριοι στόχοι της παρούσας διατριβής είναι η αιτιολόγηση της χρήσης χαρτοφυλακίων μεταευρετικών αλγορίθμων σε προβλήματα βελτιστοποίησης διαφόρων τύπων και η ανάπτυξη νέων μοντέλων παράλληλων χαρτοφυλακίων αλγορίθμων με δυναμικά προσαρμοζόμενα σχέδια κατανομής υπολογιστικών πόρων. Στο πρώτο μέρος της διατριβής δίνονται κίνητρα για τη χρήση χαρτοφυλακίων αλγορίθμων (Κεφάλαια 1-3) και διερευνάται η χρησιμότητα των μηχανισμών κατανομής υπολογιστικών πόρων σε μεταευρετικούς αλγορίθμους. Στη συνέχεια προτείνονται παράλληλα χαρτοφυλάκια αλγορίθμων για απαιτητικά προβλήματα όπως ο σχεδιασμός κρυπτογραφικά ισχυρών S-boxes καθώς και ο χρονοπρογραμματισμός φωτεινών σηματοδοτών σε περιβάλλοντα έξυπνων πόλεων.

Στο δεύτερο μέρος της εργασίας, προτείνονται δύο νέα παράλληλα μοντέλα χαρτοφυλακίων αλγορίθμων, τα οποία υλοποιούν νέους μηχανισμούς κατανομής υπολογιστικών πόρων. Το πρώτο μοντέλο είναι ένα *χαρτοφυλάκιο αλγορίθμων βασισμένο σε συναλλαγές*, το οποίο χρησιμοποιεί ένα νέο μηχανισμό κατανομής πόρων, σύμφωνα με τον οποίο οι αλγόριθμοί του μπορούν να πωλούν λύσεις κερδίζοντας επιπλέον χρόνο εκτέλεσης. Το μοντέλο είναι αυτόνομο και επιτρέπει στους αλγορίθμους να αλληλεπιδρούν όταν πληρούνται συγκεκριμένες συνθήκες (για παράδειγμα στασιμότητα αναζήτησης). Το μοντέλο εφαρμόζεται σε τρία απαιτητικά προβλήματα όπως η ανίχνευση κυκλικών πινάκων στάθμισης, ο προσδιορισμός μεγέθους παρτίδας σε συστήματα παραγωγής με επιστροφές και ανακατασκευή προϊόντων, καθώς και η μεταφορά εμπορευμάτων σε ανθρωπιστικές εφοδιαστικές αλυσίδες. Το δεύτερο προτεινόμενο μοντέλο είναι ένα *χαρτοφυλάκιο αλγορίθμων βασισμένο σε προβλέψεις*, το οποίο χρησιμοποιεί τεχνικές πρόβλεψης χρονοσειρών για την πρόβλεψη της απόδοσης των αλγορίθμων που το αποτελούν. Οι προβλέψεις χρησιμοποιούνται για τον κατάλληλο διαμοιρασμό των διαθέσιμων υπολογιστικών πόρων στους αλγορίθμους του χαρτοφυλακίου. Το μοντέλο εφαρμόζεται και πάλι στην ανίχνευση κυκλικών πινάκων στάθμισης, προσφέροντας πολύτιμα συμπεράσματα σχετικά με την παραμετροποίησή του.

Chapter 1

Introduction

1.1 Overview

Optimization problems are met in every aspect of science and engineering. They emerge in a variety of types and forms in the majority of decision-making applications. Undoubtedly, the plethora of optimization problems has increased the demand for innovative solution methodologies. A high number of optimization methods [2] with diverse characteristics has appeared over the last decades to match the special properties of the existing problems and address their complexities. Their primary target has been the effectiveness in terms of solution quality and the efficiency in terms of computational resources. Despite the multitude of available optimization algorithms, strong theoretical results such as the No Free Lunch theorem [3] suggest that there is no universal algorithm that can tackle all problems equally well. Nevertheless, experimental evidence suggests that specific algorithmic instances can be particularly effective and efficient in specific optimization problems [4, 5]. Thus, the ability to identify the most appropriate algorithm eventually determines the boundary between success and failure when challenging optimization problems are confronted.

A central problem in applied optimization lies in the selection of the most appropriate algorithm and/or its parameter setting for tackling a given problem. The so called *a*lgorithm selection problem has concentrated the attention of the research community for many decades [6,7]. Algorithm selection methods usually operate offline [5] and involve decisions of high complexity. The difficulties can be mitigated under deep knowledge of the problem at hand as well as experience from the practitioner's side. When limited a priori knowledge of the problem exists, preliminary experimentation is required for the selection of a suitable algorithm among a set of candidates. In this case, a trial-and-error phase takes place involving the execution of a number of algorithms and performance comparisons through statistical methodologies [5]. This is habitually an error-prone and time-consuming process as it may requires more time than the solution of the problem itself. This is due to the computational resources spent by the used algorithmic approaches as required by the statistical methodologies.

Even if a single algorithm is selected, its performance exhibits fluctuations during the optimization procedure. While an algorithm may prove to be efficient at early stages of the optimization procedure, it may exhibit declining performance after a critical number of iterations. In fact, there are exploration-intensive algorithms that generally perform better at the beginning of the optimization process whereas exploitation-intensive ones are more useful at the end [8]. In general, a single algorithm cannot satisfy both requirements equally well. Therefore, selecting algorithms from both categories rather than a single one can be a wise choice to combine their diverse characteristics and complementary properties.

In this framework, *Algorithm Portfolios* (APs) [9,10] have been proposed as general models for building algorithmic schemes that alleviate deficiencies originating from the selection of a single algorithm for tackling a specific optimization problem. Specifically, APs define schemes that harness multiple individual algorithms, which share the available computational resources. APs can be either homogeneous [11] (consisting of instances of the same algorithm) or heterogeneous [12,13] (consisting of different algorithms). If a single processing unit is used, the AP's constituent algorithms are interchangeably executed according to a resource-assignment schedule. When many processing units are available, the algorithms run in parallel, each one occupying one processing unit [10]. Standard algorithm portfolios assume no interaction among their constituent algorithms [14]. However, recent studies with interactive

models suggested that performance benefits can be achieved [4].

The distribution of the available computational resources among the portfolio's constituents algorithms is crucial for its performance [12,13]. The choice of the appropriate resources allocation plan is usually an offline process, namely it is conducted prior to the execution of the portfolio. When historical performance data of the algorithms are available, they can be used to distribute the available resources among the algorithms, accordingly. Alternatively, preliminary experiments can be conducted to determine the fraction of computational resources allocated to each constituent algorithm. However, the assignment of prespecified portions of computational resources may be inefficient, since it does not take into account the online dynamic of each algorithm. In such cases, online techniques for the dynamic allocation of resources during the portfolio's run can be beneficial especially when addressing challenging problems.

In order to alleviate the problem of algorithm selection and to exploit the diverse performance dynamic of different algorithms, metaheuristics have been frequently used within algorithm portfolio frameworks [4, 11–13, 15, 16]. Metaheuristics can be especially useful to tackle optimization problems where good (sub-)optimal solutions are needed in reasonable time. They are distinguished into two groups, namely population-based and trajectory-based (or local-based) algorithms. Population-based algorithms use populations of individuals that explore the search space iteratively. Each individual stores a candidate solution of the problem under consideration. On the other hand, trajectory-based algorithms employ a single search point that is modified iteratively during the optimization process.

1.2 Thesis contribution

In the first part of the dissertation, motivation for the use of algorithm portfolios is provided. First, the usefulness of an appropriate computational budget allocation technique to a well studied metaheuristic algorithm, namely the Particle Swarm Optimization algorithm (PSO) [17,18] is identified. The standard PSO algorithm allocates the total available budget of function evaluations equally and concurrently among the particles of the swarm. In contrast to the plain PSO algorithm a new variant of PSO is proposed in Chapter 2 where each particle is dynamically assigned different computational budget based on the quality of its neighborhood. The main goal is

to favor particles of high-quality neighborhoods by asynchronously equipping them with additional function evaluations. For this purpose, quality criteria are defined to assess each neighborhood in terms of solutions quality and diversity. Established stochastic techniques are employed for the final selection among the particles. Different variants are proposed by combining various quality criteria in a single-objective or multi-objective manner. The proposed approach is assessed on widely used test suites as well as on a set of real-world problems. Experimental evidence reveals the effectiveness of the proposed approach and its competitiveness against other PSO-based variants as well as different established algorithms. The following contributions are achieved:

- A PSO variant that employs neighborhood-based ranking is proposed.
- Two neighborhood quality criteria as well as one neighborhood diversity criterion are proposed.
- Two alternative selection probability schemes are proposed; a linear schema and a nonlinear one.
- The new PSO variant is evaluated on a widely used benchmark of several objective functions.
- The proposed approach is compared against other PSO-based variants as well as different established metaheuristics.

In Chapter 3, two essential parallel algorithm portfolio models are proposed. The first model is generic and can be used by any optimization algorithm. In the present thesis, it has been tested for the design of cryptographically strong S-boxes [19, 20]. The second model is especially designed for population-based algorithms. Its use is beneficial in cases where the function evaluations are expensive in terms of computational time. This model is assessed on the traffic light scheduling problem [21] that arises in smart cities environments. The following contributions are achieved:

- Two simplistic parallel algorithm portfolio models are proposed; a generic and a population-based one.
- The generic algorithm portfolio model is applied on the design of cryptographically strong S-boxes.

- The population-based algorithm portfolio model is applied on the traffic light scheduling problem.
- Thorough experimentation is conducted under different parameter settings, offering interesting conclusions.

In Chapter 4, a trading-based parallel algorithm portfolio is proposed. The portfolio exploits a novel trading-based budget allocation mechanism that distributes the available computational resources among its constituent algorithms. Specifically, the portfolio introduces a market-based environment where each constituent algorithm can buy solutions from the rest or sell its own ones for additional running time. Thus, best-performing algorithms gain more running time than the rest, as they sell solutions more frequently. The resource allocation occurs indirectly through the trading of solutions and dynamically during the optimization process. The portfolio is highly autonomous and allows the algorithms to individually interact whenever specific conditions (e.g., search stagnation) are identified. It is applied on three challenging problems, namely the detection of circulant weighing matrices in combinatorics [22], the lot-sizing planning in production environments with returns and remanufacturing [23], and the transportation of commodities in humanitarian logistics [24]. Thorough experimentation under different parameter settings offers insightful conclusions and demonstrates the potential of the portfolio. The following contributions are achieved:

- A parallel algorithm portfolio that employs a sophisticated trading-based mechanism is proposed.
- The mechanism allocates the available computational resources indirectly through the trading of solutions and dynamically during the optimization process.
- It is applied on three challenging problems, namely the detection of circulant weighing matrices in combinatorics, the lot-sizing planning in production environments with returns and remanufacturing, and the transportation of commodities in humanitarian logistics.
- Thorough experimentation under different parameter settings demonstrates the potential of the portfolio on the considered problems.

In Chapter 5, a forecasting-based parallel algorithm portfolio is proposed. The portfolio employs a novel forecasting-based mechanism adopted from prevalent time series forecasting techniques. Specifically, the mechanism is used to predict the forthcoming performance of the portfolio's constituent algorithms based on current and past performance data. The predictions are then used to assign computational resources by favoring the most promising constituent algorithms. The portfolio is applied on the detection of circulant weighing matrices in combinatorics. Extensive experimentation under different parameterizations offers interesting results that reveal the combined algorithmic power of the portfolio. The following contributions are achieved:

- A parallel algorithm portfolio that employs a novel forecasting-based mechanism is proposed.
- Three prominent forecasting time series techniques are employed by the attained mechanism.
- The portfolio predicts the performance of its constituent algorithms and allocates the available computational resources accordingly.
- It is applied on the detection of circulant weighing matrices in combinatorics, revealing its efficiency and effectiveness.

1.3 Thesis Layout

The thesis is organized as follows: Chapter 2 provides the necessary background in metaheuristics and presents a novel budget allocation mechanism that is incorporated into the PSO algorithm. Chapter 3 presents two basic parallel algorithm portfolio models along with applications on the detection of cryptographically strong S-boxes and on the traffic light scheduling problem. Chapter 4 presents a new trading-based algorithm portfolio, which is demonstrated on three challenging problems, namely the detection of circulant weighing matrices in combinatorics, the lot-sizing planning in production environments with returns and remanufacturing, and the transportation of commodities in humanitarian logistics. In Chapter 5 a new forecasting-based algorithm portfolio is exposed, which is applied on the detection of circulant weighing matrices in combinatorics. Finally, Chapter 6 concludes the dissertation and outlines directions for future work.

Chapter 2

Metaheuristic Optimization

2.1 Intoduction

In this chapter, the employed metaheuristic algorithms are outlined. Without loss of generality, the continuous bound constrained minimization problem is considered,

$$\min_{\mathbf{x}\in\mathcal{X}\subset\mathbb{R}^n} f(\mathbf{x}), \tag{2.1}$$

where $\mathcal{X}$ is the search space under consideration defined as an n-dimensional hyperbox. In the following paragraphs, $\mathcal{R}$ defines a real-valued random number in the range (0,1).

Algorithm 2.1 Simulated Annealing

Input: Dimension of problem (n), computational budget, SA parameters

Output: Best detected solution

$t \leftarrow 0$, $\mathbf{x} \leftarrow \mathbf{x}^{(t)}$, $T \leftarrow T^{(t)}$
while (not termination) **do**
  for $(i = 1 \dots S_T)$ **do**
    randomly select $\mathbf{y} \in N_{\mathbf{x}^{(t)}} \cap \mathcal{X}$
    $\varepsilon \leftarrow f(\mathbf{y}) - f\left(\mathbf{x}^{(t)}\right)$
    if $(\varepsilon < 0)$ `OR` $(\mathcal{R} < \exp(-\varepsilon/T))$ **then**
      $\mathbf{x}^{(t)} \leftarrow \mathbf{y}$
    end if
  end for
  $T^{(t+1)} \leftarrow \alpha T^{(t)}$
  $t \leftarrow t + 1$
  `UpdateBest` $(\mathbf{x}^{(t)}, \mathbf{x}^*)$
end while
return $\mathbf{x}^*$

2.2 Simulated Annealing

Simulated Annealing (SA) is a popular trajectory-based metaheuristic, particularly used in discrete optimization problems [25]. SA is equipped with a hill-climbing mechanism, which is based on the probabilistic acceptance of non-improving solutions in order to alleviate local minimizers. Let $\mathbf{x}^{(t)} \in \mathcal{X}$ be the current position at iteration t, and $N_{\mathbf{x}^{(t)}} \subset \mathcal{X}$ be its neighborhood. Also, let $T^{(t)}$ be a parameter, called the *temperature*, which controls the probability of accepting non-improving solutions. The algorithm starts with a random initial position $\mathbf{x}^{(0)}$ and a (usually high) initial temperature $T^{(0)}$. At each iteration, SA performs a number, S_T, of inner steps with fixed temperature value. Then, the temperature is scaled according to a user-defined *cooling factor* $\alpha \in (0, 1)$, and the algorithm proceeds to the next iteration.

At each inner step, a point $\mathbf{y}$ is randomly and uniformly selected from $N_{\mathbf{x}^{(t)}}$. In case of improvement (better objective function value), the algorithm moves to the new position $\mathbf{y}$. If $\mathbf{y}$ is a non-improving position, the algorithm accepts it with probability $p = \exp(-\varepsilon/T)$, where $\varepsilon = f(\mathbf{y}) - f\left(\mathbf{x}^{(t)}\right)$. Note that the probability p increases with T. This promotes the frequent acceptance of non-improving solutions at early stages

of the optimization procedure, thereby enhancing the exploration capability of the algorithm. As T decreases, the algorithm tends to accept mostly improving solutions, amplifying its exploitation dynamic.

SA terminates execution when a predefined computational budget is exceeded or if it fails to improve the best detected solution for a number of iterations. The main procedure of the considered SA is given in Algorithm 2.1. For a detailed presentation of SA, the reader is referred to [26].

2.3 Tabu Search

Tabu Search (TS) is one of the most popular and well studied metaheuristics. Since its introduction in [27,28], TS has been applied on numerous problems spanning various fields of discrete optimization [29–31]. The basic motivation for the development of TS originated from the necessity of search algorithms to overcome local minimizers.

TS belongs to the class of trajectory-based search methods and it is based on a *local search* (LS) procedure. The algorithm is initialized on a randomly selected position in the search space $\mathcal{X}$. At each iteration, the current position moves to the best position of its neighborhood in terms of objective value. Specifically, TS moves to the best neighbor of the current position regardless of improving it. This property equips TS with *hill-climbing* capability, which is necessary for alleviating local minimizers.

In order to avoid cyclic moves, TS employs a short-term memory, called the *tabu list* (TL) where recently visited positions are stored. The positions enlisted in TL are prohibited for a number of iterations, thereby preventing the algorithm from retracing the same trajectories. The size of TL can affect the algorithm's performance. Proper values for TL are typically problem-dependent.

Moreover, in order to avoid restraining the search in narrow parts of the search space, TS is usually applied within a multistart framework. Thus, the best position of the current trajectory is recorded and, if not improved for a number of iterations, the algorithm is restarted to a new (randomly selected) initial position. Eventually, the algorithm terminates when it exceeds a predefined computational budget or the best position fails to improve for a maximum number of restarts.

Let $\mathbf{x}^{(t)} \in \mathcal{X}$ be the current position of the algorithm at iteration t, $N_{\mathbf{x}^{(t)}} \subset \mathcal{X}$ be the set of all neighbors of $\mathbf{x}^{(t)}$, $TL^{(t)}$ be the tabu list of size s_{TL} at iteration t, $N_{\mathbf{x}^{(t)}}$ be the set of all immediate neighbors of $\mathbf{x}^{(t)}$ not included in TL, and $\mathbf{x}^*$ be the best

Algorithm 2.2 Tabu Search

Input: Dimension of problem (n), computational budget, TS parameters

Output: Best detected solution

$t \leftarrow 0$
while (not termination) **do**
  $\mathbf{x}^{(t+1)} \leftarrow \arg \min_{\mathbf{y} \in N_{\mathbf{x}^{(t)}} \cap \mathcal{X}} f(\mathbf{y})$
  $TL^{(t+1)} \leftarrow TL^{(t)} \cup \left\{\mathbf{x}^{(t+1)}\right\}$ and remove the oldest entry from $TL^{(t+1)}$
  if $\left(f\left(\mathbf{x}^{(t+1)}\right) < f\left(\mathbf{x}^{*}\right)\right)$ **then**
    $\mathbf{x}^{*} \leftarrow \mathbf{x}^{(t+1)}$
  end if
  $t \leftarrow t + 1$
end while
return $\mathbf{x}^{*}$

visited position. Then the main trajectory-generating procedure of TS is outlined in Algorithm 2.2. Detailed analysis of the TS algorithm can be found in [30].

2.4 Iterated Local Search

Iterated Local Search (ILS) [32] defines a simple and straightforward framework for probing complex search spaces. Its main requirement is the use of a suitable local search procedure for the problem at hand. The local search is initiated to a randomly selected point $\mathbf{x}_{\text{init}}$ and generates a trajectory that eventually reaches the nearest local minimizer, $\mathbf{x}_{\text{min}}$. This is achieved by iteratively selecting steepest descent moves in the neighborhood of the current position.

In discrete spaces, the close neighborhood of a point is defined as the finite set of points with the smallest distance from it. Typically, Hamming distance is used for this purpose. The local search procedure scans the whole neighborhood of the current point and makes the move of highest improvement (*neighborhood-best* strategy). Alternatively, it can move to the first improving sequence found in the neighborhood (*first-best* strategy). The detected local minimizer is archived in a set S_{min}. Then, a new trajectory is started from a new initial sequence [32].

In its simplest form, ILS generates new trajectories by randomly sampling new initial sequences in the search space according to a (typically uniform) distribution. This is the well known *Random Restarts* approach. The most common stopping cri-

Algorithm 2.3 Iterated Local Search

Input: Dimension of problem (n), computational budget, ILS parameters

Output: Best detected solution

$\mathbf{x}_{\text{init}} \leftarrow$ `GetInitialSequence`$(\mathcal{X})$
$\mathbf{x}_{\text{min}} \leftarrow$ `LocalSearch`$(\mathbf{x}_{\text{init}})$
$S_{\text{min}} \leftarrow \{\mathbf{x}_{\text{min}}\}$
while (not termination) **do**
  if $(\mathcal{R} < \rho)$ **then**
    $\mathbf{x}_{\text{init}} \leftarrow$ `GetInitialSequence`(S_{min})
  else
    $\mathbf{x}_{\text{init}} \leftarrow$ `GetInitialSequence`$(\mathcal{X})$
  end if
  $\mathbf{x}_{\text{min}} \leftarrow$ `LocalSearch`$(\mathbf{x}_{\text{init}})$
  $S_{\text{min}} \leftarrow S_{\text{min}} \cup \{\mathbf{x}_{\text{min}}\}$
end while
$\mathbf{x}^* \leftarrow \arg\min_{\mathbf{x}_{\text{min}} \in S_{\text{min}}} f(\mathbf{x}_{\text{min}})$
return $\mathbf{x}^*$

teria are the detection of a prespecified number of local minimizers or a maximum computational budget in terms of running time or function evaluations. Although random restarts were shown to be sufficient in various problems, relevant research suggests that efficiency can be increased if already detected local minimizers from the set S_{min} are exploited during the search. Typically, this refers to the generation of new initial positions by perturbing previously detected local minimizers.

The two initialization approaches can be combined. Naturally, this scheme introduces new parameters to the algorithm. Specifically, the user needs to specify a probability $\rho \in [0, 1]$ of using perturbation-based restarts as well as the criteria for selecting the local minimizers from the set S_{min}.

The ILS algorithm is given in pseudocode in Algorithm 2.3. The sampling procedures for the search space $\mathcal{X}$ and the set S_{min} are implemented by the function `GetInitialSequence()`. For a comprehensive presentation of ILS the reader is referred to [32].

Algorithm 2.4 General Variable Neighborhood Search

Input: Dimension of problem (n), computational budget, VNS parameters

Output: Best detected solution

while (not termination) **do**
  $k_1 \leftarrow 1$
  while $(k_1 < K)$ **do**
    $\mathbf{x}'' \leftarrow \texttt{Perturb}\,(x, N_{\mathbf{x},k_1})$
    $k_2 \leftarrow 1$
    while $(k_2 < K)$ **do**
      $\mathbf{x}'' \leftarrow \arg\min_{\mathbf{y} \in N_{\mathbf{x}',k_2} \cap \mathcal{X}} f(\mathbf{y})$
      if $(f(\mathbf{x}'') < f(\mathbf{x}'))$ **then** $\mathbf{x}' \leftarrow \mathbf{x}''$ and $k_2 \leftarrow 1$ **else** $k_2 \leftarrow k_2 + 1$
    end while
    if $(f(\mathbf{x}') < f(\mathbf{x}))$ **then** $\mathbf{x} \leftarrow \mathbf{x}'$ and $k_1 \leftarrow 1$ **else** $k_1 \leftarrow k_1 + 1$
  end while
  $\mathbf{x}^* \leftarrow \mathbf{x}$
end while
return $\mathbf{x}^*$

2.5 Variable Neighborhood Search

Variable Neighborhood Search (VNS) [33] is a trajectory-based metaheuristic, originally designed to solve combinatorial optimization problems [34]. Important applications of VNS include the continuous location-allocation problem [35], the resource-constrained project scheduling problem [36], and the simple plant location problem [37]. A number of different variants have been proposed in literature [2]. In present thesis, the General VNS approach is employed, which couples an LS procedure with a perturbation mechanism, both guided by a systematic change of K predefined neighborhood structures.

Given an initial position $\mathbf{x} \in \mathcal{X}$ and initial neighborhood index $k = 1$, the algorithm randomly perturbs $\mathbf{x}$ and receives a new position $\mathbf{x}'$ in its neighborhood $N_{\mathbf{x},k}$. Then, local search is applied to find the best point $\mathbf{x}'' \in N_{\mathbf{x}',k}$. If $\mathbf{x}''$ does not improve $\mathbf{x}'$, the local search is applied anew in the $k+1$ neighborhood. Otherwise, $\mathbf{x}''$ becomes the new $\mathbf{x}'$ and the index k is reset to 1, continuing the same procedure for the new $\mathbf{x}'$. If there is no success after exceeding all the K neighborhoods, the algorithm backtracks to the original $\mathbf{x}$ and changes its neighborhood. If all neighborhoods are exceeded

without success, the algorithm is applied on a new initial point $\mathbf{x}$ for $k = 1$. The pseudocode of VNS is outlined in Algorithm 2.4, based on the presentation in [34]. For a comprehensive presentation of VNS the reader is referred to [2].

2.6 Differential Evolution

Differential Evolution (DE) is a well studied population-based algorithm used for continuous optimization. It was introduced by Storn and Price in [38] and, since then, it has gained increasing popularity [39]. The DE algorithm employs a population of N candidate solutions,

$$P = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\},$$

where each n-dimensional vector $\mathbf{x}_i$ is called an *individual*. The algorithm begins with an initialization phase where the individuals are randomly (usually uniformly) initialized over the corresponding search space $\mathcal{X}$. The cornerstone of the algorithm is the exploration phase where the individuals iteratively probe the search space by sampling new points through *mutation*, *crossover*, and *selection* operators.

At each iteration t, a mutated vector $\mathbf{v}_i$ is generated for each individual $\mathbf{x}_i$ by combining other individuals of the population. In the present thesis, the following well known mutation operators are considered:

$$\text{DE1} : \mathbf{v}_i^{(t+1)} = \mathbf{x}_{\text{best}}^{(t)} + F\Big(\mathbf{x}_{r_1}^{(t)} - \mathbf{x}_{r_2}^{(t)}\Big), \tag{2.2}$$

$$\text{DE2} : \mathbf{v}_i^{(t+1)} = \mathbf{x}_{r_1}^{(t)} + F\Big(\mathbf{x}_{r_2}^{(t)} - \mathbf{x}_{r_3}^{(t)}\Big), \tag{2.3}$$

$$\text{DE3} : \mathbf{v}_i^{(t+1)} = \mathbf{x}_i^{(t)} + F\Big(\mathbf{x}_{\text{best}}^{(t)} - \mathbf{x}_i^{(t)}\Big) + F\Big(\mathbf{x}_{r_1}^{(t)} - \mathbf{x}_{r_2}^{(t)}\Big), \tag{2.4}$$

$$\text{DE4} : \mathbf{v}_i^{(t+1)} = \mathbf{x}_{\text{best}}^{(t)} + F\Big(\mathbf{x}_{r_1}^{(t)} - \mathbf{x}_{r_2}^{(t)}\Big) + F\Big(\mathbf{x}_{r_3}^{(t)} - \mathbf{x}_{r_4}^{(t)}\Big), \tag{2.5}$$

$$\text{DE5} : \mathbf{v}_i^{(t+1)} = \mathbf{x}_{r_1}^{(t)} + F\Big(\mathbf{x}_{r_2}^{(t)} - \mathbf{x}_{r_3}^{(t)}\Big) + F\Big(\mathbf{x}_{r_4}^{(t)} - \mathbf{x}_{r_5}^{(t)}\Big), \tag{2.6}$$

where $\mathbf{x}_{\text{best}}^{(t)}$ denotes the individual with the best function value at iteration t. Consider the set,

$$I = \{1, 2, \ldots, N\}$$

of the indices of individuals. The indices,

$$r_j \in I \setminus \{i\}, \qquad j = 1, 2, \ldots, 5,$$

Algorithm 2.5 Differential Evolution

Input: Dimension of problem (n), computational budget, DE parameters, $I = \{1, 2, \ldots, N\}$

Output: Best detected solution

initialize population $P = \left\{\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)}, \ldots, \mathbf{x}_N^{(0)}\right\}$ randomly in $\mathcal{X}$ and set $t \leftarrow 0$
while (not termination) **do**
  for $(i = 1 \ldots N)$ **do**
    take random, mutually different indices $r_1, r_2, r_3 \in \{1, \ldots, N\} \setminus \{i\}$
    $\mathbf{v}_i^{(t+1)} \leftarrow \mathbf{x}_{r_1}^{(t)} + F\left(\mathbf{x}_{r_2}^{(t)} - \mathbf{x}_{r_3}^{(t)}\right)$
    take random index $\ell \in \{1, \ldots, n\}$
    for $(j = 1 \ldots n)$ **do**
      if $(\mathcal{R} \leqslant CR)$ OR $(j = \ell)$ **then** $u_{ij}^{(t+1)} \leftarrow v_{ij}^{(t+1)}$ **else** $u_{ij}^{(t+1)} \leftarrow x_{ij}^{(t)}$
      restrict $u_{ij}^{(t+1)}$ in $\mathcal{X}$ if boundary is violated
    end for
  end for
  for $(i = 1 \ldots N)$ **do**
    if $\left(f\left(\mathbf{u}_i^{(t+1)}\right) \leqslant f\left(\mathbf{x}_i^{(t)}\right)\right)$ **then** $\mathbf{x}_i^{(t+1)} \leftarrow \mathbf{u}_i^{(t+1)}$ **else** $\mathbf{x}_i^{(t+1)} \leftarrow \mathbf{x}_i^{(t)}$
  end for
  set $t \leftarrow t + 1$ and $\mathbf{x}^* \leftarrow \arg\min_{\mathbf{x} \in P^{(t)}} f(\mathbf{x})$
end while
return $\mathbf{x}^*$

are randomly selected and mutually different. The *differential weight*, $F \in [0, 2]$, is a user-defined constant that controls the degree of expansion towards the directions defined by the difference vectors.

Mutation is followed by crossover, where a trial vector $\mathbf{u}_i$ is produced for each individual $\mathbf{x}_i$ in the following way:

$$u_{ij}^{(t)} = \begin{cases} v_{ij}^{(t+1)}, & \text{if } \mathcal{R} \leqslant CR \text{ or } j = RI(i), \\ x_{ij}^{(t)}, & \text{otherwise,} \end{cases} \tag{2.7}$$

where $CR \in [0, 1]$ is a user-defined scalar called the *crossover probability*, and $RI(i) \in \{1, 2, \ldots, n\}$ is a random integer uniformly selected for each individual $\mathbf{x}_i$ of the population.

Finally, selection decides whether the trial vector shall replace the corresponding original individual. Specifically, the replacement occurs if the trial vector improves in

function value the original individual, i.e.,

$$\mathbf{x}_i^{(t+1)} = \begin{cases} \mathbf{u}_i^{(t+1)}, & \text{if } f\left(\mathbf{u}_i^{(t+1)}\right) < f\left(\mathbf{x}_i^{(t)}\right) \\ \mathbf{x}_i^{(t)}, & \text{otherwise.} \end{cases} \tag{2.8}$$

The DE algorithm has been shown to be sensitive on its parameters N, F, and CR. Therefore, appropriate parameterization has significant impact on its performance and it is highly dependent on the considered problem. A detailed pseudocode of the DE algorithm can be found in Algorithm 2.5. Additionally, a concise presentation of research related to the DE algorithm can be found in [39].

Enhanced Differential Evolution

In [40] an enhanced DE (eDE) variant was proposed. It defines an alternative mutation scheme, while crossover is based on probabilistic selection between the new and the DE2 scheme of Eq. (2.3). Moreover, the algorithm is enhanced by using restart to alleviate local minima.

Putting it formally, eDE introduces the mutation scheme,

$$\mathbf{w}_i^{(t+1)} = \mathbf{x}_{r_1}^{(t)} + F_1 \left(\mathbf{x}_{\text{best}}^{(t)} - \mathbf{x}_{r_1}^{(t)}\right) + F_2 \left(\mathbf{x}_{r_1}^{(t)} - \mathbf{x}_{\text{worst}}^{(t)}\right), \tag{2.9}$$

where $\mathbf{x}_{r_1}^{(t)}$ is a randomly selected individual, $F_1, F_2 \in [0, 2]$ are called the *differential weights*, and $\mathbf{x}_{\text{best}}^{(t)}$, $\mathbf{x}_{\text{worst}}^{(t)}$ denote the best and worst individuals at iteration t, respectively. The trial vector is given as follows,

$$u_{ij}^{(t+1)} = \begin{cases} w_{ij}^{(t+1)}, & \text{if } \left(\mathcal{R} \leqslant CR \text{ or } j = RI(i)\right) \text{ and } \mathcal{R} \geqslant \left(1 - \frac{t}{t_{\max}}\right), \\ v_{ij}^{(t+1)}, & \text{if } \left(\mathcal{R} \leqslant CR \text{ or } j = RI(i)\right) \text{ and } \mathcal{R} < \left(1 - \frac{t}{t_{\max}}\right), \\ x_{ij}^{(t)}, & \text{otherwise,} \end{cases} \tag{2.10}$$

where $t_{\max}$ is the total number of iterations. The rest of the parameters are identical to the standard DE. Also, note that v_{ij} is the j-th component of the mutation vector $\mathbf{u}_i$ produced through Eqs. (2.2)-(2.6).

A restarting mechanism is also incorporated into eDE to avoid premature convergence. The restarting mechanism is applied on each individual except for the best one, which is kept unaltered. Specifically, restarts from mild perturbations $\mathbf{x}'_i$ of current individuals $\mathbf{x}_i$ are adopted, as follows,

$$x'_{ij} = x_{ij} \pm 1. \tag{2.11}$$

According to this scheme, the probability of plunging into local minima is drastically decreased, while the local search capability is enhanced through the perturbation of individuals' position by ± 1. The sign "$+$" or "$-$" in Eq. (2.11) is randomly selected with probability 0.5 for each j. The bias is selected equal to 1 since it constitutes the smallest step size in integer search spaces.

2.7 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population-based algorithm that models social behavior to effectively solve global optimization problems by guiding swarms of particles towards the most promising regions of the search space. It was originally introduced by Eberhart and Kennedy [17] in 1995 and, since then, it has gained increasing popularity. This can be ascribed to its efficiency and effectiveness in solving hard optimization problems with minor programming effort. Up-to-date there is a considerable amount of PSO-based applications in various scientific and technological fields [18, 41–45].

Synchronous Particle Swarm Optimization

In this section, a presentation of the original (synchronous) PSO algorithm is provided. Consider the sets,

$$I = \{1, 2, \ldots, N\}, \qquad D = \{1, 2, \ldots, n\},$$

which denote the indices of the search points and the indices of direction components, respectively. PSO employs a set of search points,

$$S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\},$$

which is called a *swarm*, to iteratively probe the search space $\mathcal{X}$. Each search point is an n-dimensional vector,

$$\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in})^\top \in \mathcal{X}, \quad i \in I,$$

called a *particle*. Each particle explores the search space by moving to new positions (candidate solutions) in $\mathcal{X}$ and adjusts its exploratory behavior according to its own findings as well as the findings of the other particles.

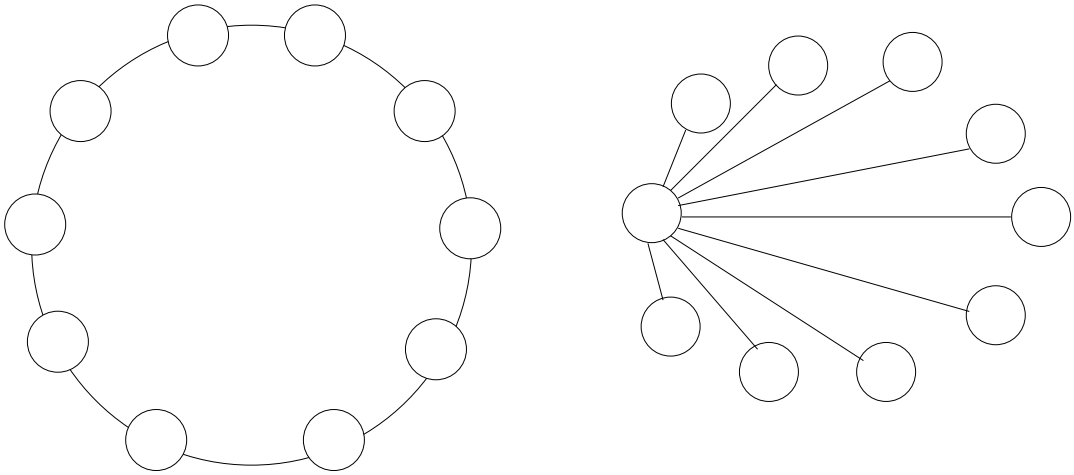


Figure 2.1: Neighborhood topologies: ring (left) and star (right).

During its quest for better solutions, each particle records in memory the *best position* it has encountered. In minimization problems, this position has the lowest objective value among all positions visited by the particle. If t denotes the iteration counter and $\mathbf{x}_i^{(t)}$ are the subsequent positions of the i-th particle, then its best position is denoted as,

$$\mathbf{p}_i = (p_{i1}, p_{i2}, \ldots, p_{in})^\top \in \mathcal{X}, \quad i \in I,$$

and defined as,

$$\mathbf{p}_i^{(t)} = \arg \min_{\tau=0,1,2,\ldots,t} \left\{ f\left(\mathbf{x}_i^{(\tau)}\right) \right\}.$$

The particle moves in the search space by using an adaptable position shift, called *velocity*,

$$\mathbf{v}_i = (v_{i1}, v_{i2}, \ldots, v_{in})^\top, \quad i \in I,$$

which is added to its current position.

The velocity of each particle is updated at each iteration by taking into consideration its own best position and the best position among a set of adjacent particles, which constitute its *neighborhood* [46, 47]. The adjacency between particles is determined according to arbitrary interconnection schemes that allow groups of particles to exchange information among them. These schemes are called *neighborhood topologies*, and they are usually visualized as undirected graphs where nodes denote the particles and edges denote communication channels. Figure 2.1 illustrates two common neighborhood topologies.

Various neighborhood topologies have been proposed in the literature. The most common one is the *ring* topology, illustrated in the left part of Fig. 2.1, where each

particle assumes as neighbors the particles with adjacent indices. The size of the neighborhood is determined by a parameter r called the *neighborhood's radius*. Formally, a ring neighborhood of radius r of the i-th particle is defined by the set of indices,

$$NB_{i,r} = \{i-r,\ \ldots\ ,\ i-1,\ i,\ i+1,\ \ldots\ ,\ i+r\}. \tag{2.12}$$

This means that the best position among the ones with indices from $i-r$ up to $i+r$ is used for the i-th particle's velocity update. The indices are assumed to recycle at their limits, i.e., the particle with index 1 follows immediately after the one with index N.

Based on the neighborhood size, two prevailing PSO models have been established. The first one, called the *global PSO* model (denoted as *gbest*), assumes the whole swarm as neighborhood of each particle. Thus, the overall best position of the swarm is used to update all particles' velocities. This approach was mainly used in early PSO variants and exhibited rapid convergence (exploitation) properties. However, rapid convergence was habitually accompanied by loss of diversity, leading to premature convergence in undesirable suboptimal solutions. On the other hand, using significantly smaller neighborhoods can enhance the exploration properties of the swarm. This is attributed to the limited connectivity among the particles, which restricts the rapid diffusion of the detected best positions to the rest of the swarm. This approach defines the *local PSO* model (denoted as *lbest*).

Let $\mathbf{p}_{g_{(i,t)}}$ denote the best position in the neighborhood of the i-th particle at iteration t, i.e.,

$$g_{(i,t)} = \arg\min_{j \in NB_{i,r}} \left\{ f\left(\mathbf{p}_j^{(t)}\right) \right\}.$$

Then, based on the definitions above, the update equations of PSO are given as follows [48]:

$$\mathbf{v}_i^{(t+1)} = \chi\Big[\mathbf{v}_i^{(t)} + c_1\mathcal{R}_1 \otimes \left(\mathbf{p}_i^{(t)} - \mathbf{x}_i^{(t)}\right) + c_2\mathcal{R}_2 \otimes \left(\mathbf{p}_{g_{(i,t)}}^{(t)} - \mathbf{x}_i^{(t)}\right)\Big], \tag{2.13}$$

$$\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \mathbf{v}_i^{(t+1)}, \tag{2.14}$$

where $i \in I$, and $\otimes$ denotes componentwise multiplication of vectors. The parameter χ is called the *constriction coefficient* and it is used to clamp the velocities in order to avoid the swarm explosion effect [48]. The scalars c_1 and c_2 are called the *cognitive* and *social* parameter, respectively, and they are used to bias velocity towards either the particle's own best position or the neighborhood's best position. The parameters $\mathcal{R}_1$ and $\mathcal{R}_2$ are random vectors that induce stochasticity in the algorithm. Their components are drawn from the uniform distribution $\mathcal{U}(0,1)$.

Algorithm 2.6 Synchronous Particle Swarm Optimization

Input: Dimension of problem (n), computational budget, PSO parameters, $I = \{1, 2, \ldots, N\}$

Output: Best detected solution

initialize swarm $S = \left\{\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)}, \ldots, \mathbf{x}_N^{(0)}\right\}$ randomly in $\mathcal{X}$ and set $t \leftarrow 0$
initialize best positions $\mathbf{p}_i^{(t)} \leftarrow \mathbf{x}_i^{(t)}$ and velocities $\mathbf{v}_i^{(t)}$, $\forall\, i = 1, \ldots, N$
while (not termination) **do**
  for $(i = 1 \ldots N, j = 1 \ldots n)$ **do**
    $v_{ij}^{(t+1)} \leftarrow \chi\Big[v_{ij}^{(t)} + \mathcal{R}_1\Big(p_{ij}^{(t)} - x_{ij}^{(t)}\Big) + \mathcal{R}_2\Big(p_{g_{(i,t)}j}^{(t)} - x_{ij}^{(t)}\Big)\Big]$
    $x_{ij}^{(t+1)} \leftarrow x_{ij}^{(t)} + v_{ij}^{(t+1)}$
  end for
  for $(i = 1 \ldots N)$ **do**
    if $\left(f\left(\mathbf{x}_i^{(t+1)}\right) \leqslant f\left(\mathbf{p}_i^{(t)}\right)\right)$ **then** $\mathbf{p}_i^{(t+1)} \leftarrow \mathbf{x}_i^{(t+1)}$ **else** $\mathbf{p}_i^{(t+1)} \leftarrow \mathbf{p}_i^{(t)}$
  end for
  update indices $g_{(i,t)}, \forall\, i = 1, \ldots, N$
  set $t \leftarrow t + 1$ and $\mathbf{x}^* \leftarrow \arg\min_{\mathbf{p}_i^{(t)}} f\left(\mathbf{p}_i^{(t)}\right)$
end while
return $\mathbf{x}^*$

After updating all particles, their new positions compete against their best positions. Thus, the best position of each particle is updated as follows,

$$\mathbf{p}_i^{(t+1)} = \begin{cases} \mathbf{x}_i^{(t+1)}, & \text{if } f\left(\mathbf{x}_i^{(t+1)}\right) < f\left(\mathbf{p}_i^{(t)}\right), \\ \mathbf{p}_i^{(t)}, & \text{otherwise,} \end{cases} \tag{2.15}$$

where $i \in I$.

The presented variant of PSO is supported by thorough stability and convergence analysis [48], which suggested the general-purpose parameter setting,

$$\chi = 0.729, \quad c_1 = c_2 = 2.05.$$

This is considered to be a satisfactory setting that produces balanced convergence speed for the algorithm. Nevertheless, alternative successful settings have also been proposed in the literature [49].

Instead of using the χ coefficient, Eq. (2.13) can be redefined by using the inertia weight of the particle, denoted as ω, as follows:

$$\mathbf{v}_i^{(t+1)} = \omega \mathbf{v}_i^{(t)} + c_1 \mathcal{R}_1 \otimes \left(\mathbf{p}_i^{(t)} - \mathbf{x}_i^{(t)}\right) + c_2 \mathcal{R}_2 \otimes \left(\mathbf{p}_{g_{(i,t)}}^{(t)} - \mathbf{x}_i^{(t)}\right) \tag{2.16}$$

The inertia weight can be adaptive and change linearly throughout the optimization process. A widely used approach is based on the following rule:

$$\omega = \omega_{\max} - \frac{(\omega_{\max} - \omega_{\min})\, t}{t_{\max}}, \tag{2.17}$$

where $\omega_{\min}$ and ω_{max} define its range, t is the iteration counter, and $t_{\max}$ is the maximum number of iterations. At the beginning of the optimization process, Eq. (2.17) allows the inertia weight to take high values, thereby promoting exploration, whereas as ω reduces, better exploitation properties are achieved.

In case of discrete optimization problems, the update of the velocity can be properly modified. As suggested in [21, 50], each element of the velocity vector is transformed as follows:

$$v_{ij}^{(t+1)} = \begin{cases} \left\lfloor v_{ij}^{(t+1)} \right\rfloor, & \text{if } \mathcal{R} \leqslant \lambda, \\ \left\lceil v_{ij}^{(t+1)} \right\rceil, & \text{otherwise}, \end{cases} \tag{2.18}$$

where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ are the floor and ceiling functions, respectively. The parameter λ determines the probability of using the floor or ceiling function in the computation of the velocity.

The standard PSO algorithm allocates one function evaluation per particle per iteration. Hence, at the end of its execution, all particles have spent equal portions of the available computational budget. Moreover, the update of Eqs. (2.13), (2.14), and (2.15), is synchronous, i.e., the new best positions are determined only after the position update of all particles. A detailed pseudocode of the sychronous version of the PSO algorithm can be found in Algorithm 2.6. Alternatively, asynchronous PSO variants have been developed.

Asynchronous Particle Swarm Optimization

Asynchronous PSO variants have been developed as alternatives to the standard (synchronous) approach. Contrary to synchronous PSO, in the asynchronous model each particle updates and communicates at once its new best position to its neighbors, without waiting for the rest of the particles to update their memory. The immediate exposition of the particles to new findings has significant impact on their convergence speed. Also, it can radically reduce the algorithm's runtime in parallel implementations on inhomogeneous systems or problems with high diversity of function evaluation time.

On the other hand, rapid convergence of asynchronous PSO can lead the swarm to deceitful positions more frequently than the synchronous approach. Thus, it increases the probability of getting trapped in low-quality solutions. Therefore, special attention is required when selecting between the synchronous and the asynchronous model.

2.8 Computational Budget Allocation

The proper distribution of computational resources among the operations or procedures of a metaheuristic algorithm can have significant impact on its performance. Given the initial computational budget, the important decision lies on the allocation schedule of the available resources. The resources allocation plan can be *static*, i.e. predefined and unchanged throughout the optimization process. Alternatively, a *dynamic* allocation plan can be used, which is deployed along with the algorithm. During its execution, the algorithm offers feedback to the allocation mechanism in order to adjust its decision.

Numerous research works consider metaheuristic algorithms that exploit budget allocation mechanisms. A large body of work has been devoted to the study of population-based algorithms equipped with the Optimal Computing Budget Allocation (OCBA) method [51] in order to cope with optimization problems contaminated by noise [52–55]. Additionally, there are a few research works that use metaheuristics to address resources allocation issues that are met in a multitude of optimization problems [56].

2.8.1 Neighborhood-based Budget Allocation

The standard PSO algorithm considers all particles of the swarm to be equally important. Thus, it synchronously allocates the same fraction of function evaluations to each one. On the other hand, it would be reasonable to promote the search in the most promising regions of the search space by favoring the particles that probe such regions. Due to the inherent collective dynamics of PSO, these particles communicate their experience also to their neighbors, thereby offering them an opportunity to enhance their performance. For this reason, the idea of using neighborhood characteristics and qualities to identify and favor some of the particles in the budget allocation procedure is appealing.

In this section, a novel PSO variant is proposed, called *Particle Swarm Optimization with Neighborhood-based Budget Allocation*, henceforth denoted as PSO-NBA. The new algorithm employs two essential budget allocation strategies to assess the quality of the neighborhoods. The two strategies are based on single-objective and multi-objective scoring modes, respectively. The single-objective approach is based on the total or, alternatively, on the best information carried by the neighborhood in terms of objective values. The multi-objective approach takes into consideration also another aspect of quality, namely the diversity of the neighborhood. In this case, each neighborhood is assessed on the basis of a 2-dimensional scoring vector. The first component of the vector is identical to the solution quality criterion of the single-objective approach. The second component of the vector depends on the diversity of the best positions of the particles that comprise the neighborhood. Then, a scheme that is based on the concept of Pareto dominance is used for the selection among the neighborhoods.

In literature there are works that propose rank-based PSO variants. In [57] the algorithm uses only a fraction of the particles to update velocity. This approach is solely based on the global (gbest) PSO model, neglecting the neighborhoods. In [58] the proposed approach uses ranking in order to replace low-fitness particles with better ones. A relevant (although not rank-based) asynchronous PSO variant is PSO-DLI [1], which employs a special scheme to allocate function evaluations to some of the particles while the rest remain idle. The proposed approach differs also from these approaches, since neighborhood ranking schemes are used to allocate the available computational budget in a sophisticated manner. It shall be mentioned that this is the first study that uses rank-based criteria to assess the quality of neighborhoods and dynamically distribute the available computational budget among the corresponding particles.

2.8.1.1 Neighborhood Quality Criteria

The two essential properties that define the dynamics of any population-based optimization algorithm are *exploration* (diversification) and *exploitation* (intensification). The first one is the ability of the algorithm to explore different parts of the search space, while the second one is the ability to perform more refined search around the discovered solutions. Proper balancing between these properties has been associated with highly competitive optimization algorithms.

It is easily inferred (and experimentally verified) that these two properties are

Table 2.1: Neighborhood quality criteria of PSO-NBA.

Type	Criterion	Abbrev.	Description
Solution quality	*SumBest*	SB	Sum of all objective values in the neighborhood.
	LocalBest	LB	Best objective value in the neighborhood.
Diversity	*AvgDev*	AD	Average standard deviation of direction components of the best positions that comprise the neighborhood.

intimately related with two performance indices of the algorithm, namely *solution quality* and *diversity*. The most successful approaches are expected to retain adequate diversity in the swarm such that search stagnation is alluded, while concurrently improving solution quality within reasonable time limits.

Transferring these concepts from swarm level to the neighborhood level, two types of neighborhood quality criteria are considered. The first type refers to solution quality and consists of two alternative schemes, while the second type refers to diversity. Specifically, the first solution quality criterion, denoted as *SumBest* (SB), is based on the total solution information carried by the neighborhood in terms of objective values. Thus, each neighborhood is assessed according to the collective achievements of its members.

The second solution quality criterion, denoted as *LocalBest* (LB), takes into consideration only the best position attained by the neighborhood's members. Thus, it clearly promotes elitism. Regarding diversity, a criterion denoted as *AvgDev* (AD) is considered, which assesses each neighborhood in terms of diversity of the best positions that comprise it. The three criteria are summarized in Table 2.1, and they are formally defined below.

SumBest (SB)

Let $NB_{i,r}$ be the neighborhood of the i-th particle as defined in Eq. (2.12). Then, its SB ranking score at iteration t is defined as,

$$SBR_i = \sum_{k \in NB_{i,r}} f\left(\mathbf{p}_k^{(t)}\right), \quad i \in I. \tag{2.19}$$

Thus, the SB score assesses the neighborhood's quality in terms of the sum of the objective values of all best positions that comprise it. In order to facilitate the use

of SB scores for the computation of the neighborhoods' selection probabilities, a normalization step takes place,

$$SBR_i^* = \frac{SBR_i}{\sum\limits_{m \in I} SBR_m}, \quad i \in I. \tag{2.20}$$

Lower values of the SB ranking score correspond to neighborhoods that possess lower cumulative information in terms of their objective values. These neighborhoods are considered to be superior than the ones with higher scores and, hence, their corresponding particles shall be assigned higher selection probabilities in subsequent steps of the algorithm.

LocalBest (LB)

Let again $NB_{i,r}$ be the neighborhood of the i-th particle. Then, the LB ranking score for this neighborhood is defined as,

$$LBR_i = \min_{k \in NB_{i,r}} f\left(\mathbf{p}_k^{(t)}\right), \quad i \in I. \tag{2.21}$$

The LB score promotes elitism by assessing the neighborhood's quality only in terms of the best position involved in it. Normalization takes place also in this case,

$$LBR_i^* = \frac{LBR_i}{\sum\limits_{m \in I} LBR_m}, \quad i \in I. \tag{2.22}$$

Similarly to SB, particles with neighborhoods of lower LB ranking scores shall be assigned higher selection probabilities.

AvgDev (AD)

This diversity measure is based on the average standard deviation of the direction components of the best positions that comprise the neighborhood $NB_{i,r}$. Thus, if $NB_{i,r} = \{k_1, \ldots, k_r\}$ and $D = \{1, 2, \ldots, n\}$, the standard deviation per direction component $j \in D$ is first computed,

$$\sigma_j^{[i]} = \text{standard deviation of vector } \begin{pmatrix} p_{k_1 j}^{(t)} \\ \vdots \\ p_{k_r j}^{(t)} \end{pmatrix}. \tag{2.23}$$

where p_{kj} stands for the j-th component of the best position $\mathbf{p}_k$, $k \in NB_{i,r}$.

Then, the AD ranking score for the neighborhood is obtained by averaging the standard deviations over all dimensions,

$$AD_i = \frac{1}{n}\sum_{j=1}^{n}\sigma_j^{[i]}. \tag{2.24}$$

The obtained values are normalized as follows,

$$AD_i^* = \frac{AD_i}{\sum\limits_{m\in I} AD_m}, \quad i \in I. \tag{2.25}$$

Obviously, neighborhoods with higher AD scores contain more dispersed best positions. Thus, they are preferable against neighborhoods of lower scores in order to promote exploration.

Also, contrary to the solution quality scores SB and LB, which are based on objective values, the AD scores are based on the actual positions of the neighborhood's members in the search space.

2.8.1.2 Selection Probability

After the computation of the neighborhoods' ranking scores, each particle is assigned a *selection probability* based on the score of its neighborhood. Two alternative selection probability schemes were considered. Let xBR denote the selected ranking scheme (SB or LB), i.e.,

$$xBR_i^* = SBR_i^* \ \text{ or } \ LBR_i^*, \quad \forall i \in I.$$

Let also,

$$Q = \left\{xBR_{k_1}^*, xBR_{k_2}^*, \ldots, xBR_{k_N}^*\right\}, \quad k_i \in I,$$

be the ordering of the neighborhoods' ranking scores, sorted from the highest to the lowest value, and

$$q_i = \text{position of } i\text{-th neighborhood's score } xBR_i^* \text{ in Q}.$$

Then, the first selection probability scheme is the well known *linear ranking* that is widely used in Genetic Algorithms (GAs) [59]. This scheme assigns selection probabilities that are linear with respect to xBR_i^* as follows,

$$LPR_i = 2 - s + 2(s-1)\frac{q_i - 1}{(N-1)}, \quad i \in I, \tag{2.26}$$

where the parameter $s \in [1, 2]$ is called the *selection pressure*, and N is the total number of neighborhoods (equal to swarm size). Note that intense elitism is promoted when

$s = 2$, while equal selection probabilities are assigned to all neighborhoods when it is equal to $s = 1$.

The corresponding selection probability of the i-th particle becomes,

$$SP_i = \frac{LPR_i}{\sum_{m \in I} LPR_m}, \quad i \in I. \tag{2.27}$$

Henceforth, this scheme will be denoted as L (linear).

The second selection probability scheme comes again from the field of GAs and it is nonlinear, henceforth denoted as NL,

$$NLPR_i = (xBR_i^*)^{-\rho}, \quad i \in I, \tag{2.28}$$

where ρ is a positive integer. This scheme resembles the *power selection* operator in GAs [59]. The corresponding selection probabilities for this scheme are given as,

$$SP_i = \frac{NLPR_i}{\sum_{m \in I} NLPR_m}, \quad i \in I. \tag{2.29}$$

Clearly, higher values of the power weight ρ favor elitism since they result in higher selection probabilities for the neighborhoods with lower ranking scores xBR_i^*.

The neighborhoods' selection probabilities, computed either linearly through Eqs. (2.26) and (2.27) or nonlinearly through Eqs. (2.28) and (2.29), are used as input in a stochastic selection mechanism that determines the particle that will receive the next function evaluation. This mechanism can use either the selection probabilities solely or take into consideration also the AD diversity criterion defined in Section 2.8.1.1. The first case is referred as the *Single-Objective Budget Allocation Strategy* (SOBA), and the second one as the *Multi-Objective Budget Allocation Strategy* (MOBA). Both strategies are analyzed in the following sections.

2.8.1.3 Single-Objective Budget Allocation Strategy

In the *Single-Objective Budget Allocation* (SOBA) strategy, the selection probabilities are fed as input in a stochastic selection mechanism, neglecting the AD diversity criterion. The employed selection mechanism is the *fitness proportionate selection* technique from GAs literature, also known as *roulette-wheel* [59]. This scheme makes a randomized decision among the competitors based on their selection probabilities.

Obviously, particles with neighborhoods of higher values SP_i have higher probability of being selected. However, it is still possible that particles with inferior selection

Algorithm 2.7 SOBA strategy (for SB and LB) and MOBA strategy (for LWA and DWA)

Input: Strategy (S), computational budget ($FE_{\max}$), PSO parameters, $I = \{1, 2, \ldots, N\}$

Output: Best detected solution

initialize $\mathbf{x}_i$, $\mathbf{v}_i$, $\mathbf{p}_i$, $\forall i \in I$
compute selection probabilities SP_i, $\forall i \in I$
$t \leftarrow 0$
while $(t \leqslant FE_{max})$ **do**
  $k \leftarrow$ `RouletteWheel`$(SP_1, \ldots, SP_N)$
  update $\mathbf{v}_k$ and $\mathbf{x}_k$ according to Eqs. (2.13) and (2.14)
  update $\mathbf{p}_k$ according to Eq. (2.15)
  if ($\mathbf{p}_k$ has changed) **then**
    if (S = SOBA with SB or LB) **then**
      $I_k = \{i \in I; k \in NB_{i,r}\}$
      update neighborhoods $NB_{j,r}$, $j \in I_k$
      compute SP_i, $\forall i \in I$
    else if (S = MOBA with LWA or DWA) **then**
      compute $w_1(t)$, $w_2(t)$, according to Eq. (2.32) or Eq. (2.33)
      update SP_i, $\forall i \in I$
    end if
  end if
  $t \leftarrow t + 1$
end while

probabilities are selected due to the stochastic nature of the selection scheme. A pseudocode for the application of the SOBA selection strategy is given in Algorithm 2.7.

2.8.1.4 Multi-Objective Budget Allocation Strategy

The *multi-objective budget allocation* (MOBA) strategy takes into consideration both solution quality and diversity of the neighborhood. In general, there are two alternative multi-objective approaches to combine the two criteria. The first one is the *weighted aggregation* approach, which uses weighted combinations of the two criteria. The second one is the *Pareto front* approach, which is based on the concept of Pareto dominance. Both approaches are described in the following sections.

Weighted Aggregation

The *weighted aggregation* is a popular technique for coping with multiple objectives [60, 61]. Its popularity lies in the transformation of the multi-objective problem to a single-objective one, which allows the use of a wide variety of optimization methods.

In this approach, the two objectives are the solution quality, which is related to the exploitation property of the algorithm, and the diversity, which is related to the exploration property. Nonnegative weights are used to balance their contribution in an aggregated score defined as,

$$F = w_1(t) \times \text{quality} + w_2(t) \times \text{diversity}, \tag{2.30}$$

where $w_2(t) = 1 - w_1(t)$, and t is the counter of function evaluations. The quality-based component is the selection probability SP_i as computed in Section 2.8.1.2. The diversity-based component is AD_i^* defined in Section 2.8.1.1. Therefore, Eq. (2.30) becomes,

$$F_i = w_1(t)\, SP_i + w_2(t)\, AD_i^*, \quad i \in I \tag{2.31}$$

with $w_2(t) = 1 - w_1(t)$. Notice that both SP_i and AD_i^* are better when they receive higher values. Thus, higher aggregated values F_i are better. The values F_i are normalized and fed as probabilities in a roulette-wheel selection scheme similarly to SOBA.

The weights can either remain fixed or be dynamically adjusted during the optimization process. In general, different phases of the optimization process require different exploration-exploitation trade-off of the algorithm. Since the case of fixed weights neglect this necessity, dynamically changing weights were adopted in the proposed approach.

There are two widely used schemes for dynamically changing weighted aggregation. The first one is the *linear weighted aggregation* (henceforth denoted as LWA), where the weights are defined as follows,

$$w_1(t) = \frac{t}{FE_{\max}}, \qquad w_2(t) = 1 - w_1(t), \tag{2.32}$$

where $FE_{\max}$ is the maximum budget of function evaluations and t is their counter. Note that, at the early stages of the optimization process the diversity component is favored in order to promote better exploration of the search space, whereas the quality component is promoted at later stages in order to intensify the search near the most promising candidate solutions.

Algorithm 2.8 MOBA strategy (PFA approach)

Input: Computational budget ($FE_{\max}$), PSO parameters, $I = \{1, 2, \ldots, N\}$

Output: Best detected solution

initialize $\mathbf{x}_i$, $\mathbf{v}_i$, $\mathbf{p}_i$, (xBR_i^*, AD_i^*), $\forall i \in I$
$t \leftarrow 0$
while $(t \leqslant FE_{max})$ **do**
  $I' \leftarrow$ `Tournament`$\{(xBR_1^*, AD_1^*), \ldots, (xBR_N^*, AD_N^*)\}$
  $I^* \leftarrow$ `Non-Dominated`(I')
  for all $i^* \in I^*$ **do**
    update $\mathbf{v}_{i^*}$, $\mathbf{x}_{i^*}$ according to Eqs. (2.13) and (2.14)
    update $\mathbf{p}_{i^*}$ according to Eq. (2.15)
  end for
  if (some $\mathbf{p}_{i^*}$ has changed) **then**
    for all $i^* \in I^*$ **do**
      update xBR_j^*, for all j with $i^* \in NB_{j,r}$, according to Eq. (2.19) or Eq. (2.21)
      update AD_j^*, for all j with $i^* \in NB_{j,r}$, according to Eq. (2.24)
    end for
  end if
  $t \leftarrow t + 1$
end while

The second scheme is the *dynamic weighted aggregation* (denoted as DWA). The weights are modified as follows,

$$w_1(t) = |\sin(2\pi t \,/\, FR)\,|, \qquad w_2(t) = 1 - w_1(t), \tag{2.33}$$

where t is the counter of function evaluations and FR is the weights' change frequency. The use of the trigonometric function implies the interchange between exploration and exploitation, repeatedly.

Pseudocode for the LWA and DWA schemes is given in Algorithm 2.7. Note that, the sole difference between the SOBA and the weighted aggregation approach lies in the employed neighborhood scoring scheme.

Pareto Front Approach

In the *Pareto front* approach (henceforth denoted as PFA), the 2-dimensional scoring vector is maintained,

$$(xBR_i^*, AD_i^*), \quad i \in I,$$

Table 2.2: Dimensions and ranges of the test problems.

Problem	Dimension	Range
TP0	10, 50, 100	$[-100, 100]^n$
TP1	10, 50, 100	$[-30, 30]^n$
TP2	10, 50, 100	$[-5.12, 5.12]^n$
TP3	10, 50, 100	$[-600, 600]^n$
TP4	10, 50, 100	$[-20, 30]^n$
TP5	10	$[-2, 2]^{10}$
TP6	6	$[-10, 10]^6$
TP7	5	$[-10, 10]^5$
TP8	8	$[-10, 10]^8$
TP9	10	$[-10, 10]^{10}$
TP10	20	$[-10, 10]^{20}$

for each neighborhood, where the xBR_i^* is related to solution quality while AD_i^* is the diversity criterion (see Sections 2.8.1.2 and 2.8.1.1, respectively). Alternatively, the selection probability SP_i can be used instead of xBR_i^* with minor modifications.

The core idea behind PFA is the promotion of the non-dominated neighborhoods with respect to the two criteria, in terms of the multi-objective optimization concepts of domination and Pareto optimality [62]. Thus, a neighborhood with scoring vector (xBR_i^*, AD_i^*) is dominated by another one with scoring vector (xBR_j^*, AD_j^*) if it holds that,

$$xBR_j^* < xBR_i^* \quad \text{and} \quad AD_j^* \geqslant AD_i^*,$$

or,

$$AD_j^* > AD_i^* \quad \text{and} \quad xBR_j^* \leqslant xBR_i^*.$$

Note that larger values of diversity and lower values of the solution quality score are preferable.

The non-dominated neighborhoods are candidates for gaining function evaluations through a *tournament selection* scheme. Specifically, at each iteration of the algorithm a prespecified number (tournament size) of particles are selected from the swarm. The particles (among the selected) whose neighborhoods are non-dominated are awarded one function evaluation each. Obviously, the allocated number of func-

Table 2.3: Parameter values for the considered SOBA and MOBA strategies.

Parameter	Value
PSO model	lbest
PSO parameters	χ = 0.729, $c1$ = $c2$ = 2.05
Neighborhood topology	Ring
Neighborhood radius	1
Quality criteria	SumBest (SB), LocalBest (LB)
Diversity criterion	AvgDev(AD)
Selection scheme	Linear (L), Nonlinear (NL)
Selection pressure	$s \in \{1.0, 1.5, 2.0\}$
Nonlinear weight	$\rho \in \{1.0, 2.0\}$
Problem dimensions	$n = \{10, 50, 100\}$
Swarm size	$N = 10 \times n$
Function evaluations	$FE_{\max} = 1000 \times n$
Tournament size	$T \in \{N/2, N/3, N/5\}$
Weight's change frequency	$FR = 200$
Number of experiments	100 per approach

tion evaluations can differ from one iteration to another. The pseudocode of the PFA appoach is given in Algorithm 2.8.

The use of tournament selection instead of all non-dominated neighborhoods allows to address search stagnation. Specifically, it was frequently observed that a few (usually one or two) neighborhoods could dominate all others at early stages of the algorithm's execution and, thus, collect almost all the allocated computational budget. This was proved to be detrimental for the algorithm's exploration ability, leading to search stagnation. The stochasticity of tournament selection provides the option of assigning function evaluations also to particles with neighborhoods of low quality and diversity, thereby amplifying the algorithm's exploration capability.

2.8.2 Experimental Results

PSO-NBA was initially assessed over two test suites. The first one consists of five widely used test functions (TP0-TP4), while the second one contains six problems (TP5-TP10) that come from real-world applications and they are modeled as systems of nonlinear equations. The descriptions of the test problems are provided in Appendix A, while their dimensions and ranges used in the experimental setting are

Table 2.4: Results for the SOBA approach for test problems TP0-TP4 (standard test suite).

Problem	Dimension	Algorithm	Mean	StD	Min	Max
TP0	10	SB/L/2.0	$3.535e-02$	$3.528e-02$	$2.919e-04$	$1.645e-01$
		LB/L/2.0	$2.131e-02$	$2.303e-02$	$1.404e-03$	$1.153e-01$
		SB/NL/1.0	$8.199e-02$	$3.127e-01$	$4.868e-06$	$2.889e+00$
		LB/NL/2.0	$\mathbf{9.406e-26}$	$8.806e-25$	$1.523e-35$	$8.807e-24$
	50	SB/L/2.0	$2.092e+03$	$4.243e+02$	$1.305e+03$	$3.343e+03$
		LB/L/2.0	$1.980e+03$	$4.118e+02$	$8.238e+02$	$3.322e+03$
		SB/NL/2.0	$5.378e+00$	$1.349e+01$	$3.603e-03$	$1.003e+02$
		LB/NL/2.0	$\mathbf{3.116e-08}$	$1.332e-07$	$1.762e-012$	$1.293e-06$
	100	SB/L/2.0	$1.758e+04$	$2.471e+03$	$1.218e+04$	$2.364e+04$
		LB/L/2.0	$1.680e+04$	$2.199e+03$	$1.093e+04$	$2.532e+04$
		SB/NL/2.0	$3.055e+02$	$1.728e+03$	$1.835e-04$	$1.015e+04$
		LB/NL/2.0	$\mathbf{1.025e+02}$	$1.021e+03$	$5.849e-05$	$1.021e+04$
TP1	10	SB/L/2.0	$2.096e+01$	$2.129e+01$	$3.641e+00$	$1.309e+02$
		LB/L/2.0	$\mathbf{1.944e+01}$	$2.379e+01$	$2.576e+00$	$1.359e+02$
		SB/NL/1.0	$6.709e+02$	$9.798e+02$	$3.819e+00$	$4.183e+03$
		LB/NL/1.0	$2.841e+03$	$1.542e+04$	$1.177e-01$	$9.001e+04$
	50	SB/L/2.0	$7.509e+05$	$2.821e+05$	$2.696e+05$	$1.721e+06$
		LB/L/2.0	$6.395e+05$	$2.642e+05$	$1.250e+05$	$1.365e+06$
		SB/NL/1.0	$3.279e+03$	$1.538e+04$	$6.913e+01$	$9.015e+04$
		LB/NL/2.0	$\mathbf{3.031e+03}$	$1.541e+04$	$1.832e+01$	$9.016e+04$
	100	SB/L/2.0	$1.411e+07$	$3.222e+06$	$6.315e+06$	$2.110e+07$
		LB/L/2.0	$1.311e+07$	$2.922e+06$	$7.330e+06$	$2.147e+07$
		SB/NL/1.0	$8.806e+04$	$3.708e+05$	$2.359e+02$	$3.069e+06$
		LB/NL/2.0	$\mathbf{1.442e+03}$	$9.031e+03$	$1.621e+02$	$9.060e+04$
TP2	10	SB/L/2.0	$1.025e+01$	$3.122e+00$	$2.479e+00$	$1.808e+01$
		LB/L/2.0	$9.866e+00$	$3.169e+00$	$4.150e+00$	$1.755e+01$
		SB/NL/2.0	$9.233e+00$	$3.253e+00$	$2.985e+00$	$1.845e+01$
		LB/NL/2.0	$\mathbf{7.302e+00}$	$3.347e+00$	$9.950e-01$	$1.792e+01$
	50	SB/L/2.0	$2.751e+02$	$2.650e+01$	$1.985e+02$	$3.284e+02$
		LB/L/2.0	$\mathbf{2.707e+02}$	$2.125e+01$	$2.093e+02$	$3.222e+02$
		SB/NL/2.0	$2.934e+02$	$3.642e+01$	$1.588e+02$	$3.530e+02$
		LB/NL/2.0	$2.793e+02$	$4.174e+01$	$1.668e+02$	$3.598e+02$
	100	**SB/L/2.0**	$\mathbf{7.746e+02}$	$3.730e+01$	$6.324e+02$	$8.545e+02$
		LB/L/2.0	$7.758e+02$	$3.902e+01$	$6.610e+02$	$8.473e+02$
		SB/NL/2.0	$8.544e+02$	$4.779e+01$	$6.957e+02$	$9.499e+02$
		LB/NL/2.0	$8.392e+02$	$5.525e+01$	$6.920e+02$	$9.258e+02$
TP3	10	SB/L/2.0	$3.166e-01$	$1.326e-01$	$9.189e-02$	$6.199e-01$
		LB/L/2.0	$2.896e-01$	$1.180e-01$	$7.101e-02$	$6.369e-01$
		SB/NL/1.0	$1.350e-01$	$1.060e-01$	$7.396e-03$	$5.944e-01$
		LB/NL/1.0	$\mathbf{7.808e-02}$	$5.022e-02$	$0.000e+00$	$2.753e-01$
	50	SB/L/2.0	$2.008e+01$	$4.572e+00$	$1.052e+01$	$4.020e+01$
		LB/L/2.0	$1.853e+01$	$4.383e+00$	$8.683e+00$	$3.427e+01$
		SB/NL/2.0	$5.325e-01$	$8.326e-01$	$3.273e-05$	$4.649e+00$
		LB/NL/2.0	$\mathbf{1.034e-02}$	$1.817e-02$	$2.463e-010$	$8.768e-02$
	100	SB/L/2.0	$1.575e+02$	$2.284e+01$	$9.160e+01$	$2.192e+02$
		LB/L/2.0	$1.566e+02$	$2.039e+01$	$1.150e+02$	$2.062e+02$
		SB/NL/2.0	$2.101e+00$	$1.315e+01$	$5.689e-05$	$9.397e+01$
		LB/NL/2.0	$\mathbf{3.826e-01}$	$4.391e-01$	$2.724e-03$	$3.173e+00$
TP4	10	SB/L/2.0	$1.037e-01$	$1.042e-01$	$2.589e-02$	$9.993e-01$
		LB/L/2.0	$7.962e-02$	$6.811e-02$	$6.170e-03$	$3.781e-01$
		SB/NL/2.0	$1.580e-01$	$3.890e-01$	$2.774e-04$	$1.646e+00$
		LB/NL/2.0	$\mathbf{1.176e-02}$	$1.155e-01$	$9.948e-014$	$1.155e+00$
	50	SB/L/2.0	$7.821e+00$	$6.142e-01$	$6.135e+00$	$9.182e+00$
		LB/L/2.0	$\mathbf{7.675e+00}$	$5.482e-01$	$6.279e+00$	$8.775e+00$
		SB/NL/2.0	$9.738e+00$	$8.693e-01$	$7.080e+00$	$1.122e+01$
		LB/NL/2.0	$9.513e+00$	$7.807e-01$	$7.493e+00$	$1.084e+01$
	100	SB/L/2.0	$1.224e+01$	$4.918e-01$	$1.089e+01$	$1.353e+01$
		LB/L/2.0	$\mathbf{1.214e+01}$	$4.772e-01$	$1.081e+01$	$1.311e+01$
		SB/NL/2.0	$1.416e+01$	$5.042e-01$	$1.309e+01$	$1.521e+01$
		LB/NL/2.0	$1.416e+01$	$4.283e-01$	$1.300e+01$	$1.491e+01$

Table 2.5: Results for the SOBA approach for test problems TP5-TP10 (nonlinear systems).

Problem	Dimension	Algorithm	Mean	StD	Min	Max
TP5	10	SB/L/2.0	$6.312e-03$	$3.557e-03$	$1.035e-03$	$1.855e-02$
		LB/L/2.0	$5.523e-03$	$3.189e-03$	$1.070e-03$	$1.625e-02$
		SB/NL/1.0	$5.155e-04$	$1.912e-03$	$3.978e-07$	$1.538e-02$
		LB/NL/2.0	$\mathbf{4.833e-10}$	$2.108e-09$	$6.556e-015$	$1.221e-08$
TP6	6	**SB/L/2.0**	$\mathbf{3.751e-03}$	$5.196e-03$	$4.176e-05$	$2.578e-02$
		LB/L/2.0	$4.020e-03$	$7.317e-03$	$7.693e-06$	$4.181e-02$
		SB/NL/1.0	$1.514e-01$	$2.579e-01$	$6.463e-09$	$9.859e-01$
		LB/NL/1.0	$9.961e-02$	$2.432e-01$	$0.000e+00$	$9.363e-01$
TP7	5	SB/L/2.0	$1.904e-01$	$1.360e-01$	$2.017e-02$	$6.531e-01$
		LB/L/1.5	$\mathbf{1.741e-01}$	$1.066e-01$	$1.610e-02$	$5.573e-01$
		SB/NL/1.0	$3.078e-01$	$2.387e-01$	$3.518e-02$	$1.179e+00$
		LB/NL/1.0	$2.201e-01$	$1.721e-01$	$6.671e-03$	$7.199e-01$
TP8	8	SB/L/2.0	$3.419e-01$	$2.033e-01$	$2.969e-02$	$1.176e+00$
		LB/L/2.0	$\mathbf{2.926e-01}$	$1.552e-01$	$7.430e-02$	$7.061e-01$
		SB/NL/1.0	$3.167e-01$	$2.444e-01$	$1.055e-02$	$1.085e+00$
		LB/NL/1.0	$3.208e-01$	$2.509e-01$	$5.308e-03$	$9.437e-01$
TP9	10	SB/L/2.0	$8.820e-02$	$6.530e-02$	$8.032e-04$	$3.955e-01$
		LB/L/2.0	$7.770e-02$	$5.849e-02$	$5.178e-03$	$2.526e-01$
		SB/NL/1.0	$4.683e-02$	$4.217e-02$	$6.689e-04$	$2.013e-01$
		LB/NL/2.0	$\mathbf{1.648e-02}$	$1.933e-02$	$1.856e-05$	$9.591e-02$
TP10	20	SB/L/2.0	$2.227e-04$	$2.215e-04$	$1.107e-06$	$8.203e-04$
		LB/L/2.0	$1.236e-04$	$2.127e-04$	$2.809e-07$	$1.199e-03$
		SB/NL/1.0	$2.099e-03$	$6.839e-03$	$1.053e-278$	$5.399e-02$
		LB/NL/1.0	$\mathbf{4.881e-07}$	$2.167e-06$	$2.220e-262$	$1.497e-05$

reported in Table 2.2. Further experimentation was also conducted on the test suite proposed at the special issue on *Scalability of Evolutionary Algorithms and Other Metaheuristics for Large-Scale Optimization Problems* of the *Soft Computing* journal [63]. This test suite consists of 19 problems that include problems from the CEC 2008 challenge, shifted problems, as well as hybrid composition functions.

Four variants of PSO-NBA were considered, namely the SOBA strategy and the MOBA strategy with the LWA, DWA, and PFA schemes. These approaches were combined with the SB and LB scoring schemes under different parameter settings. The complete set of parameter values that were used in the experiments is reported in Table 2.3. In total, there were 36 individual PSO-NBA variants defined by all combinations of these schemes and parameter values.

The experimental evaluation consisted of two stages. In the first stage, the most promising among the different PSO-NBA variants for all test problems was identified.

In the second stage, the distinguished variants were further assessed against different algorithms (either PSO-based or not). The obtained results are presented in detail in the following sections.

2.8.2.1 Assessment of SOBA Strategy

As described in Section 2.8.1.3, the SOBA strategy quantifies the quality of each neighborhood according to a single rank-based score. The SB and LB scoring schemes of Section 2.8.1.1 were employed for this purpose and both were combined with the linear (L) and the nonlinear (NL) approaches described in Section 2.8.1.2, in order to compute the corresponding selection probabilities.

For the linear approach, three different values of selection pressure were considered, namely $s \in \{1.0, 1.5, 2.0\}$. In the nonlinear approach, two different values for the power weight were used, namely $\rho \in \{1.0, 2.0\}$. These combinations result in ten SOBA variants that are henceforth denoted as,

$$\mathsf{X} \ / \ \mathsf{Y} \ / \ \mathsf{Z}$$

where $\mathsf{X} \in \{\mathrm{SB}, \mathrm{LB}\}$ and $\mathsf{Y} \in \{\mathrm{L}, \mathrm{NL}\}$. If Y=L then $\mathsf{Z} \in \{1.0, 1.5,\ 2.0\}$ stands for the selection pressure. If Y=NL then $\mathsf{Z} \in \{1.0, 2.0\}$ (power weight). For example, LB/NL/2.0 stands for the SOBA variant with LocalBest neighborhood scoring and nonlinear probability selection with selection pressure $s = 2.0$.

A total of 100 independent experiments were performed for each test problem and algorithm variant. In all experiments, the available computational budget was equal to $1000 \times n$ function evaluations, where n stands for the problem's dimension. For each experiment, the best solution found by the algorithm as well as its objective value were recorded.

Table 2.4 reports the mean, standard deviation, minimum, and maximum of the 100 solutions' objective values per algorithm and problem instance for TP0-TP4. For presentation purposes, results only for the variants with the best values of selection pressure in the L schemes are reported. The same holds also for the power weights in the NL schemes. Thus, four variants are reported per test problem and dimension. Also, the algorithm with the smallest mean is boldfaced per problem instance.

A quick inspection of Table 2.4 verifies that there is no single variant dominating all the rest. This was anticipated, since the combination of different schemes

and parameter values can equip the algorithm with significantly different exploration/exploitation properties. However, it is clearly identified that some variants habitually exhibit good performance. Specifically, the best SB/L approach (SumBest with linear ranking) was the one with selection pressure $s = 2.0$ in all test problems. This value corresponds to a purely elitist linear ranking (see Section 2.8.1.2). The same holds also for the best LB/L approach (LocalBest with linear ranking), where $s = 2.0$ was again the dominant selection pressure value.

Thus, the experimental evidence suggests that the linear ranking variants of PSO-NBA perform better under high selection elitism. This is a consequence of the algorithm's exploration dynamic, which is increased due to the neighborhood-based budget allocation scheme. The increased exploration is counterbalanced with the intense exploitation imposed through selection elitism.

Elitism was proved to be beneficial also for the nonlinear (NL) selection schemes. Indeed, power weight $\rho = 2.0$ was shown to be superior than $\rho = 1.0$ in 10 out of 15 cases for the SB/NL variants, and in 13 out of 15 cases for the LB/NL variants, as reported in Table 2.4. Obviously, the power selection with $\rho = 2.0$ in Eq. (2.28) provides a significant advantage to neighborhoods with better solution quality by assigning them exponentially higher selection probabilities. Therefore, selection elitism is promoted also in this case. Another interesting observation is that the superiority of $\rho = 1.0$ (observed only in NL-based variants) was restricted in the 10-dimensional instances of the problems with the exception of TP1. This exception can be ascribed to the fact that TP1 becomes an easier problem when its dimension increases [64].

Overall, the LB/NL/2.0 variant was the most successful one, outperforming the rest in 9 out of 15 cases. Also, the LB-based variants dominated the SB-based variants in all cases except one. Finally, the dominant variants were based on the NL scheme in 10 out of 15 cases.

A similar set of experiments was conducted also for TP5-TP10. All SOBA-based variants of PSO-NBA were applied on these problems, using the same experimental setting and analysis as previously. The results for this case are reported in Table 2.5. As can be seen, the LB/NL approaches were dominant in half of the problems and, specifically, the ones of higher dimension (TP5, TP9, and TP10). In the rest, the variants that are based on linear ranking (SB/L and LB/L) exhibited the best performance. Thus, dimensionality was verified to play a crucial role on efficiency.

The aforementioned observations identify clear tendencies and indications regard-

Table 2.6: Results for the MOBA weighted aggregation approaches (LWA and DWA) for test problems TP0-TP4 (standard test suite).

Problem	Dimension	Algorithm	Mean	StD	Min	Max
TP0	10	DW/SB/L/2.0	$3.630e-01$	$2.251e-01$	$5.058e-02$	$9.873e-01$
		LW/LB/L/2.0	$3.405e-01$	$2.452e-01$	$2.447e-02$	$1.493e+00$
		DW/SB/NL/1.0	$7.666e-03$	$1.835e-02$	$7.828e-06$	$1.430e-01$
		DW/LB/NL/2.0	$\mathbf{1.992e-15}$	$1.227e-14$	$1.236e-22$	$1.171e-13$
	50	LW/SB/L/2.0	$3.128e+03$	$4.198e+02$	$2.149e+03$	$4.294e+03$
		LW/LB/L/2.0	$3.052e+03$	$4.056e+02$	$2.008e+03$	$3.832e+03$
		DW/SB/NL/2.0	$\mathbf{6.121e+00}$	$2.768e+01$	$7.749e-03$	$2.363e+02$
		DW/LB/NL/2.0	$1.300e+01$	$1.174e+02$	$1.763e-05$	$1.174e+03$
	100	LW/SB/L/2.0	$2.163e+04$	$1.731e+03$	$1.725e+04$	$2.582e+04$
		DW/LB/L/2.0	$2.136e+04$	$1.947e+03$	$1.598e+04$	$2.582e+04$
		DW/SB/NL/2.0	$1.353e+04$	$7.436e+03$	$4.331e+01$	$2.525e+04$
		DW/LB/NL/2.0	$\mathbf{1.269e+04}$	$7.356e+03$	$1.838e+01$	$2.493e+04$
TP1	10	LW/SB/L/2.0	$3.417e+01$	$2.548e+01$	$6.264e+00$	$1.383e+02$
		LW/LB/L/2.0	$3.057e+01$	$2.338e+01$	$4.785e+00$	$1.446e+02$
		LW/SB/NL/1.0	$2.332e+01$	$3.548e+01$	$4.790e-01$	$2.089e+02$
		LW/LB/NL/1.0	$\mathbf{2.156e+01}$	$3.934e+01$	$1.631e-02$	$2.234e+02$
	50	LW/SB/L/2.0	$1.122e+06$	$2.704e+05$	$5.704e+05$	$1.795e+06$
		LW/LB/L/2.0	$1.109e+06$	$2.918e+05$	$4.950e+05$	$1.713e+06$
		LW/SB/NL/2.0	$7.607e+03$	$1.965e+04$	$1.950e+02$	$9.306e+04$
		DW/LB/NL/2.0	$\mathbf{4.047e+03}$	$1.770e+04$	$2.238e+01$	$9.027e+04$
	100	LW/SB/L/2.0	$1.717e+07$	$2.412e+06$	$1.075e+07$	$2.258e+07$
		LW/LB/L/2.0	$1.662e+07$	$2.122e+06$	$1.085e+07$	$2.109e+07$
		DW/SB/NL/2.0	$1.686e+04$	$2.952e+04$	$5.324e+02$	$1.459e+05$
		LW/LB/NL/2.0	$\mathbf{3.752e+03}$	$9.879e+03$	$3.929e+02$	$9.757e+04$
TP2	10	DW/SB/L/2.0	$1.155e+01$	$2.976e+00$	$4.488e+00$	$2.054e+01$
		LW/LB/L/2.0	$1.066e+01$	$2.611e+00$	$5.273e+00$	$1.881e+01$
		LW/SB/NL/2.0	$1.036e+01$	$3.514e+00$	$3.239e+00$	$2.040e+01$
		LW/LB/NL/2.0	$\mathbf{8.997e+00}$	$3.495e+00$	$2.252e+00$	$1.813e+01$
	50	**LW/SB/L/2.0**	$\mathbf{2.932e+02}$	$1.974e+01$	$2.295e+02$	$3.427e+02$
		LW/LB/L/2.0	$2.943e+02$	$1.777e+01$	$2.411e+02$	$3.426e+02$
		LW/SB/NL/2.0	$3.129e+02$	$2.673e+01$	$1.966e+02$	$3.572e+02$
		LW/LB/NL/2.0	$3.063e+02$	$3.032e+01$	$2.174e+02$	$3.672e+02$
	100	LW/SB/L/2.0	$8.127e+02$	$2.755e+01$	$7.487e+02$	$8.725e+02$
		LW/LB/L/2.0	$\mathbf{8.124e+02}$	$2.738e+01$	$7.312e+02$	$8.610e+02$
		LW/SB/NL/2.0	$8.830e+02$	$3.096e+01$	$7.892e+02$	$9.481e+02$
		LW/LB/NL/2.0	$8.665e+02$	$3.122e+01$	$7.624e+02$	$9.261e+02$
TP3	10	LW/SB/L/2.0	$4.752e-01$	$1.339e-01$	$9.608e-02$	$8.261e-01$
		DW/LB/L/2.0	$4.543e-01$	$1.362e-01$	$1.699e-01$	$8.738e-01$
		LW/SB/NL/2.0	$1.928e-01$	$1.322e-01$	$2.464e-02$	$7.160e-01$
		DW/LB/NL/2.0	$\mathbf{1.038e-01}$	$6.630e-02$	$7.396e-03$	$3.327e-01$
	50	LW/SB/L/2.0	$2.913e+01$	$4.014e+00$	$1.977e+01$	$3.863e+01$
		DW/LB/L/2.0	$2.811e+01$	$3.823e+00$	$1.760e+01$	$3.660e+01$
		DW/SB/NL/2.0	$6.145e-01$	$2.668e+00$	$1.637e-03$	$2.628e+01$
		DW/LB/NL/2.0	$\mathbf{3.970e-01}$	$4.026e-01$	$1.424e-03$	$1.613e+00$
	100	LW/SB/L/2.0	$1.966e+02$	$1.701e+01$	$1.535e+02$	$2.424e+02$
		LW/LB/L/2.0	$1.927e+02$	$1.824e+01$	$1.415e+02$	$2.443e+02$
		DW/SB/NL/2.0	$1.270e+02$	$6.175e+01$	$8.447e-01$	$2.356e+02$
		DW/LB/NL/2.0	$\mathbf{1.101e+02}$	$6.169e+01$	$3.474e+00$	$2.254e+02$
TP4	10	LW/SB/L/2.0	$4.714e-01$	$3.097e-01$	$6.339e-02$	$1.699e+00$
		DW/LB/L/2.0	$4.864e-01$	$3.246e-01$	$1.117e-01$	$1.627e+00$
		LW/SB/NL/2.0	$2.485e-01$	$4.314e-01$	$1.808e-03$	$1.597e+00$
		LW/LB/NL/2.0	$\mathbf{1.106e-01}$	$3.406e-01$	$1.189e-06$	$1.524e+00$
	50	LW/SB/L/2.0	$8.918e+00$	$4.258e-01$	$7.794e+00$	$9.861e+00$
		LW/LB/L/2.0	$\mathbf{8.863e+00}$	$3.804e-01$	$7.517e+00$	$9.573e+00$
		LW/SB/NL/2.0	$1.020e+01$	$4.392e-01$	$8.762e+00$	$1.108e+01$
		LW/LB/NL/2.0	$1.021e+01$	$4.662e-01$	$8.583e+00$	$1.097e+01$
	100	**LW/SB/L/2.0**	$\mathbf{1.306e+01}$	$3.285e-01$	$1.190e+01$	$1.375e+01$
		LW/LB/L/2.0	$1.312e+01$	$3.119e-01$	$1.217e+01$	$1.373e+01$
		LW/SB/NL/2.0	$1.433e+01$	$3.070e-01$	$1.328e+01$	$1.490e+01$
		LW/LB/NL/2.0	$1.428e+01$	$2.843e-01$	$1.352e+01$	$1.485e+01$

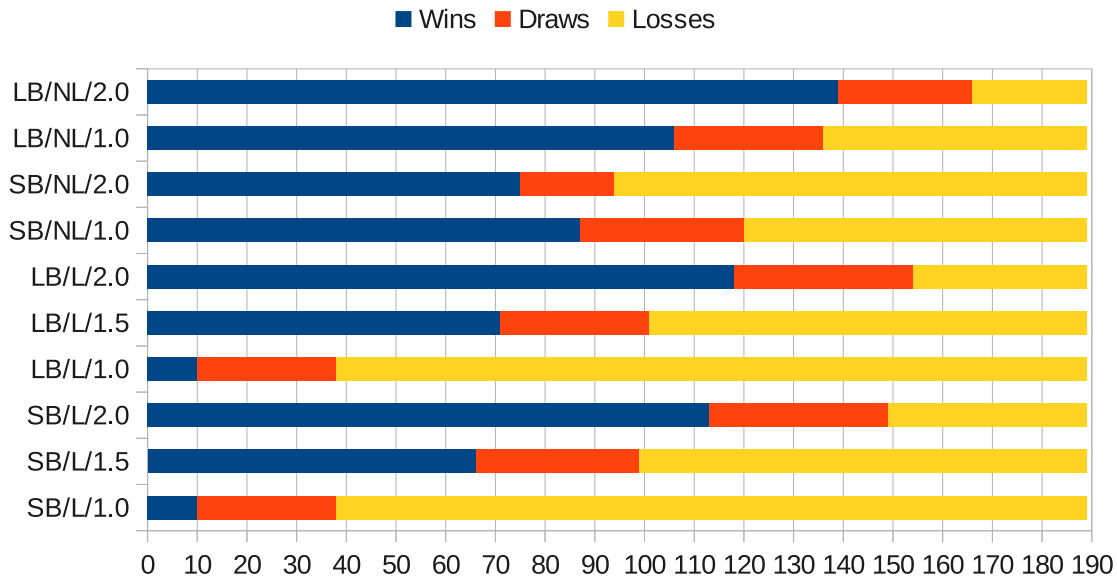


Figure 2.2: Number of wins, draws, and losses for all SOBA variants.

ing the superiority of some schemes. However, further statistical evidence (e.g., the reported standard deviations in the tables) suggested that some of the observed differences might be statistically insignificant. In order to gain more sound insight, pairwise statistical significance tests for all variants were conducted. Specifically, Wilcoxon rank-sum tests at significance level 99% were conducted for each pair of the studied variants (including the ones that are not reported in Tables 2.4 and 2.5). Recall that there were 10 algorithmic variants and 21 different problem instances in total. Thus, each variant had 9 competitors over 21 problem instances, which results in $9 \times 21 = 189$ statistical tests in total per algorithmic variant. For each test where algorithm A was superior than B with statistical significance, a *win* for A and a *loss* for B were counted. If there was no statistical significance between them, a *draw* for both algorithms was counted. The results of these tests are illustrated in Figs. 2.2-2.4.

Figure 2.2 illustrates the number of wins, losses, and draws for all studied variants. The superiority of the LB/NL/2.0 variant is confirmed against the rest. On the other hand, SB/L/1.0 and LB/L/1.0 are evidently the worst combinations as they exhibit the highest number of losses. Besides each individual variant, the four main categories were collectively considered with respect to the combination of quality criterion and selection probability, namely SB/L, LB/L, SB/NL, and LB/NL. For each category, the number of wins, draws, and losses was computed as the sum of the corresponding

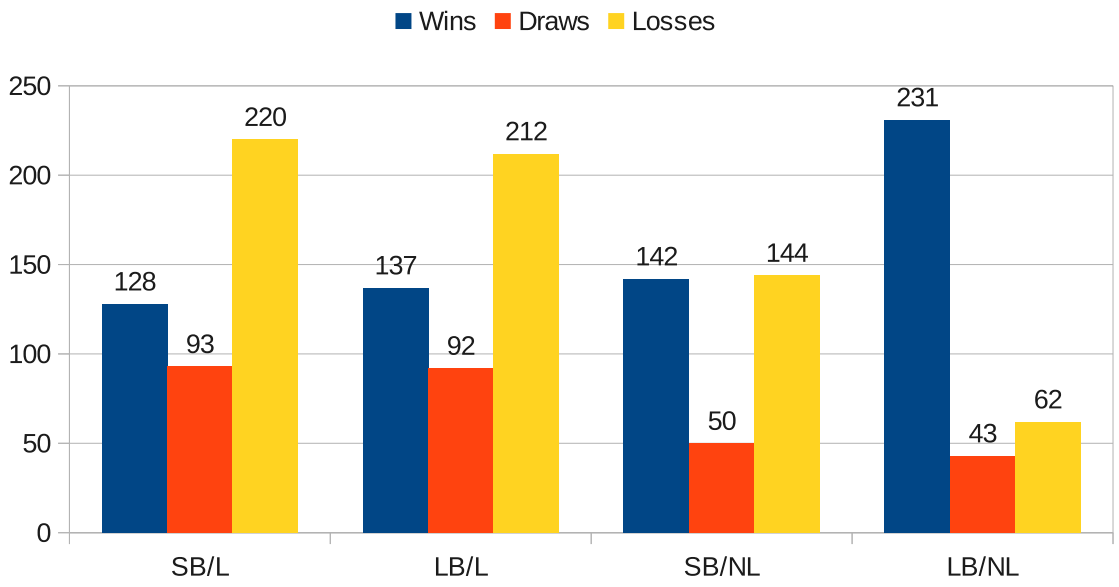


Figure 2.3: Aggregate number of wins, draws, and losses for different combinations of quality criteria and selection probability in SOBA-based variants.

values for all variants that comprise it. The results are reported in Fig. 2.3 where the tendency of LB/NL to produce more efficient variants can be clearly verified. The combinations SB/L and LB/L are the worst (they have the highest number of losses), exhibiting similar behavior between them.

In order to further probe the influence of the selection pressure and the nonlinear weight, Wilcoxon rank-sum tests were performed between pairs of variants that use the same neighborhood scoring approach (SB or LB) and probability selection scheme (L or NL) but different values of selection pressure. These results are reported in Fig. 2.4 and denoted as L/2.0 vs L/1.0, L/2.0 vs L/1.5, and L/1.5 vs L/1.0, where L/s stands for all approaches with linear ranking (L) and selection probability s. Similar analysis was conducted also for the nonlinear approaches (NL) for different values of the power weight. This case is denoted as NL/2.0 vs NL/1.0 in Fig. 2.4. There is an apparently monotonic superiority for the selection pressure values, i.e., $s = 2.0$ prevails $s = 1.5$, which in turn prevails $s = 1.0$. Again, this verifies the benefits of increased elitism in the proposed PSO-NBA variants. The same can be inferred also for the nonlinear weight, since the elitistic choice $\rho = 2.0$ has almost twice as many wins as $\rho = 1.0$.

Table 2.7: Results for the MOBA weighted aggregation approaches (LWA and DWA) for test problems TP5-TP10 (nonlinear systems).

Problem	Dimension	Algorithm	Mean	StD	Min	Max
TP5	10	LW/SB/L/2.0	$2.405e-02$	$8.260e-03$	$6.498e-03$	$5.362e-02$
		LW/LB/L/2.0	$2.183e-02$	$8.869e-03$	$5.610e-03$	$5.908e-02$
		LW/SB/NL/2.0	$4.517e-03$	$5.577e-03$	$2.076e-04$	$3.597e-02$
		DW/LB/NL/2.0	$\mathbf{8.575e-06}$	$6.669e-05$	$1.835e-10$	$6.651e-04$
TP6	6	LW/SB/L/2.0	$9.637e-03$	$8.184e-03$	$5.393e-04$	$3.941e-02$
		LW/LB/L/2.0	$7.083e-03$	$5.662e-03$	$5.466e-04$	$2.818e-02$
		LW/SB/NL/1.0	$4.218e-03$	$5.612e-03$	$1.146e-05$	$2.426e-02$
		LW/LB/NL/2.0	$\mathbf{1.060e-03}$	$3.762e-03$	$2.321e-16$	$2.509e-02$
TP7	5	LW/SB/L/2.0	$1.753e-01$	$1.029e-01$	$2.367e-02$	$5.233e-01$
		LW/LB/L/2.0	$1.780e-01$	$9.777e-02$	$2.886e-02$	$4.870e-01$
		LW/SB/NL/1.0	$1.947e-01$	$1.303e-01$	$7.659e-03$	$7.236e-01$
		LW/LB/NL/1.0	$\mathbf{1.411e-01}$	$1.008e-01$	$6.790e-03$	$5.604e-01$
TP8	8	LW/SB/L/2.0	$4.043e-01$	$1.708e-01$	$7.351e-02$	$9.027e-01$
		LW/LB/L/2.0	$3.387e-01$	$1.687e-01$	$5.565e-02$	$8.826e-01$
		LW/SB/NL/2.0	$3.583e-01$	$1.632e-01$	$5.907e-02$	$8.943e-01$
		LW/LB/NL/2.0	$\mathbf{2.334e-01}$	$1.864e-01$	$7.806e-03$	$7.858e-01$
TP9	10	DW/SB/L/2.0	$1.566e-01$	$8.544e-02$	$1.537e-02$	$4.686e-01$
		LW/LB/L/2.0	$1.401e-01$	$8.279e-02$	$1.056e-02$	$4.241e-01$
		DW/SB/NL/1.0	$8.990e-02$	$7.890e-02$	$1.983e-03$	$5.038e-01$
		DW/LB/NL/2.0	$\mathbf{3.616e-02}$	$6.486e-02$	$4.262e-04$	$5.240e-01$
TP10	20	LW/SB/L/2.0	$1.235e-03$	$1.196e-03$	$1.832e-05$	$5.835e-03$
		DW/LB/L/2.0	$9.602e-04$	$1.107e-03$	$1.039e-05$	$7.384e-03$
		LW/SB/NL/1.0	$4.581e-04$	$1.152e-03$	$1.776e-15$	$9.323e-03$
		DW/LB/NL/1.0	$\mathbf{4.877e-06}$	$1.583e-05$	$4.878e-134$	$1.100e-04$

2.8.2.2 Assessment of MOBA Strategy

The MOBA strategy assesses each neighborhood by using two criteria instead of one, as described in Section 2.8.1.4. The first criterion is solution quality while the second one is diversity of the best positions involved in the neighborhood. Two different ways to cope with the multi-objective scoring were considered, namely weighted aggregation (LWA and DWA) and the Pareto front (PFA) approach. For notation purposes, the formalism of Section 2.8.2.1 was extended as follows:

$$\mathtt{A} \ / \ \mathtt{X} \ / \ \mathtt{Y} \ / \ \mathtt{Z},$$

where $\mathtt{A}$ takes the values DW (for DWA), LW (for LWA), and PF (for PFA), while $\mathtt{X} \in \{\mathrm{SB}, \mathrm{LB}\}$ and $\mathtt{Y} \in \{\mathrm{L}, \mathrm{NL}\}$. Experimental results for all MOBA approaches are reported and analyzed in the following sections.

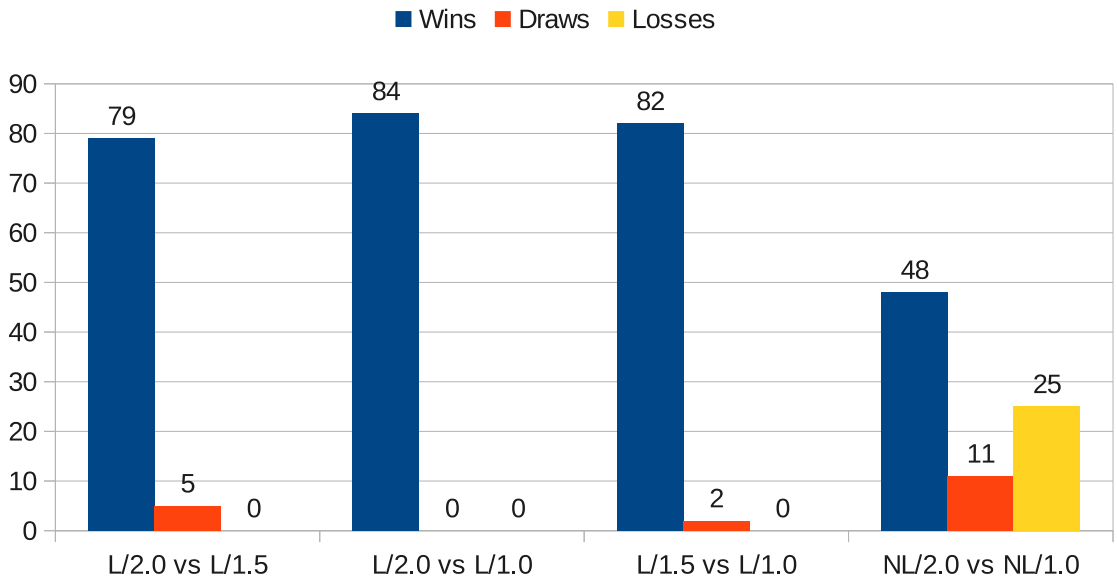


Figure 2.4: Aggregate number of wins, draws, and losses for pairs of variants with the same selection probability scheme (L or NL) but different parameters in SOBA-based variants.

Results for Weighted Aggregation Approaches

Initially, the weighted aggregation approach was studied, which is based on the conversion of the multi-objective scoring to a single-objective one as described in Section 2.8.1.4. Both the linear weighted aggregation (LWA) and the dynamic weighted aggregation (DWA) approaches were considered. The neighborhoods' quality was determined based on the SB and LB schemes, while diversity was quantified through the AD scheme (see Section 2.8.1.1). The L and NL selection probability approaches (see Section 2.8.1.2) were combined with the aforementioned schemes. The parameter setting of Table 2.3 was used also here, resulting in ten LWA and ten DWA variants.

All variants were applied on all instances of test problems TP0-TP10. The best variants were distinguished per problem instance on the basis of the average best solution value within the prespecified computational budget over 100 experiments. The results for TP0-TP4 are reported in Table 2.6 and for TP5-TP10 in Table 2.7.

Table 2.6 offers interesting evidence. First, the LB-based variants clearly dominate the SB-based variants in 12 out of 15 cases. Moreover, the NL/2.0 approaches performed better in 11 out of 15 cases, while L/2.0 approaches were the best in the

Table 2.8: Results for the PFA approach of MOBA strategy for test problems TP0-TP10.

Problem	Dimension	Algorithm	Mean	StD	Min	Max
TP0	10	**PF/SB/2**	$\mathbf{4.956e-03}$	$9.674e-03$	$3.582e-05$	$7.846e-02$
		PF/LB/2	$7.788e-03$	$8.127e-03$	$1.266e-04$	$4.324e-02$
	50	**PF/SB/2**	$\mathbf{1.059e+01}$	$5.525e+00$	$1.802e+00$	$2.796e+01$
		PF/LB/2	$2.527e+01$	$1.220e+01$	$5.353e+00$	$7.520e+01$
	100	**PF/SB/2**	$\mathbf{1.489e+02}$	$6.499e+01$	$5.378e+01$	$3.799e+02$
		PF/LB/2	$2.524e+02$	$9.412e+01$	$1.171e+02$	$7.451e+02$
TP1	10	**PF/SB/5**	$\mathbf{1.346e+01}$	$1.532e+01$	$2.117e+00$	$8.625e+01$
		PF/LB/3	$1.350e+01$	$2.095e+01$	$2.389e-01$	$1.138e+02$
	50	PF/SB/3	$4.120e+03$	$1.259e+04$	$2.363e+02$	$9.155e+04$
		PF/LB/3	$\mathbf{3.651e+03}$	$9.885e+03$	$6.327e+02$	$9.083e+04$
	100	**PF/SB/2**	$\mathbf{3.076e+04}$	$2.883e+04$	$4.238e+03$	$1.160e+05$
		PF/LB/2	$4.517e+04$	$3.261e+04$	$1.270e+04$	$1.586e+05$
TP2	10	PF/SB/3	$8.552e+00$	$3.468e+00$	$9.954e-01$	$1.994e+01$
		PF/LB/5	$\mathbf{8.224e+00}$	$3.116e+00$	$2.985e+00$	$1.866e+01$
	50	**PF/SB/5**	$\mathbf{1.439e+02}$	$2.614e+01$	$9.102e+01$	$2.098e+02$
		PF/LB/5	$1.463e+02$	$2.943e+01$	$7.830e+01$	$1.990e+02$
	100	**PF/SB/5**	$\mathbf{3.814e+02}$	$4.841e+01$	$2.734e+02$	$5.077e+02$
		PF/LB/5	$4.096e+02$	$6.149e+01$	$2.753e+02$	$5.311e+02$
TP3	10	**PF/SB/2**	$\mathbf{2.142e-01}$	$1.265e-01$	$4.135e-02$	$6.241e-01$
		PF/LB/3	$2.288e-01$	$1.266e-01$	$3.214e-02$	$5.560e-01$
	50	**PF/SB/2**	$\mathbf{1.092e+00}$	$5.068e-02$	$1.011e+00$	$1.296e+00$
		PF/LB/2	$1.205e+00$	$7.270e-02$	$1.070e+00$	$1.388e+00$
	100	**PF/SB/2**	$\mathbf{2.224e+00}$	$6.657e-01$	$1.311e+00$	$5.931e+00$
		PF/LB/2	$3.280e+00$	$8.334e-01$	$1.718e+00$	$6.719e+00$
TP4	10	**PF/SB/2**	$\mathbf{3.168e-02}$	$6.641e-02$	$1.923e-03$	$6.358e-01$
		PF/LB/2	$3.543e-02$	$3.993e-02$	$5.304e-03$	$3.435e-01$
	50	**PF/SB/2**	$\mathbf{2.265e+00}$	$5.141e-01$	$7.843e-01$	$3.532e+00$
		PF/LB/2	$2.308e+00$	$4.078e-01$	$1.226e+00$	$3.431e+00$
	100	PF/SB/2	$3.806e+00$	$5.387e-01$	$2.812e+00$	$5.817e+00$
		PF/LB/2	$\mathbf{3.761e+00}$	$4.558e-01$	$2.879e+00$	$5.400e+00$
TP5	10	**PF/SB/2**	$\mathbf{2.100e-03}$	$1.444e-03$	$5.228e-05$	$7.846e-03$
		PF/LB/2	$3.202e-03$	$2.261e-03$	$7.199e-05$	$1.527e-02$
TP6	6	PF/SB/5	$5.067e-03$	$1.053e-02$	$3.121e-06$	$6.372e-02$
		PF/LB/3	$\mathbf{1.329e-03}$	$2.891e-03$	$7.233e-07$	$1.680e-02$
TP7	5	PF/SB/3	$1.698e-01$	$1.113e-01$	$9.561e-03$	$5.112e-01$
		PF/LB/5	$\mathbf{1.395e-01}$	$9.246e-02$	$9.646e-03$	$4.269e-01$
TP8	8	PF/SB/3	$3.098e-01$	$2.272e-01$	$2.095e-02$	$1.268e+00$
		PF/LB/2	$\mathbf{2.539e-01}$	$1.872e-01$	$1.803e-02$	$8.879e-01$
TP9	10	PF/SB/2	$4.831e-02$	$5.041e-02$	$3.136e-04$	$2.945e-01$
		PF/LB/2	$\mathbf{4.070e-02}$	$3.635e-02$	$1.515e-03$	$2.136e-01$
TP10	20	PF/SB/5	$1.428e-04$	$3.736e-04$	$5.837e-15$	$3.225e-03$
		PF/LB/5	$\mathbf{1.364e-04}$	$2.927e-04$	$2.037e-11$	$1.477e-03$

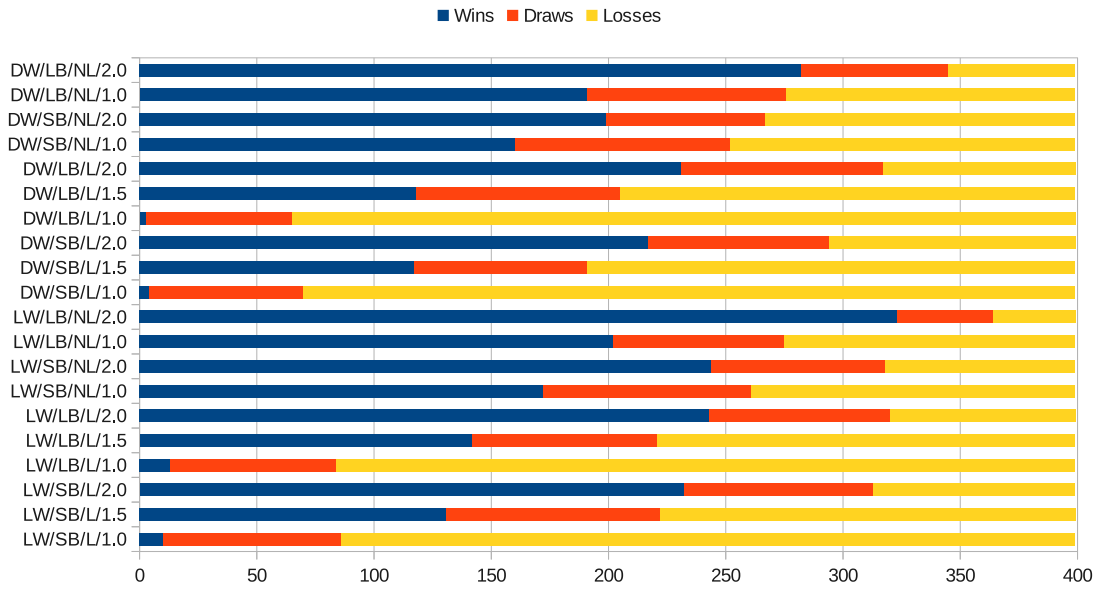


Figure 2.5: Number of wins, draws, and losses for the MOBA-based variants LWA and DWA.

rest 4 cases. These findings are aligned with the ones for the SOBA strategy in the previous section.

However, the picture becomes complicated when DWA is considered against LWA. In Table 2.6, there is no clear tendency for either of the two approaches. In fact, DWA was superior in 7 out of 15 cases, while LWA appeared to be a better choice for the rest 8 problem instances. Therefore, no clear conclusion can be derived from these results. Yet, it is ascertained that DWA performs better when combined with LB/NL approaches. On the other hand, the LWA approaches do not favor a single combination. Indeed, LW/LB/NL appears 4 times in Table 2.6, while LW/LB/L and LW/SB/L appear 2 times each.

The second set of test problems offers similar conclusions. As can be seen in Table 2.7, DWA is distinguished in half of the cases and LWA in the rest. However, this time it can be seen that all variants are based on the LB/NL combination. Interestingly, DWA dominates in the three high-dimensional problems (TP5, TP9, and TP10), while LWA is distinguished in the lower-dimensional cases.

In order to facilitate comparisons between different variants, Wilcoxon rank-sum tests were conducted among all LWA and DWA variants at significance level 99%, similarly to the SOBA approach. Figure 2.5 illustrates the number of wins, draws, and

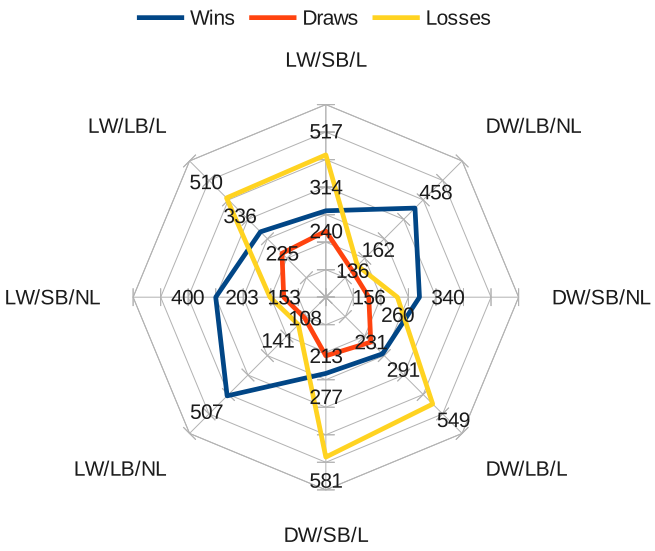


Figure 2.6: Aggregate number of wins, draws, and losses for different combinations of quality criteria and selection probability in LWA and DWA approaches.

losses per algorithmic variant. As can be seen, there is an indisputable predominance of the LB/NL/2.0 variants both for DWA and LWA, with the later exhibiting the highest number of wins. This is in line with the evidence in Tables 2.6 and 2.7.

Similarly to the SOBA approaches, the four main categories SB/L, LB/L, SB/NL, and LB/NL were also considered, both for LWA and DWA. For each category, the aggregate number of wins, draws, and losses was computed. The results are reported in the net chart of Fig. 2.6, where the previous findings can be clearly verified. Finally, L-based and NL-based approaches for both LWA and DWA were compared with different parameter values. The results are illustrated in Fig. 2.7 where the monotonic decline of performance can be verified as selection pressure decreases (suppressing elitism) as well as the superiority of higher power weight values in NL-based variants.

Results for Pareto Front Approach

The PFA approach uses a radically different mechanism for neighborhood scoring than the previous SOBA and MOBA approaches. Specifically, each neighborhood is assessed with two distinct criteria, namely solution quality and diversity. These criteria are not combined as in the weighted aggregation approaches. Instead, they are used

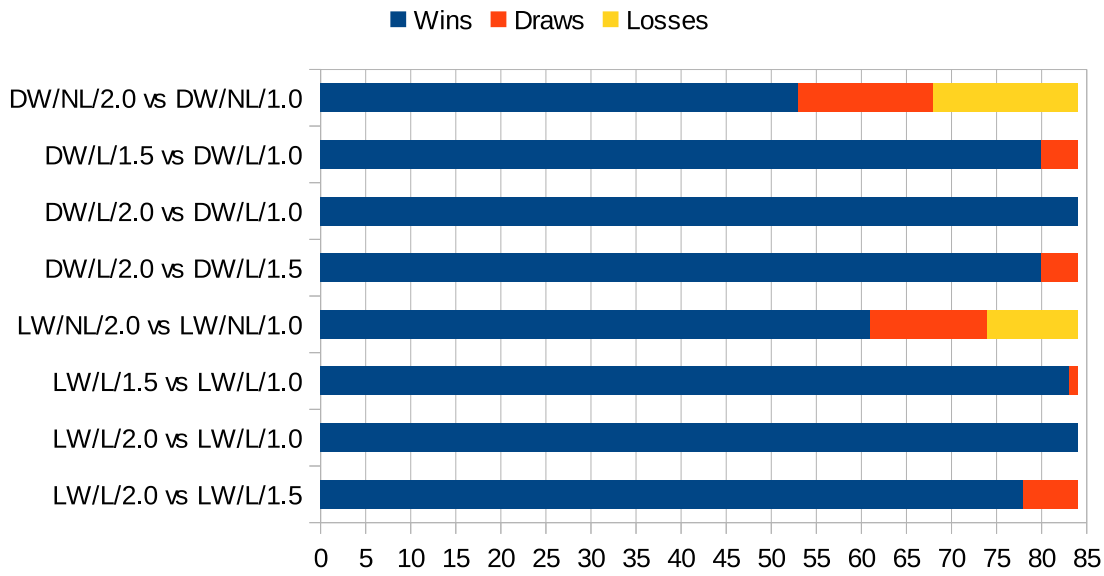


Figure 2.7: Aggregate number of wins, draws, and losses for pairs of LWA and DWA variants with identical selection probability scheme (L or NL), but different parameters.

for vectorial comparisons between neighborhoods in the sense of Pareto dominance. The comparisons are conducted through a tournament selection mechanism in order to avoid search stagnation.

The tournament size is usually an influential factor in tournament selection. For this reason, three different values were used, i.e., $T = N/2, N/3, N/5$, where N is the swarm size. The combinations of the neighborhood scoring schemes with the different tournament sizes resulted in six PFA variants. Each combination is denoted with the notation,

$$\texttt{PF / X / TS},$$

where $\texttt{X} \in \{\text{SB}, \text{LB}\}$ and $\texttt{TS} \in \{2, 3, 5\}$. For example, PF/LB/2 stands for the PFA variant with LB neighborhood scoring and tournament size $T = N/2$, whereas PF/SB/5 denotes the PFA variant with SB neighborhood scoring mode and tournament size $T = N/5$. The experimental setting was identical to the previous cases of MOBA and SOBA strategies. Table 2.8 reports the best solution values for each problem instance, averaged over 100 experiments. For presentation compactness reasons, only the best SB-based and LB-based variants are reported per case.

In the upper part of Table 2.8 (problems TP0-TP4), the variant SB/2 is distin-

Table 2.9: Aggregate numbers of wins, draws, and losses for all PSO-NBA variants for all test problems.

Strategy	Algorithm	Wins	Draws	Losses
SOBA	SB/L/1.0	25	80	630
	SB/L/1.5	277	110	348
	SB/L/2.0	438	97	200
	SB/NL/1.0	329	149	257
	SB/NL/2.0	301	84	350
	LB/L/1.0	26	81	628
	LB/L/1.5	299	106	330
	LB/L/2.0	450	109	176
	LB/NL/1.0	398	128	209
	LB/NL/2.0	**544**	**87**	**104**
MOBA	LW/SB/L/1.0	43	102	590
	LW/SB/L/1.5	189	113	433
	LW/SB/L/2.0	327	105	303
	LW/SB/NL/1.0	252	148	335
	LW/SB/NL/2.0	357	137	241
	LW/LB/L/1.0	44	102	589
	LW/LB/L/1.5	203	102	430
	LW/LB/L/2.0	340	115	280
	LW/LB/NL/1.0	306	128	301
	LW/LB/NL/2.0	**498**	**105**	**132**
	DW/SB/L/1.0	32	90	613
	DW/SB/L/1.5	170	94	471
	DW/SB/L/2.0	303	106	326
	DW/SB/NL/1.0	240	160	335
	DW/SB/NL/2.0	307	123	305
	DW/LB/L/1.0	30	85	620
	DW/LB/L/1.5	171	112	452
	DW/LB/L/2.0	325	113	297
	DW/LB/NL/1.0	292	152	291
	DW/LB/NL/2.0	**454**	**120**	**161**
	PF/SB/2	555	114	66
	PF/SB/3	542	119	74
	PF/SB/5	508	105	122
	PF/LB/2	**575**	**88**	**72**
	PF/LB/3	566	95	74
	PF/LB/5	537	90	108

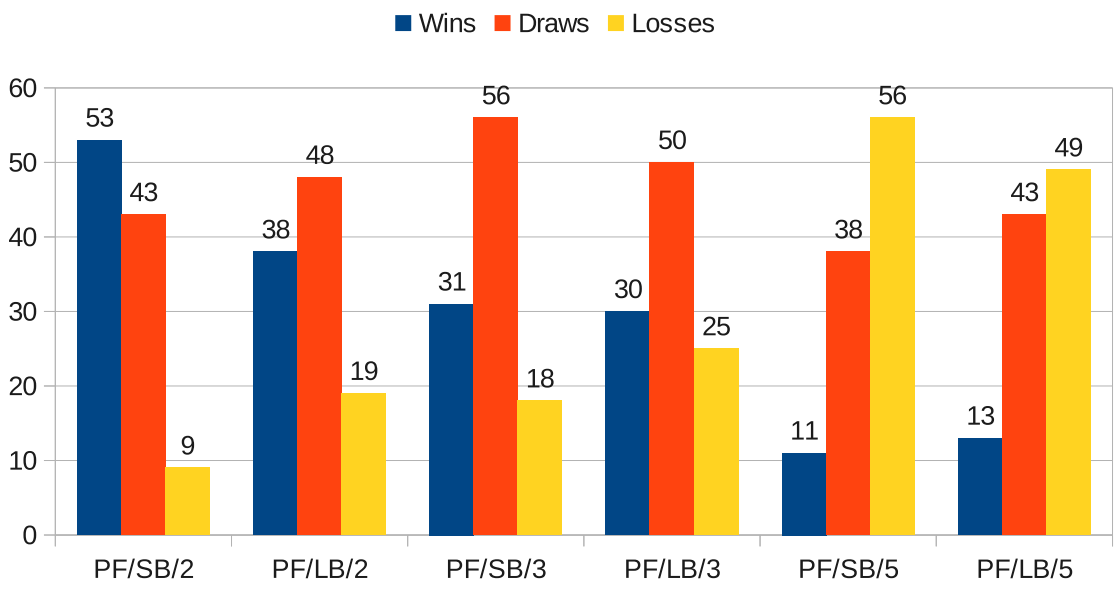


Figure 2.8: Number of wins, draws, and losses for the PFA variant of MOBA strategy.

guished in 12 out of 15 problem instances. Also, the TS $= 2$ (i.e., $T = N/2$) case appeared as the most efficient in 10 out of 15 cases. The SB approach implies reduced elitism than LB. On the other hand, smaller values of TS correspond to higher tournament sizes T, which promote elitism since the overall best individual has higher probability of being selected. Thus, it may be reasonable to assume that the lower values of TS counterbalance the selection of the SB approach in terms of elitism.

The picture changes in the lower part of Table 2.8 (TP5-TP10). In these cases, the LB-based approaches outperform the rest in all but one problem. Also, the tournament size seems to be problem-dependent. This evidence suggests that elitism plays significant role in TP5-TP10. This is in accordance with previous findings for the rest of SOBA and MOBA variants.

Following the analysis of previous sections, Wilcoxon rank-sum tests among all PFA variants were conducted. Figure 2.8 illustrates the number of wins, draws, and losses per variant. The statistical evidence clearly shows a monotonic decline of performance as TS increases. Also, the SB-based variants seem to prevail especially for lower TS values.

In order to further explore the impact of tournament size, Wilcoxon rank-sum tests were conducted between variants that use identical scoring schemes but different tournament sizes. Then, for each tournament size the corresponding number of wins,

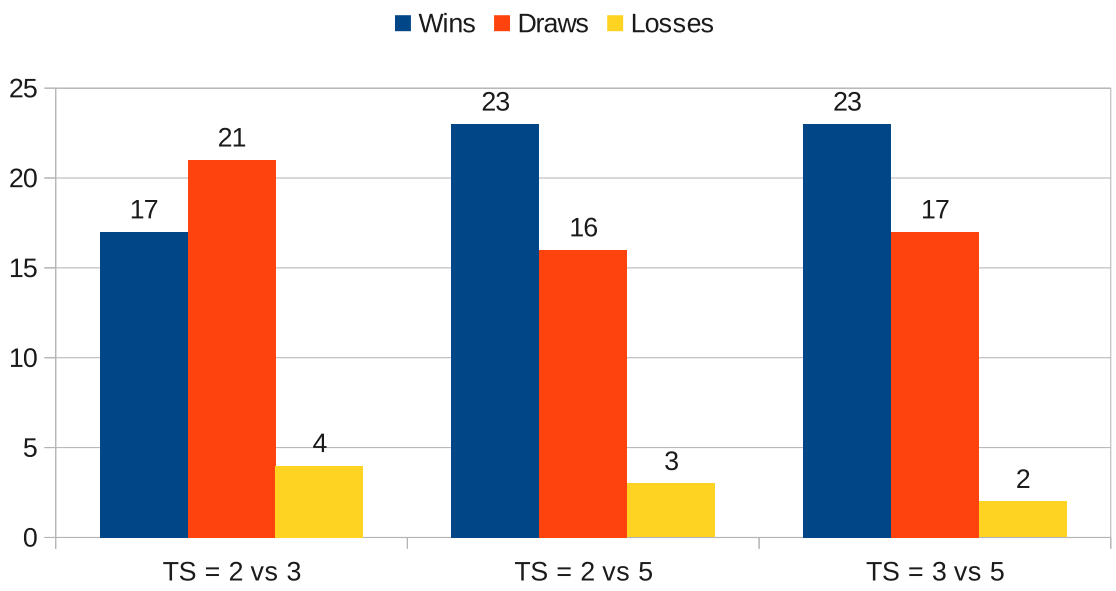


Figure 2.9: Aggregate number of wins, draws, and losses for different tournament sizes in PFA variants.

draws, and losses was summed up. The results are reported in Fig. 2.9. Clearly, $\mathtt{TS} = 2$ is the best choice, verifying the monotonic decline as its value increases. Therefore, smaller tournament sizes produce less efficient approaches evidently due to reduced elitism.

2.8.2.3 Comparative Results

In the previous sections, each strategy of the proposed PSO-NBA approach was individually studied. In this section, comparisons among all the presented variants are reported. This includes the SOBA approaches as well as all MOBA approaches (LWA, DWA, and PFA). The comparisons were all based on test problems TP0-TP10. Moreover, results from comparisons with other algorithms are reported.

First, all PSO-NBA approaches among them for all test problems were compared. Each pairwise comparison was based on Wilcoxon rank-sum tests at significance level of 99%. For each algorithm, its aggregate number of wins, draws, and losses was recorded. These results are reported in Table 2.9.

Two interesting observations can be made in Table 2.9. First, it can be easily noticed that the LB/NL/2.0 approach prevails both in SOBA and MOBA strategies

Table 2.10: Computational budgets for GA, DE, MONS, PSO, and ASY [1], in test problems TP5-TP10.

Problem	Comp. Budget
TP5	15×10^4
TP6	6×10^4
TP7	25×10^4
TP8	50×10^4
TP9	15×10^4
TP10	15×10^4

(boldfaced entries in Table 2.9). This was also pointed out in the previous sections. Secondly, it can be seen that all PFA approaches exhibit a remarkably high number of wins and small number of losses. This is a new evidence, which indicates that using diversity along with solution quality for neighborhood rating can be beneficial for the algorithm.

According to Table 2.9, the variants LB/NL/2.0 and PF/LB/2 exhibited the highest number of wins for the SOBA and the MOBA strategies, respectively. These two variants were considered for comparisons with different PSO-based algorithms under the experimental setting of Table 2.3. More specifically, they were compared against the standard sychronous Particle Swarm Optimization (PSO) algorithm as well as its asychronous version (ASY) presented in [1]. The corresponding results are reported in Tables 2.11 and 2.12 for the standard test suite and the nonlinear systems, respectively.

The reported results reveal an apparent superiority of the PSO-NBA algorithm (by orders of magnitude) against PSO and ASY for all problems and dimensions. More specifically, the SOBA-based variant LB/NL/2.0 surmounts all other in 10 out of 15 problem instances of the standard test suite (Table 2.11), while the MOBA-based variant PF/L/2 is superior in the rest 5 cases. In Table 2.12, similar results for the nonlinear systems are reported, with the two variants being distinguished in 3 cases each. Finally, it can be inferred that the SOBA strategy dominates in the three high-dimensional problems (TP5, TP9, and TP10), while the MOBA one is better in the lower-dimensional cases.

The four distinguished PSO-NBA approaches from Table 2.9 were used for further

Table 2.11: Comparative results of PSO-NBA with PSO-based variants for test problems TP0-TP4 (standard test suite).

Problem	Dimension	Algorithm	Mean	StD
TP0	10	PSO	$3.608e+00$	$2.038e+00$
		ASY	$2.067e+00$	$1.091e+00$
		PF/LB/2	$7.788e-03$	$8.127e-03$
		LB/NL/2.0	$\mathbf{9.406e-26}$	$8.806e-25$
	50	PSO	$8.801e+03$	$9.596e+02$
		ASY	$7.162e+03$	$7.755e+02$
		PF/LB/2	$2.527e+01$	$1.220e+01$
		LB/NL/2.0	$\mathbf{3.116e-08}$	$1.332e-07$
	100	PSO	$4.808e+04$	$2.913e+03$
		ASY	$3.876e+04$	$2.494e+03$
		PF/LB/2	$2.524e+02$	$9.412e+01$
		LB/NL/2.0	$\mathbf{1.025e+02}$	$1.021e+03$
TP1	10	PSO	$2.369e+03$	$1.790e+03$
		ASY	$1.270e+03$	$8.705e+02$
		PF/LB/2	$\mathbf{2.035e+01}$	$3.011e+01$
		LB/NL/2.0	$5.330e+03$	$2.072e+04$
	50	PSO	$7.382e+08$	$1.569e+08$
		ASY	$5.187e+08$	$1.214e+08$
		PF/LB/2	$3.685e+03$	$1.269e+04$
		LB/NL/2.0	$\mathbf{3.031e+03}$	$1.541e+04$
	100	PSO	$7.760e+09$	$1.135e+09$
		ASY	$5.573e+09$	$7.742e+08$
		PF/LB/2	$4.517e+04$	$3.261e+04$
		LB/NL/2.0	$\mathbf{1.442e+03}$	$9.031e+03$
TP2	10	PSO	$1.587e+01$	$3.773e+00$
		ASY	$1.563e+01$	$3.977e+00$
		PF/LB/2	$8.306e+00$	$3.390e+00$
		LB/NL/2.0	$\mathbf{7.302e+00}$	$3.347e+00$
	50	PSO	$3.508e+02$	$2.098e+01$
		ASY	$3.330e+02$	$1.751e+01$
		PF/LB/2	$\mathbf{1.601e+02}$	$3.212e+01$
		LB/NL/2.0	$2.793e+02$	$4.174e+01$
	100	PSO	$9.289e+02$	$3.046e+01$
		ASY	$8.877e+02$	$3.119e+01$
		PF/LB/2	$\mathbf{4.273e+02}$	$6.944e+01$
		LB/NL/2.0	$8.392e+02$	$5.525e+01$
TP3	10	PSO	$8.536e-01$	$1.173e-01$
		ASY	$7.369e-01$	$1.598e-01$
		PF/LB/2	$2.375e-01$	$1.306e-01$
		LB/NL/2.0	$\mathbf{8.893e-02}$	$5.447e-02$
	50	PSO	$8.095e+01$	$9.016e+00$
		ASY	$6.425e+01$	$7.925e+00$
		PF/LB/2	$1.205e+00$	$7.270e-02$
		LB/NL/2.0	$\mathbf{1.034e-02}$	$1.817e-02$
	100	PSO	$4.331e+02$	$2.505e+01$
		ASY	$3.520e+02$	$2.312e+01$
		PF/LB/2	$3.280e+00$	$8.334e-01$
		LB/NL/2.0	$\mathbf{3.826e-01}$	$4.391e-01$
TP4	10	PSO	$2.059e+00$	$4.495e-01$
		ASY	$1.706e+00$	$5.198e-01$
		PF/LB/2	$3.543e-02$	$3.993e-02$
		LB/NL/2.0	$\mathbf{1.176e-02}$	$1.155e-01$
	50	PSO	$1.370e+01$	$4.042e-01$
		ASY	$1.284e+01$	$4.087e-01$
		PF/LB/2	$\mathbf{2.308e+00}$	$4.078e-01$
		LB/NL/2.0	$9.513e+00$	$7.807e-01$
	100	PSO	$1.730e+01$	$2.400e-01$
		ASY	$1.636e+01$	$2.488e-01$
		PF/LB/2	$\mathbf{3.761e+00}$	$4.558e-01$
		LB/NL/2.0	$1.416e+01$	$4.283e-01$

Table 2.12: Comparative results of PSO-NBA with PSO-based variants for test problems TP5-TP10 (nonlinear systems).

Problem	Dimension	Algorithm	Mean	StD
TP5	10	PSO	$6.921e-02$	$1.7539e-02$
		ASY	$6.214e-02$	$1.755e-02$
		PF/LB/2	$3.202e-03$	$2.261e-03$
		LB/NL/2.0	$\mathbf{4.833e-10}$	$2.108e-09$
TP6	6	PSO	$2.765e-02$	$2.482e-02$
		ASY	$2.081e-02$	$1.388e-02$
		PF/LB/2	$\mathbf{6.827e-03}$	$2.914e-02$
		LB/NL/2.0	$1.908e-01$	$3.177e-01$
TP7	5	PSO	$2.640e-01$	$1.273e-01$
		ASY	$2.192e-01$	$1.215e-01$
		PF/LB/2	$\mathbf{1.402e-01}$	$1.006e-01$
		LB/NL/2.0	$2.904e-01$	$2.285e-01$
TP8	8	PSO	$6.120e-01$	$2.050e-01$
		ASY	$5.396e-01$	$1.974e-01$
		PF/LB/2	$\mathbf{2.539e-01}$	$1.872e-01$
		LB/NL/2.0	$3.870e-01$	$4.366e-01$
TP9	10	PSO	$2.980e-01$	$1.565e-01$
		ASY	$2.391e-01$	$1.317e-01$
		PF/LB/2	$4.070e-02$	$3.635e-02$
		LB/NL/2.0	$\mathbf{1.648e-02}$	$1.933e-02$
TP10	20	PSO	$4.617e-03$	$4.544e-03$
		ASY	$3.377e-03$	$2.518e-03$
		PF/LB/2	$3.482e-04$	$1.243e-03$
		LB/NL/2.0	$\mathbf{1.576e-06}$	$6.868e-06$

comparisons with different algorithms on TP5-TP10. For this purpose, the results for a steady state Genetic Algorithm (GA), Differential Evolution (DE), and the multi-objective MONS approach were adopted that were reported in the recent study [1]. The experimental setting that was used in [1] assumed higher computational budgets than the one used in this study. Thus, the experiments on TP5-TP10 for the distinguished PSO-NBA approaches were repeated with the new computational budget, in order to obtain comparable results. The computational budgets adopted from [1] are reported in Table 2.10. For the PSO-NBA variants, the parameters were identical to the ones used in previous sections and reported in Table 2.3 without any further fine-tuning.

All results are reported in Table 2.13. The best performance among the other algorithms as well as among PSO-NBA variants is boldfaced. The reported experimental evidence offer some useful conclusions. First, it can be seen that the MOBA approaches of PSO-NBA outperformed the SOBA one with the exception of TP10.

Table 2.13: Comparative results of PSO-NBA with different algorithms. The results of MONS, GA, DE are adopted from [1].

		TP5	TP6	TP7	TP8	TP9	TP10
MONS	Mean	$1.80e+00$	$1.00e-01$	$6.00e-01$	$1.10e+00$	$2.00e-01$	$2.00e-02$
	StD	–	–	–	–	–	–
GA	Mean	$1.01e-01$	$1.29e-02$	$9.57e-01$	$1.03e+00$	$4.53e-01$	$\mathbf{2.10e-06}$
	StD	$6.21e-02$	$2.29e-02$	$6.78e-01$	$5.50e-01$	$4.74e-01$	$1.00e-06$
DE	Mean	$\mathbf{1.44e-16}$	$\mathbf{1.29e-03}$	$\mathbf{1.01e-02}$	$\mathbf{1.29e-16}$	$\mathbf{5.20e-04}$	$6.37e-03$
	StD	$1.90e-18$	$2.47e-03$	$9.07e-04$	$5.87e-17$	$1.92e-04$	$3.74e-03$
LB/NL/2.0 (SOBA)	Mean	$1.44e-16$	$5.24e-02$	$1.46e-01$	$1.85e-01$	$2.57e-02$	$\mathbf{8.53e-10}$
	StD	$6.74e-19$	$1.66e-01$	$1.20e-01$	$1.97e-01$	$2.49e-01$	$6.84e-09$
LW/LB/NL/2.0 (MOBA)	Mean	$\mathbf{1.44e-16}$	$1.03e-05$	$\mathbf{3.31e-02}$	$\mathbf{2.09e-02}$	$4.33e-03$	$6.43e-07$
	StD	$0.00e+00$	$6.96e-05$	$1.69e-02$	$2.39e-02$	$5.13e-03$	$2.53e-06$
DW/LB/NL/2.0 (MOBA)	Mean	$1.44e-16$	$\mathbf{2.80e-06}$	$3.57e-02$	$3.40e-02$	$\mathbf{2.82e-03}$	$7.38e-09$
	StD	$5.20e-19$	$1.90e-05$	$1.94e-02$	$3.39e-02$	$4.25e-03$	$3.07e-08$
PF/LB/2 (MOBA)	Mean	$\mathbf{1.44e-16}$	$2.92e-06$	$3.79e-02$	$6.04e-02$	$3.48e-03$	$1.32e-05$
	StD	$0.00e+00$	$1.45e-05$	$2.21e-02$	$8.25e-02$	$5.77e-03$	$3.34e-05$

Secondly, the MOBA approaches performed better (by orders of magnitude) than MONS and GA in most of the problems.

Finally, it can be seen that PSO-NBA could outperform DE, which was the best algorithm among the rest, in half of the problems. It shall be taken into consideration that the results of PSO-NBA were received with the same parameters that were used in the default experimental setting without any further fine-tuning for the specific problems and computational budgets, while population sizes for the rest of the algorithms were fine-tuned per case.

2.8.2.4 Further Experiments

The PSO-NBA algorithm was further assessed on a test suite of 19 problems that was proposed as a benchmark at the special issue on *Scalability of Evolutionary Algorithms and Other Metaheuristics for Large-Scale Optimization Problems* of the *Soft Computing* journal [63]. These problems will be henceforth denoted as SC-TP0 - SC-TP18. The problems SC-TP0 - SC-TP5 belong to the CEC 2008 test suite, SC-TP6 - SC-TP10 are shifted problems, and SC-TP11 - SC-TP18 are hybrid composition functions. Their definitions as well as source codes can be obtained through online sources [1]. Com-

[1] `http://sci2s.ugr.es/eamhco/testfunctions-SOCO.pdf`

Table 2.14: Comparative results of PSO-NBA with different algorithms on test problems SC-TP0–SC-TP18.

Dim.	Algorithm	SC-TP0	SC-TP1	SC-TP2	SC-TP3	SC-TP4	SC-TP5	SC-TP6	SC-TP7	SC-TP8	SC-TP9
50	EvoPROpt	$1.22e-02$	$3.71e-01$	$1.12e+02$	$4.96e-02$	$5.13e-02$	$6.85e-03$	$2.63e-02$	$2.08e-02$	$8.02e+00$	$4.80e-02$
	SPSO2011	$0.00e+00$	$4.43e+01$	$6.36e+01$	$1.53e+02$	$5.32e-03$	$2.18e+00$	$1.18e+01$	$5.05e-03$	$1.40e+02$	$2.30e+01$
	ITHS	$2.76e+01$	$9.69e+00$	$1.07e+04$	$1.17e+01$	$1.15e+00$	$1.42e+00$	$1.08e+00$	$1.36e+03$	$4.82e+01$	$4.97e+00$
	DBC	$2.67e+03$	$4.72e+01$	$3.17e+07$	$1.92e+02$	$2.27e+01$	$1.30e+01$	$2.54e+01$	$8.91e+02$	$2.32e+02$	$1.31e+02$
	CHC	$0.00e+00$	$6.19e+01$	$1.25e+06$	$7.43e+01$	$1.67e-03$	$0.00e+00$	$0.00e+00$	$2.24e+02$	$3.10e+02$	$7.30e+00$
	G-CMA-ES	$0.00e+00$	$0.00e+00$	$7.97e-01$	$1.05e+02$	$2.96e-04$	$2.09e+01$	$0.00e+00$	$0.00e+00$	$1.66e+01$	$6.81e+00$
	PSO-NBA (linear)	$0.00e+00$	$6.40e+00$	$6.22e+01$	$1.96e+02$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$2.07e+02$	$1.03e+02$	$0.00e+00$
	PSO-NBA (nonlinear)	$0.00e+00$	$7.55e+00$	$8.87e+01$	$1.95e+02$	$6.36e-03$	$0.00e+00$	$0.00e+00$	$6.74e+00$	$1.02e+02$	$1.73e+00$
100	EvoPROpt	$4.34e-02$	$3.30e+00$	$3.98e+02$	$1.07e-01$	$3.92e-02$	$2.50e-04$	$9.17e-02$	$2.27e+03$	$2.91e+01$	$2.05e-01$
	SPSO2011	$0.00e+00$	$6.87e+01$	$2.23e+03$	$4.02e+02$	$3.65e-03$	$3.81e+00$	$5.18e+01$	$7.12e+00$	$3.81e+02$	$6.16e+01$
	ITHS	$2.03e+02$	$1.95e+01$	$2.08e+05$	$5.12e+01$	$2.60e+00$	$2.74e+00$	$4.58e+00$	$6.88e+03$	$1.31e+02$	$2.47e+01$
	DBC	$1.35e+04$	$6.52e+01$	$3.15e+08$	$4.35e+02$	$1.14e+02$	$1.47e+01$	$6.54e+01$	$4.25e+03$	$4.97e+02$	$4.06e+02$
	CHC	$0.00e+00$	$8.58e+01$	$4.19e+06$	$2.19e+02$	$3.83e-03$	$0.00e+00$	$1.40e-02$	$1.69e+03$	$5.86e+02$	$3.30e+01$
	G-CMA-ES	$0.00e+00$	$0.00e+00$	$3.88e+00$	$2.50e+02$	$1.58e-03$	$2.12e+01$	$4.22e-04$	$0.00e+00$	$1.02e+02$	$1.66e+01$
	PSO-NBA (linear)	$0.00e+00$	$2.39e+01$	$3.09e+02$	$5.10e+02$	$0.00e+00$	$7.86e-01$	$0.00e+00$	$9.67e+03$	$4.37e+02$	$8.90e+00$
	PSO-NBA (nonlinear)	$0.00e+00$	$2.60e+01$	$3.77e+02$	$5.44e+02$	$0.00e+00$	$8.93e-01$	$0.00e+00$	$1.71e+03$	$4.61e+02$	$8.00e+00$
		SC-TP10	SC-TP11	SC-TP12	SC-TP13	SC-TP14	SC-TP15	SC-TP16	SC-TP17	SC-TP18	
50	EvoPROpt	$9.68e+00$	$2.27e+00$	$4.22e+01$	$9.97e-01$	$6.38e-02$	$5.63e+00$	$6.77e+01$	$1.62e+00$	$5.03e-02$	
	SPSO2011	$1.34e+02$	$7.84e+01$	$1.20e+02$	$1.07e+02$	$1.64e+01$	$1.62e+02$	$3.45e+02$	$6.37e+01$	$1.89e+01$	
	ITHS	$4.66e+01$	$2.12e+01$	$1.98e+03$	$8.70e+00$	$1.20e+00$	$9.09e+00$	$1.19e+02$	$1.93e+00$	$1.91e+00$	
	DBC	$2.25e+02$	$1.95e+02$	$1.15e+05$	$1.18e+02$	$3.87e+01$	$2.07e+02$	$2.73e+02$	$6.10e+01$	$5.22e+01$	
	CHC	$2.16e+00$	$9.57e-01$	$2.08e+06$	$6.17e+01$	$3.98e-01$	$0.00e+00$	$2.26e+04$	$1.58e+01$	$3.59e+02$	
	G-CMA-ES	$3.01e+01$	$1.88e+02$	$1.97e+02$	$1.09e+02$	$9.79e-04$	$4.27e+02$	$6.89e+02$	$1.31e+02$	$4.76e+00$	
	PSO-NBA(linear)	$9.50e+01$	$8.48e-05$	$1.32e+02$	$1.42e+02$	$0.00e+00$	$3.85e+01$	$3.01e+02$	$6.79e+01$	$0.00e+00$	
	PSO-NBA (nonlinear)	$1.13e+02$	$7.58e+01$	$1.82e+02$	$1.50e+02$	$0.00e+00$	$1.94e+02$	$3.91e+02$	$7.02e+01$	$1.24e+00$	
100	EvoPROpt	$2.60e+01$	$5.01e+00$	$1.40e+02$	$1.24e+00$	$6.56e-02$	$8.29e+00$	$1.97e+02$	$3.34e+00$	$1.43e-01$	
	SPSO2011	$3.76e+02$	$1.68e+02$	$2.36e+02$	$2.87e+02$	$5.95e+01$	$3.31e+02$	$6.52e+02$	$1.49e+02$	$5.65e+01$	
	ITHS	$1.38e+02$	$1.16e+02$	$1.74e+04$	$3.86e+01$	$5.87e+00$	$5.71e+01$	$4.11e+02$	$6.10e+00$	$1.08e+01$	
	DBC	$5.07e+02$	$7.82e+02$	$2.95e+06$	$2.83e+02$	$1.08e+02$	$4.61e+02$	$6.01e+02$	$1.32e+02$	$1.42e+02$	
	CHC	$7.32e+01$	$1.03e+01$	$2.70e+06$	$1.66e+02$	$8.13e+00$	$2.23e+01$	$1.47e+05$	$7.00e+01$	$5.45e+02$	
	G-CMA-ES	$1.64e+02$	$4.17e+02$	$4.21e+02$	$2.55e+02$	$6.30e-01$	$8.59e+02$	$1.51e+03$	$3.07e+02$	$2.02e+01$	
	PSO-NBA(linear)	$4.46e+02$	$7.88e+01$	$3.53e+02$	$3.59e+02$	$0.00e+00$	$2.96e+02$	$6.60e+02$	$1.73e+02$	$0.00e+00$	
	PSO-NBA (nonlinear)	$4.65e+02$	$1.81e+02$	$3.89e+02$	$3.99e+02$	$0.00e+00$	$3.96e+02$	$8.21e+02$	$1.73e+02$	$2.58e+00$	

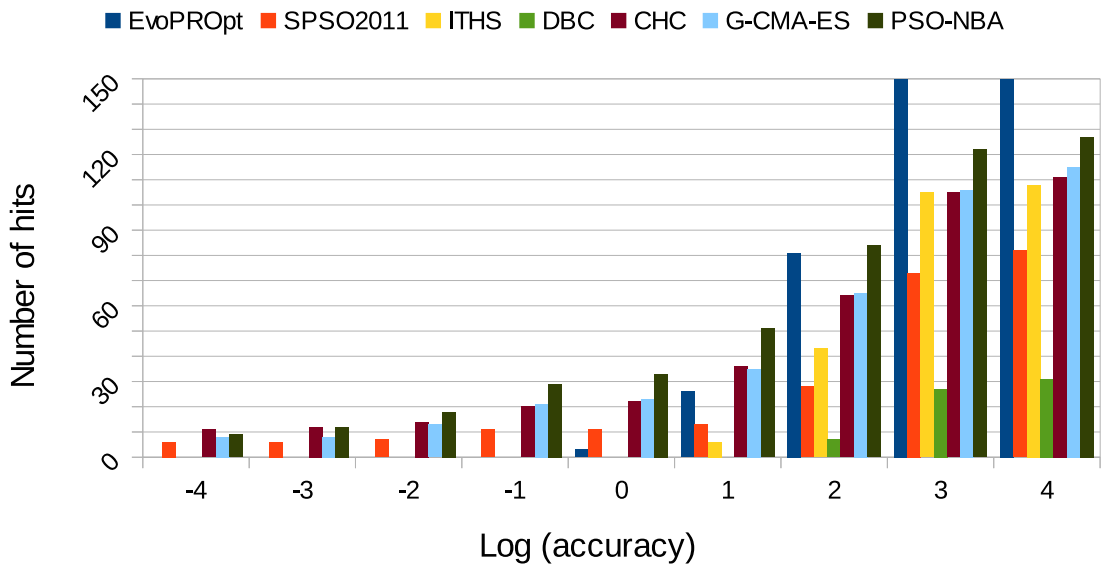


Figure 2.10: Cumulative number of hits for different accuracy levels (linear case).

parative results for different algorithms are also publicly available [2].

PSO-NBA was applied on the 50- and 100-dimensional instances of the test problems, adopting the experimental setting in [63]. In all cases, the algorithm assumed population size equal to $2 \times n$, where n stands for the problem's dimension. At each experiment, the solution error $|f_{\text{PSO-NBA}} - f^*|$ was recorded, where f^* denotes the actual global minimum of the problem and $f_{\text{PSO-NBA}}$ is the best solution value achieved by the proposed approach.

The two best SOBA approaches and the two best MOBA approaches (in terms of number of wins) from Table 2.9 were considered for further experimentation. These approaches were also compared against six established algorithms, namely the CHC Genetic (Cross-generational elitist selection, Heterogeneous recombination and Cataclysmic mutation) algorithm [65], the G-CMA-ES (Restart Covariant Matrix Evolutionary Strategy) algorithm [66], the EvoPROpt (Evolutionary Path Relinking) algorithm [67], the SPSO2011 (Standard PSO 2011) algorithm [68], the ITHS (Intelligent Tuned Harmony Search) algorithm [69,70] and the DBC (Directed Bee Colony) algorithm [71]. Note that G-CMA-ES was the dominant algorithm in the CEC 2005 challenge.

In Table 2.14, the obtained average errors for the two SOBA variants LB/NL/2.0

[2] http://sci2s.ugr.es/eamhco/SOCO − results.xls

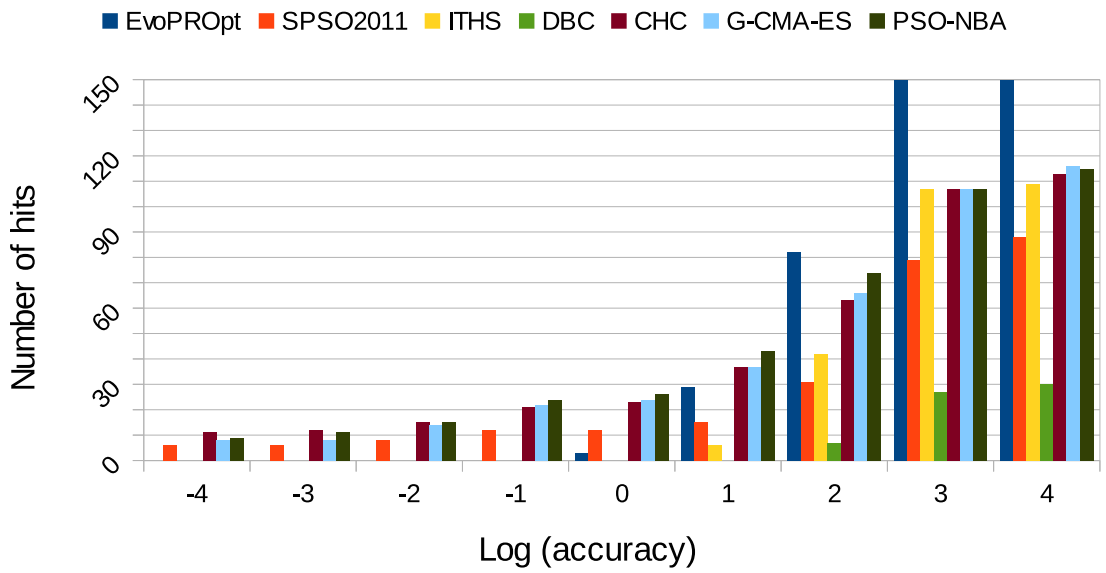


Figure 2.11: Cumulative number of hits for different accuracy levels (nonlinear case).

and LB/L/2.0 are reported. MOBA variants had slightly inferior performance, which was anticipated since in the previous experiments they were shown to perform better in lower dimensions. For this reason they are omitted from the current results. Also, in Table 2.14 the corresponding results for the rest of the algorithms are reported. The results of the EvoPROpt, G-CMA-ES, and CHC algorithms are publicly available in the aforementioned online sources, while the results of the SPSO2011, ITHS and DBC were produced with implementations that closely followed the instructions, pseudocodes, and parameter settings provided in the original sources.

A first inspection of the results reveals that PSO-NBA is highly competitive to the other algorithms. More specifically, the linear PSO-NBA variant achieved zero or marginally deviant values in 12 problem instances, while the nonlinear variant achieved similar success in 8 out of 38 problem instances. The corresponding number of successes for EvoPROpt, SPSO2011, ITHS, DBC, G-CMA-ES, and CHC were 0, 2, 0, 0, 7, and 6, respectively, out of 38 problem instances.

In order to facilitate comparisons and provide further insight on the algorithm's effectiveness, pairwise comparisons of each algorithm with the rest were conducted. At each comparison, the number of *hits* (successes) over the accuracy levels were recorded,

$$10^{-4},\ 10^{-3},\ 10^{-2},\ 10^{-1},\ 10^{0},\ 10^{1},\ 10^{2},\ 10^{3},\ 10^{4}.$$

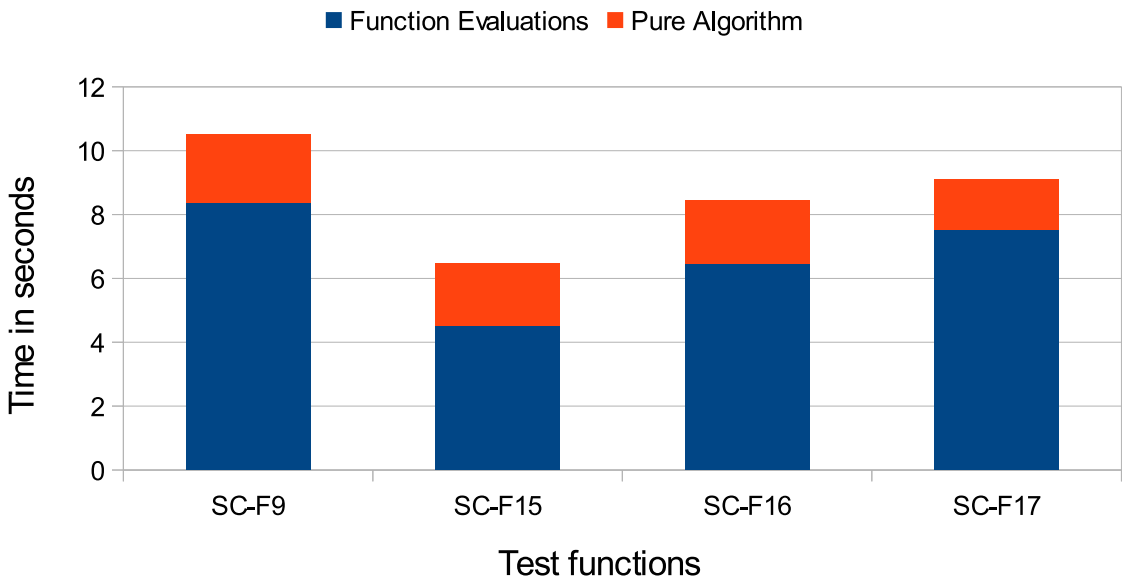


Figure 2.12: Portion of time spent on algorithmic procedures vs function evaluations.

A hit is recorded for an algorithm when it outperforms another algorithm, i.e., it achieves a lower average error for the specific problem and dimension, and both their average errors are smaller than the particular accuracy level.

Figures 2.10 and 2.11 depict the distribution of the number of hits over the predefined accuracy levels for the linear and the nonlinear case, respectively. Evidently, PSO-NBA exhibits high numbers of hits for the majority of accuracy levels, especially for the smallest ones, which are the most desirable in practice. EvoPROpt outperformed PSO-NBA only for the highest accuracy levels. Yet, it does not achieve any hit in almost half of the (smaller) accuracy levels.

The presented experimental evidence verifies that the proposed PSO-NBA approach can be very competitive also to other algorithms. Of course, the best choice among different PSO-NBA variants is always problem-dependent. However, the observations that were pointed out in the previous sections can be helpful for the practitioner.

Finally, the time complexity of the algorithm is considered. Specifically, the fraction of the time spent to algorithmic procedures against the time spent purely for function evaluations per run was investigated. Figure 2.12 illustrates the required time for the 100-dimensional instances of 4 of the most demanding problems from the current test suite. The measured time is indicative for a single experiment. De-

spite the high dimensionality of the problems, the large population size, and the lack of any optimization in the source code of the implementation, it can be clearly seen that the function evaluation dominates the time required by the algorithm. This is an indication that smaller execution times can be achieved with further optimization of the algorithm's procedures and source code.

2.9 Synopsis

In this chapter, state-of-the-art metaheuristic optimization algorithms were outlined. In the next chapters, novel parallel algorithm porfolio frameworks are proposed that harness the metaheuristic algorithms exposed in this chapter.

Moreover, PSO-NBA was introduced, which is an asynchronous PSO variant that distributes the available computational budget of function evaluations in an irregular way among the particles of the swarm. In order to select the favored particles, the algorithm assesses their neighborhoods with respect to solution quality and diversity. Particles that possess highly ranked neighborhoods have higher probability of receiving function evaluations than the rest.

Two essential budget allocation strategies were introduced, namely a single-objective and a multi-objective one. For both strategies, a multitude of PSO-NBA variants were defined. All variants were tested on a standard suite of benchmark problems as well as on problems drawn from real-life applications. The most successful variants were distinguished after statistical analysis of the results. Further experiments were conducted on an established test suite. Comparisons with various algorithms were provided.

The acquired results suggested that PSO-NBA can be highly competitive. Overall, elitistic options were shown to be beneficial on performance. Both single-objective and multi-objective strategies exhibited efficiency and robustness. The provided evidence justified the significance of budget allocation in metaheuristics, leaving ground for further research.

Chapter 3

Algorithm Portfolios

In this chapter, the concept of algorithm portfolios is introduced and a literature review is presented. Then, two standard parallel algorithm portfolio models are proposed. The first one can be applied with any optimization algorithm to enhance efficiency. It is demonstrated on the design of S-boxes, which is an important problem in cryptography. The second algorithm portfolio is specialized on population-based algorithms. This portfolio is applied on a challenging problem that originates from smart cities, namely the traffic light scheduling problem.

3.1 Introduction

The term *Algorithm Portfolio* (AP) refers to a general framework where multiple individual algorithms are combined into a single algorithmic scheme and share the

available computational resources [9]. APs emerged several years ago as a promising approach to tackle challenging optimization problems. Their computational efficiency against individual metaheuristics has lead to a constantly increasing popularity [4, 9–11, 13, 15, 72]. The rationale behind APs lies in the fact that the best algorithm for a given problem is rarely *a priori* known. Thus, employing a number of preferably diverse algorithms can increase the probability of finding a good solution, while reducing the *risk* in terms of deviation of the final result from that of a single algorithmic approach.

An AP can consist of multiple copies of one algorithm with identical or different parameters (*homogeneous AP*) or different algorithms (*heterogeneous AP*). All algorithms run concurrently in either one or multiple processing units. If a single processing unit is used, the execution of the algorithms is alternated according to a resources assignment schedule. In multi-core or parallel systems, the algorithms share the hardware resources, i.e., the number of available processing units [10].

The algorithms that comprise an AP can be either isolated (*non-interactive AP*) or communicate and exchange information with each other (*interactive AP*). The communicated information usually comprises the best solutions detected by each algorithm. This model is typically used in APs of population-based algorithms. For example, in [4] the proposed AP accommodates a number of population-based algorithms that exchange solutions via regular migrations. Non-interactive APs have also been applied on problems in operations research [14, 73] and combinatorial optimization [11].

An important issue in the design of high-performance algorithm portfolios is the choice of the appropriate resources allocation plan. The simplistic approach of assigning equal computational budgets among the constituent algorithms of the AP may provide adequate solutions in various optimization problems [74]. However, recent studies have suggested the use of sophisticated budget allocation mechanisms [13, 75] in order to take full advantage of the combined algorithmic power of the APs. To this end, algorithm portfolios that share the computational budget by exploiting dynamic resources allocation mechanisms can offer significant performance benefits.

3.2 Literature review

The first works that introduced APs for solving challenging problems appeared two decades ago [9, 10]. In [9] a general method that incorporated Las Vegas algorithms

into a single AP framework was introduced. The authors showed that the proposed method can be used to tackle a wide range of hard optimization problems. Additionally, the authors investigated the relation between the variation of the AP's performance (i.e., the risk) and its expected performance in terms of the time required to detect a solution. Also, there was evidence suggesting that cooperation among the constituent algorithms of the AP can further boost its performance.

In [10] the proposed AP consisted of a number of stochastic algorithms running interchangeably on one processor or employing more processing units without communication among them during their execution. The principal goal of that research was to show that specific APs are preferable than other ones with respect to the expected computational cost and the overall risk. For this purpose, an experimental evaluation of the AP approach on diverse hard combinatorial problems was conducted. The acquired results indicated the superiority of APs compared to other algorithmic approaches.

Fukunaga *et al.* was the first to propose AP schemes that harness metaheuristic optimization algorithms. In [11] an AP that distributed the available computational budget among algorithms with different configurations instead of a single algorithm was proposed. Unlike previous works based on Las Vegas algorithms, that AP used anytime algorithmic approaches, which can be interrupted at any time offering suboptimal solutions. Specifically, different configurations of a genetic algorithm were incorporated into the AP and its performance was investigated on the well known traveling salesman problem.

In [4] the proposed AP accommodated a number of well studied population-based algorithms to solve numerical optimization problems. The AP allocated the computational budget its constituent algorithms, which exchanged solutions via regular migrations. Additionally, that work focused on reducing the risk of an algorithmic approach on a collection of optimization problems. In contrast, previous works were interested only in measuring the risk of the application of a single algorithm on a single problem by conducting a number of independent experiments. The experimental results showed that the proposed AP framework outperformed its constituent algorithms in terms of solution quality as well as with respect to a proposed risk measure for the majority of the tested problems.

In [15] a multi-algorithm genetically adaptive method was proposed, which embraced different population-based algorithms into a single framework. In this frame-

work, multiple algorithms ran concurrently and interacted with each other by exchanging information via a shared population of search points. At each iteration, each constituent algorithm generated a fraction of the offspring. This fraction was adjusted for each algorithm during the optimization process and depended on their performance in previous iterations. Finally, the authors argued that the success of a multi-algorithm method depends on the efficiency of the constituent algorithms as well as their complementary search capabilities.

Non-interactive APs have also been applied on problems that originate in operations research [14, 73] and combinatorial optimization [11]. The lack of information exchange in this case can be beneficial in some applications where the search is prone to get rapidly biased in narrow neighborhoods of the search space. In such cases, the migration of solutions from one algorithm to another can eventually result in limited exploration capability or premature convergence of the AP [13, 76].

A crucial issue in the design of efficient and effective APs is the appropriate selection of its constituent algorithms [5, 12]. Choosing the AP's constituent algorithms from a wide range of available methods is a hard task that is usually conducted offline. In [5] several selection policies were proposed to construct APs from predefined sets of algorithms. The proposed methods were applied on several instances of a maintenance scheduling problem. Their evaluation was based on the running time required to formulate the APs. The APs were assessed on the basis of solution quality, and their performance was shown to be problem-dependent.

In [12] an automatic selection approach was proposed to address the constituent algorithms selection problem. Specifically, the AP proposed in [4] was extended with an automatic selection mechanism based on an estimated performance matrix for each algorithm from a predefined set of evolutionary algorithms. This matrix recorded the performance of each candidate algorithm during a preprocessing phase. During this phase, each algorithm was applied on the problem under consideration, consuming a fraction of the total available computational budget. Then, selection of the most promising methods took place, and they were eventually integrated into the AP. The formulated AP was executed with the remaining budget after the selection phase. Experimental results showed that the proposed AP approach outperformed its constituent algorithms in terms of solution quality.

The authors in [77] proposed an automated mechanism to construct APs for memetic algorithms. Their approach targeted at finding the best algorithm sets over

a number of combinations of crossover, mutation, and local search operators. To this end, the combinations were clustered with respect to their performance when applied on the tested problems. Then, the best combination of each cluster was selected and integrated into the AP. Finally, an online algorithm selection mechanism was used to orchestrate the AP. The proposed method was evaluated on instances of the quadratic assignment problem.

The first research work that investigated the use of APs on multi-objective optimization problems appeared in [78]. The AP consisted of a set of evolutionary algorithms, which were executed without any communication among them. Differing from previous AP approaches, each evolutionary algorithm exploited an independent population of individuals. Also, an essential characteristic of that approach was the lack of additional parameters in the AP. At each generation, the performance of each algorithm was predicted at a future point by exploiting a score calculation method. Then, the algorithm selector came into play and determined the best-performing algorithm that was to be executed for the subsequent generation. This way, the constituent algorithms were selected as a function of the available computational budget. Experimental results showed that the proposed AP can achieve better solution quality than that when using one of its constituent algorithms solely.

In [74] the proposed APs were used to tackle optimization problems contaminated by noise. In that work, two AP models were proposed. The first one distributed the available computational budget equally among the constituent algorithms while the second one suggested an unfair sharing among them. Bounds of the performance of the two AP models were provided. Also, it was shown that performance history is beneficial for the selection of the best-performing algorithm. Experimental results showed that APs are efficient especially for noisy problems as long as their constituent algorithms exhibit different convergence behaviors when applied on different problems.

In [13], an AP framework that tackles single-objective optimization problems was proposed. Several well known evolutionary algorithms were integrated into the AP, while no information exchange occurred among them during their execution. The proposed framework employed a predictive mechanism that predicted online the performance of its constituent algorithms based on past history. Specifically, the mechanism predicted the fitness values of the algorithms at some future point. This was achieved by extrapolating the convergence curves of the algorithms up to the point

of interest. The algorithm that achieved the lowest predicted function value was executed for one generation and then the performance of each algorithm was predicted again. In this way, the contituent algorithms switched between iterations as a function of the available computational budget. Extensive experimental evaluation on a well know benchmark suite revealed that the proposed AP outperformed other AP approaches as well as its constituent algorithms with respect to solution quality.

A first parallel implementation of the population-based AP framework [4] appeared in [72]. In that work, the performance of different parallel models used in the construction of APs was investigated. In particular, the efficiency of the proposed models was measured with respect to their execution times as well as by using the speedup performance metric. Additionally, the performance of the parallel AP was assessed on training neural networks with different architectures. Experimental results on a popular benchmark suite showed the robustness of the proposed parallel AP compared to its sequential counterparts. Also, is was demonstrated that speedup is affected by the distribution of the individuals of the algorithms in the shared population of the AP.

3.3 Parallel Algorithm Portfolios

In this section, two parallel algorithm portfolio models are considered. Firstly, a non-interactive AP model is presented where algorithms run individually without any communication among them. Next, an AP model focused on population-based algorithms is proposed.

3.3.1 Standard Model

A non-interactive AP is considered, which consists of M metaheuristic algorithms that run in parallel according to a master-slave model. Each constituent algorithm runs individually on a single slave node without information exchange. Whenever an algorithm detects a new solution $\mathbf{x}_i$ that improves its current best one, it sends it along with its function value $f(\mathbf{x}_i)$ to the master node. The master node is responsible for all bookkeeping operations, storing the best solutions discovered by the algorithms and retaining the current overall best solution $\mathbf{x}_{\text{best}}$ and its function value f_{best}. Pseudocode of the AP is provided in Algorithm 3.1 (master node) and Algorithm 3.2 (slave nodes).

Algorithm 3.1 Parallel algorithm portfolio: master node

Input: Number of slaves (M)

Output: Best detected solution

initialize M slave-nodes and assign an algorithm to each one
$t \leftarrow 0$, $f_{\text{best}} \leftarrow \infty$
while (nodes still running) **do**
  receive $\mathbf{x}_i$, $f_i = f(\mathbf{x}_i)$ from node i
  if ($f_i < f_{\text{best}}$) **then**
    $f_{\text{best}} \leftarrow f_i$, $\mathbf{x}_{\text{best}} \leftarrow \mathbf{x}_i$
  end if
end while
return $\mathbf{x}_{\text{best}}$

Algorithm 3.2 Parallel algorithm portfolio: slave nodes

Input: Allocated execution time (T_{exec})

initialize assigned algorithm
while (T_{exec} not exceeded) **do**
  execute one iteration of the algorithm
  if (new best $\mathbf{x}_i$ found) **then**
    send $\mathbf{x}_i$, $f_i = f(\mathbf{x}_i)$ to master node
  end if
end while
terminate slave node

3.3.2 Algorithm Portfolio for Population-based Algorithms

Hard optimization problems require long computational times. In the case of metaheuristics, the most costly part of their execution is the evaluation of the objective function. Parallel AP frameworks for population-based metaheuristics have prevailed in recent years to mitigate this deficiency [15, 72]. In this section, an AP framework that is based on the well known *parallelization of population* approach [79] is considered. This approach proposes the division of the population into parts, each one assigned to a different processing unit.

Also, a simple master-slave parallelization model is employed, where the main algorithm runs on the master node and the slave nodes are devoted to the com-

putation process. Specifically, the function evaluations of the population are divided into equal parts, each one assigned to a slave node running on a single processor. At each iteration, the algorithm requires the evaluation of the individuals. First, the master sends the solution vectors (individuals) to the assigned slave nodes. Then, each slave computes the function values of the assigned individuals. Eventually, each slave returns the computed function values back to the master, where they are further exploited for the next iteration of the algorithm.

An obvious advantage of using the considered AP is the load distribution of the function evaluations among different processing units. Therefore, its use is even more beneficial in cases where each function evaluation requires high computation times. Also, note that the employed parallel AP model does not affect the quality of the detected solutions (nor it changes the search model) compared to the serial algorithm.

3.4 Application in Design of S-boxes

Substitution boxes (S-boxes) constitute essential parts of modern cryptographic applications. In essence, S-boxes are multi-input, multi-ouput Boolean functions that map binary input to binary output values. S-boxes lie at the core of symmetric-key cryptographic algorithms. Specifically, they are used to conceal the relation between the input key and the encrypted output message. Therefore, they have crucial impact on the algorithm's security quality [80, 81].

The design of suitable S-boxes has been an active research area for several decades with significant applications in symmetric-key cryptography standards. A widely known example is the Data Encryption Standard (DES), which was introduced in 1977 and it is based on eight 6-input 4-output S-boxes [82]. DES has been proved to be vulnerable to crack attacks such as linear cryptanalysis and parallel brute-forcing. Thus, it was superseded by the Advanced Encryption Standard (AES), which was proposed in 2000 and its implementation is also based on S-boxes [83]. Specifically, it employs a properly designed Rijndael S-box that is resistant to linear [84] and differential [85] cryptanalysis, which are most popular cipher-attacks. Today, AES is widely considered as a highly secure standard.

The problem of designing S-boxes with desirable properties has been tackled both through algebraic techniques [19, 20] and computational methods [86–88]. Among

them, a large body of work has been devoted to the application of metaheuristic algorithms. Typical examples are Evolutionary Algorithms and Swarm Intelligence approaches, such as Particle Swarm Optimization [86], Differential Evolution [86], and Genetic Algorithms [87]. Also, trajectory-based algorithms such as Simulated Annealing have been used [88]. Despite the reported effectiveness of these algorithms, their running time efficiency has been habitually neglected in the relevant studies.

This fact can raise concerns, since most of the studied computational algorithms require a remarkably large amount of time to produce S-boxes of high nonlinearity and low autocorrelation. Indicatively, running time has been reported to reach even several days of uninterrupted execution of the algorithm for some approaches [88]. In addition, formal experimental and statistical analysis of the algorithms' performance has been rarely reported in relevant literature. Instead, most efforts have been concentrated on reporting an S-box of good quality. Hence, existing performance assessments and comparisons of different metaheuristics on the S-boxes design problem are rather obscure. Nevertheless, it is widely perceived that the necessity for efficient and effective algorithms is indispensable for the specific problem type.

This section focuses on the application of parallel APs on the design of S-boxes. Previous research suggested that *hill-climbing* properties can be highly beneficial for metaheuristics in the detection of S-boxes with desirable properties [89]. Thus, a simple and efficient parallel AP approach that comprises the well studied Tabu Search (TS) [90] and Simulated Annealing (SA) [25] algorithms is considered. The selection of the specific constituent algorithms is motivated by their recognized efficiency and their inherent hill-climbing capabilities. This is the first time that TS as well as the APs approach are used for the specific problem.

The proposed algorithmic approaches are evaluated and statistically analyzed on widely used problem instances. In order to make the problem even more challenging, tight running time constraints are considered. Thus, in a first experimentation phase the plain sequential TS and SA algorithms are applied and analyzed on the S-boxes design problem. The use of sequential algorithms is the most frequent approach reported in relevant literature. In a second experimentation phase, homogeneous and heterogeneous APs are composed using combinations of different TS and SA variants. Then, the capability of the APs in outperforming TS and SA with respect to solution quality and time efficiency is experimentally studied.

The considered AP model is inherently parallel, i.e., the constituent algorithms are

simultaneously executed on multiple processors. In order to perform fair comparisons with the sequential algorithms, which are executed on a single processor, the parallel AP is assigned exactly the same total running time with the sequential approaches. This time is then equally allocated to the AP's constituent algorithms. Thus, the total running time of the AP cannot exceed that of the sequential algorithms; it is simply exploited differently. The obtained experimental results suggest that the simultaneous execution of multiple algorithms can improve performance in terms of solution quality and time efficiency. Also, various speculations regarding the dynamics of the APs and the TS and SA algorithms are verified on the investigated problems.

3.4.1 Problem Formulation

In this section, the mathematical formulation of the S-boxes design problem is provided. Let

$$f : \mathcal{B}^n \rightarrow \mathcal{B}^m, \qquad \mathcal{B} = \{0, 1\},$$

be a *vectorial Boolean function* that maps n Boolean input values to m Boolean output values. This function is called an n-input m-output S-box or, simply, an $n \times m$ S-box. In case of a single output, i.e., $m = 1$, the S-box degenerates to a Boolean function. The following definitions refer to Boolean function properties.

Definition 3.1. (*Polarity truth table*) Let $\mathbf{x} \in \mathcal{B}^n$. A useful representation of a Boolean function is the polarity truth table defined as,

$$\hat{f}(\mathbf{x}) = (-1)^{f(\mathbf{x})},$$

which maps the output values of the Boolean function from the set $\{0, 1\}$ to the set $\{-1, 1\}$.

Definition 3.2. (*Linear Boolean function*) Let $\mathbf{x}, \mathbf{w} \in \mathcal{B}^n$. A linear Boolean function is defined as,

$$L_w(\mathbf{x}) = w_1 x_1 \oplus w_2 x_2 \oplus \cdots \oplus w_n x_n,$$

where $w_i x_i$ denotes the bitwise AND operation for all $i \in \{1, 2, \ldots, n\}$, and $\oplus$ denotes the bitwise XOR. A linear Boolean function in polarity form is denoted as $\hat{L}_w(\mathbf{x})$.

Definition 3.3. (*Affine Boolean function*) The set of affine Boolean functions includes the set of linear Boolean functions and their complements, i.e., all functions of the form,

$$A_{w,c}(\mathbf{x}) = L_w(\mathbf{x}) \oplus \mathbf{c}, \qquad \mathbf{c} \in \mathcal{B}.$$

Definition 3.4. (*Walsh-Hadamard transform of Boolean function*) The Walsh-Hadamard transform (WHT) of a Boolean function f is defined as,

$$\hat{F}_f(\mathbf{w}) = \sum_{\mathbf{x}\in\mathcal{B}^n} \hat{f}(\mathbf{x})\, \hat{L}_w(\mathbf{x}),$$

and it measures the correlation between the Boolean function f and the linear Boolean function $\hat{L}_w$ with $\mathbf{x} \in \mathcal{B}^n$.

Definition 3.5. (*Nonlinearity of Boolean function*) The nonlinearity NL_f of a Boolean function f is defined as,

$$NL_f = \frac{1}{2}\left(2^n - WH_{\max}\left(f, \mathbf{w}\right)\right),$$

where,

$$WH_{\max}(f, \mathbf{w}) = \max_{\mathbf{w}\in\mathcal{B}^n} \left|\hat{F}_f(\mathbf{w})\right|,$$

i.e., it is the maximum absolute value of WHT. The computation of $WH_{\max}(f, \mathbf{w})$, except for the Boolean function f, also requires to specify a linear function through the vector $\mathbf{w}$.

Definition 3.6. (*Autocorrelation of Boolean function*) The autocorrelation of a Boolean function f measures its self-similarity. It is defined as,

$$\hat{r}_f(\mathbf{s}) = \sum_{\mathbf{x}\in\mathcal{B}^n} \hat{f}(\mathbf{x})\, \hat{f}(\mathbf{x} \oplus \mathbf{s}),$$

where $\mathbf{s} \in \mathcal{B}^n$. The maximum absolute value of the autocorrelation of a function f is defined as,

$$AC_f = \max_{\mathbf{s}\in\mathcal{B}^n\setminus\{0^n\}} \left|\sum_{\mathbf{x}\in\mathcal{B}^n} \hat{f}(\mathbf{x})\, \hat{f}(\mathbf{x} \oplus \mathbf{s})\right|,$$

where 0^n denotes the null vector over $\mathcal{B}^n$.

Theorem 3.1. *(*Parseval's theorem*) It holds that*,

$$\sum_{\mathbf{w}\in\mathcal{B}^n} \left(\hat{F}_f(\mathbf{w})\right)^2 = 2^{2n},$$

which in turn results in $WH_{\max}(f, \mathbf{w}) \geqslant 2^{n/2}$.

Definition 3.7. (*Balanced Boolean function*) If the number of 0-valued outputs of a Boolean function is equal to the number of its 1-valued outputs, then the Boolean function is called balanced.

Boolean functions compose S-boxes. An S-box, $f : \mathcal{B}^n \to \mathcal{B}^m$, is a combination of m single-output Boolean functions. In order to extend the theoretical background from Boolean functions to S-boxes, the m output values of the S-box can be transformed into a single-output Boolean function. Let $f_\beta(\mathbf{x})$, $\beta \in \mathcal{B}^m$, be a linear combination of the m output values of the S-box f,

$$f_\beta(\mathbf{x}) = \beta_1 f_1(\mathbf{x}) \oplus \beta_2 f_2(\mathbf{x}) \oplus \cdots \oplus \beta_m f_m(\mathbf{x}),$$

where $f_i(\mathbf{x})$, $i \in \{1, 2, \ldots, m\}$, denotes the i-th output bit of the S-box. There are $2^m - 1$ such linear combinations, excluding the trivial case of the linear combination with the null vector. The aforementioned nonlinearity and autocorrelation properties of Boolean functions can be extended to S-boxes as follows.

Definition 3.8. (*Walsh-Hadamard transform of S-box*) The Walsh-Hadamard transform (WHT) of an S-box is defined as,

$$\hat{F}_\beta(\mathbf{w}) = \sum_{\mathbf{x} \in \mathcal{B}^n} \hat{f}_\beta(\mathbf{x})\, \hat{L}_w(\mathbf{x}). \tag{3.1}$$

Definition 3.9. (*Nonlinearity of S-box*) The nonlinearity NL_f of an S-box is defined as,

$$NL_f = \frac{1}{2}\left(2^n - WH_{\max}\left(f_\beta, \mathbf{w}\right)\right), \tag{3.2}$$

where $WH_{\max}(f_\beta, \mathbf{w})$ is the minimum among all maximum absolute values of WHT over the $2^m - 1$ linear combinations of the output values of the S-box.

Definition 3.10. (*Autocorrelation of S-box*) The autocorrelation of an S-box is denoted as $\hat{r}_\beta(\mathbf{s})$, where $\mathbf{s} \in \mathcal{B}^n \setminus \{0^n\}$, and $\beta \in \mathcal{B}^m \setminus \{0^m\}$, and it is the highest autocorrelation value over the $2^m - 1$ linear combinations of the output values of the S-box.

Definition 3.11. (*Balanced S-box*) An S-box is called balanced if the number of preimages of each output is 2^{n-m}.

Definition 3.12. (*Bijective S-box*) An S-box is called bijective if $n = m$ and the number of preimages of each output is 1. Note that a bijective S-box is also balanced. Bijective S-boxes are also denoted as $n \times n$ S-boxes.

The quality of an S-box is primarily assessed by its properties that render it invulnerable to common attacks such as linear [84] and differential cryptanalysis [85]. In this context, nonlinearity and autocorrelation are essential properties. S-boxes of *high*

nonlinearity and *low autocorrelation* become less vulnerable to attacks. Consequently, these two properties are typically adopted as quality measures of S-boxes and, hence, they are at the center of interest in the underlying optimization problems.

In order to maximize nonlinearity, the definition of Eq. (3.2) can be used. Equivalently, the maximization problem is transformed into a minimization problem over all possible S-boxes f as follows,

$$\min_{f} \max_{\substack{\beta \in \mathcal{B}^m \\ \mathbf{w} \in \mathcal{B}^n}} \left| \hat{F}_\beta(\mathbf{w}) \right|, \tag{3.3}$$

where $\hat{F}_\beta(\mathbf{w})$ is the WHT defined in Eq. (3.1). Regarding autocorrelation, the minimization problem is defined as,

$$\min_{f} \max_{\substack{\beta \in \mathcal{B}^m \setminus \{0^m\} \\ \mathbf{s} \in \mathcal{B}^n \setminus \{0^n\}}} \left| \hat{r}_\beta(\mathbf{s}) \right|, \tag{3.4}$$

where $\hat{r}_\beta(\mathbf{s})$ is the autocorrelation defined in Definition 3.10.

The minimization problems of Eqs. (3.3) and (3.4) are consistently considered in the relevant literature for the design of S-boxes of high nonlinearity and low autocorrelation [86–88]. Typically, the nonlinearity is considered as the primary optimization objective, while autocorrelation is mostly reported for the final solution. Probably, this is due to the computational overhead imposed by the concurrent handling of both objectives.

3.4.2 Implementation Details

The bijective S-boxes design problem is tackled through non-interactive APs consisting of the state-of-the-art SA and TS algorithms running in parallel according to a master-slave model. Details about the employed AP framework can be found in Section 3.3.1, and about the SA and TS algorithms in Sections 2.1 and 2.2, respectively.

Special attention is paid on the representation of the solution vector. The goal is to detect bijective $n \times n$ S-boxes with n binary inputs and n binary outputs. The number of combinations of all binary inputs is equal to 2^n, and each binary input vector has an n-bit output vector. Therefore, the S-box can be represented as a binary solution vector of dimension,

$$D_{\text{bin}} = 2^n \times n.$$

Evidently, the S-boxes design problem is equivalent to a high-dimensional binary optimization problem.

Moreover, the studied S-boxes are bijective as well as balanced. Therefore, each one of the 2^n binary output vectors is mapped to only one of the 2^n binary input vectors. Taking into consideration this property, the solution-generation technique reported in [91] is adopted. This technique constructs all the 2^n possible output vectors of the S-box. Then a candidate solution (S-box) is formed by assigning each n-bit output vector to only one of the 2^n input vectors. Initially the assignment is randomly and uniformly performed as in [86, 87].

Using this technique, the optimization algorithms need to find the proper mapping between the 2^n binary input vectors and the 2^n binary output vectors, i.e., the one that maximizes nonlinearity and minimizes autocorrelation of the resulting S-box. This is achieved by searching for the best position permutation of the 2^n binary output vectors in the S-box. Thus, the D_{bin}-dimensional binary optimization problem is transformed into a D-permutation problem where,

$$D = 2^n. \tag{3.5}$$

This technique retains vectors that satisfy the requirement for equal number of 0 and 1 in the solution, and it has been successfully used with Cartesian Genetic Programming [91]. Note that the resulting search spaces are vast even for small values of n, since their cardinality is equal to the factorial $(2^n)!$.

The evaluation of the generated candidate solutions is based on both nonlinearity and autocorrelation although with different priority. Specifically, higher priority is given to nonlinearity and lower priority to autocorrelation. This is in accordance with the reported significance of the two objectives in relevant literature [92, 93]. It results in a two-stage solution assessment scheme that imitates multi-objective optimization, consisting of the following rules:

(i) Between two candidate solutions, the one that achieves the highest nonlinearity is preferable.

(ii) Between two candidate solutions of equal nonlinearity, the one that has the lowest autocorrelation is preferable.

The algorithms are compared on the basis of their final solution quality, i.e., its nonlinearity and autocorrelation, according to these rules. In case of equivalent solutions in both nonlinearity and autocorrelation, the required running times of the algorithms to achieve their final solutions serve as the tiebreaker.

Table 3.1: Problem size, dimension D of permutation vector, cardinality $|\mathcal{X}|$ of search space, and available time budget (in minutes) per experiment.

Problem	D	$\lvert\mathcal{X}\rvert$	Time (min)
5×5	32	$\sim 10^{35}$	2
6×6	64	$\sim 10^{89}$	60
7×7	128	$\sim 10^{215}$	480
8×8	256	$\sim 10^{506}$	1440

Table 3.2: Parameter settings of the algorithms.

Algorithm	Parameter	Description	Value(s)
AP	M	Number of nodes	$3, 5$
TS	s_{TL}	Tabu list size	D
	T_{noimp}	Non-improving iterations before restart	100
SA	S_T	Number of inner steps	$1500, 2500$
	$T^{(0)}$	Initial temperature	1.0
	α	Cooling factor	$0.5, 0.98$
	T_{noimp}	Non-improving iterations before restart	50

3.4.3 Experimental Results

The sequential TS and SA algorithms as the baseline approaches for comparisons were considered. Note that SA has been previously used on S-boxes design problems [88, 94] while TS is used for the first time. Various instances of the proposed parallel APs with different constituent TS and SA variants were also considered. Each AP followed a master-slave model and initializes a number of nodes. These can be either individual processors or threads in a multi-core processor. In the experiments the APs were run on 3 and 5 nodes. Henceforth, the following notation is used:

(i) TS: sequential TS algorithm.

(ii) SA: sequential SA algorithm.

(iii) $AP(TS, k)$: homogeneous AP on k CPUs, comprising solely TS instances.

Table 3.3: Results for the sequential TS algorithms.

Problem	Alg.		Median			Best		
$n \times m$			NL	AC	Time (msec)	NL	AC	Time (msec)
5×5	TS		10	16	15184.24	10	16	1492.77
	TS_r		10	16	7603.43	10	16	1564.04
6×6	TS		20	32	7936.34	20	32	3365.81
	TS_r		20	32	7342.95	20	32	3502.38
7×7	TS	*	46	48	7332689.32	46	48	4540495.85
	TS_r		46	56	2870727.66	46	48	2242817.24
8×8	TS		98	80	78001050.97	100	80	61694764.99
	TS_r		98	88	9287593.35	100	80	83956598.72

(iv) $AP(SA, k)$: homogeneous AP on k CPUs, comprising solely SA instances.

(v) $AP(TS, SA, k)$: heterogeneous AP on k CPUs, comprising both TS and SA instances.

In all parallel APs, the master node served only for bookkeeping and solution storage purposes, while the slave nodes were devoted to the algorithms. Thus, the APs on 3 and 5 nodes contained 2 and 4 algorithms, respectively.

The experimental evaluation was conducted on the `saw` cluster of the Sharcnet[1] consortium. Each node of this cluster consists of 8 cores (2 sockets $\times$ 4 cores per socket) using Intel© Xeon 2.83 GHz processors with 16 GB RAM. All source codes were developed in the `C` programming language. For the parallelization, the OpenMPI project[2] libraries were used with the `gcc` compiler.

The considered test suite included four bijective S-boxes design problems that have been commonly used for benchmarking in relevant works. In parallel computing environments, the running time of the algorithms has special merit [95]. Usually, strict restrictions apply on the maximum available time per user, while significant costs may apply to users occupying the machines for long time. Adhering to these requirements, running time was considered to be the computational budget in the

[1] http://www.sharcnet.ca

[2] http://www.open-mpi.org

Table 3.4: Results for the sequential SA algorithms.

Problem	Alg.		Median			Best		
$n \times m$			NL	AC	Time (msec)	NL	AC	Time (msec)
5×5	SA_1	*	10	16	23985.54	10	16	11168.44
	SA_2		10	16	48594.72	10	16	37122.27
	SA_3	*	10	16	9057.50	10	16	4995.97
	SA_4		10	24	12210.65	10	16	35346.25
	SA_r	*	10	16	44607.71	10	16	22363.94
6×6	SA_1	*	20	32	2568.39	20	32	2379.13
	SA_2		20	32	34616.83	22	40	814711.13
	SA_3	*	20	32	3983.65	20	32	3961.48
	SA_4		20	32	67908.50	20	32	3951.24
	SA_r		20	32	13223.55	20	32	2853.44
7×7	SA_1	*	46	48	28461662.76	46	48	1375571.42
	SA_2		46	56	243804.27	46	56	51618.37
	SA_3	*	46	48	15404001.48	46	48	1639139.68
	SA_4		46	56	405697.99	46	56	142187.42
	SA_r		46	56	192213.00	46	48	4992287.41
8×8	SA_1	*	100	88	2827104.23	100	80	1769426.69
	SA_2		100	88	30456410.60	100	88	1019285.79
	SA_3	*	100	88	2644293.84	100	80	2230341.29
	SA_4		100	88	37945562.16	100	88	1890570.36
	SA_r	*	100	88	9521700.16	100	80	10031840.50

experimental study. Different budget was provided for each test problem, taking its complexity into consideration. The considered test problems, the dimension D of the corresponding permutation vectors, the cardinality of the search space, and the corresponding time budget per experiment are reported in Table 3.1. The imposed time limits are challenging since, in similar works, running times that span even several days have been reported.

The execution of each algorithm was terminated as soon as it exceeded the prede-

fined running time. In the parallel APs, the total running time was equally allocated to the constituent algorithms. A number of 25 independent experiments were conducted per algorithm and test problem. At each experiment, the best detected solution (S-box) and the elapsed running time until its detection was recorded.

In statistical comparisons, the algorithms were assessed on the basis of solution quality, i.e., the nonlinearity (primarily) and autocorrelation (secondarily) of the achieved solutions, as analyzed in Section 3.4.2. In case of ties under these criteria, the required running times for attaining the solutions were used to identify the dominant algorithm.

3.4.3.1 Results of sequential algorithms

The first round of experiments was devoted to the sequential versions of TS and SA. It is well established that parameterization affects the algorithms' performance. For this reason, the following two settings for both algorithms were considered:

(i) Fixed parameter values according to trial-and-error preprocessing.

(ii) Randomly assigned parameters.

The obtained parameter values for the first case are reported in Table 3.2. Note that the parameters S_T and α of SA exhibited two different promising values each, resulting in four distinct SA instances. Henceforth, the following notation for the sequential approaches will be used:

(1) TS: TS with fixed parameter $s_{TL} = D$.

(2) TS_r: TS with randomly and uniformly selected s_{TL} in the range $[D, 2D]$.

(3) SA_1: SA with $S_T = 1500$ and $\alpha = 0.5$.

(4) SA_2: SA with $S_T = 1500$ and $\alpha = 0.98$.

(5) SA_3: SA with $S_T = 2500$ and $\alpha = 0.5$.

(6) SA_4: SA with $S_T = 2500$ and $\alpha = 0.98$.

(7) SA_r: SA with randomly and uniformly selected S_T and α in the ranges $[1500, 2500]$ and $[0.5, 0.98]$, respectively.

For each test problem and algorithm, the obtained solutions over the 25 experiments were ranked according to their quality, i.e., their nonlinearity (NL) value (primarily), their autocorrelation (AC) value (secondarily), and the required running time otherwise. The median and the best solutions in the ranking are reported for the sequential TS and SA algorithms in Tables 3.3 and 3.4, respectively. The median was preferred against the mean due to the discrete (integer) nature of the NL and AC objectives, as well as due to its use in the considered statistical significance tests.

As reported in Table 3.3, the two TS approaches achieved identical median solutions in terms of NL and AC for the 5×5 and 6×6 problems. In the rest of the problems, the AC values achieved by TS were better than that of TS_r. This is reasonable since TS admitted carefully selected parameters through preprocessing. The best solutions were identical in all cases. For the most challenging 7×7 and 8×8 problems, TS_r achieved a solution of inferior AC quality but same NL value with TS. On the other hand, as the problem dimension increases, TS_r seems to require shorter running time than TS to attain solutions of equal quality with the reported medians.

The experimentation was supported by pairwise Wilcoxon rank-sum tests at significance level 95% among the algorithms. For each test problem, the solutions provided by TS were compared against those of TS_r with respect to NL (primarily), AC (secondarily), and running time otherwise. In case of statistically significant differences between the two algorithms, the dominating one was awarded a *win*, while a *loss* was counted for the other. In case of statistical indifference, both algorithms were awarded a *draw*. The conducted tests revealed insignificant differences in almost all test problems, verifying the mild parameter sensitivity of the TS algorithm on the considered problems. The exception to this was the 7×7 problem, where TS was the dominant algorithm in terms of wins. This is marked with the star "*" symbol in Table 3.3.

The corresponding results for the SA algorithms are reported in Table 3.4. Although there are only few differences in the reported median and best values for the NL, considerable differences are observed for the AC values. A closer look in Table 3.4 reveals that the SA_1 and SA_3 variants, which assumed the $\alpha = 0.5$ parameter, exhibited superior performance, always attaining the best solution. This indicates that rapid modulation of the temperature parameter can be beneficial for the specific problems under the used strict computational budgets. The only exception is the 6×6 problems, where the SA_2 variant discovered the overall best solution in terms of NL.

Table 3.5: Results of the APs for the 5×5 and 6×6 test problems.

Algorithm		Median			Best		
		NL	AC	Time (msec)	NL	AC	Time (msec)
				5×5 S-box			
TS		10	16	15184.24	10	16	1492.77
$AP(TS, 3)$	*	10	16	6576.10	10	16	1039.90
$AP(TS, 5)$	* (2)	10	16	4080.50	10	16	643.20
TS_r		10	16	7603.43	10	16	1564.04
$AP(TS_r, 3)$		10	16	4430.70	10	16	814.80
$AP(TS_r, 5)$	* (1)	10	16	3651.60	10	16	1272.50
SA_3		10	16	9057.50	10	16	4995.97
$AP(SA_3, 3)$		10	16	3988.60	10	16	3988.60
$AP(SA_3, 5)$		10	16	10040.00	10	16	6645.80
SA_r		10	16	44607.71	10	16	22363.94
$AP(SA_r, 3)$	*	10	16	6479.80	10	16	4654.60
$AP(SA_r, 5)$	*	10	16	6790.80	10	16	2176.20
$AP(TS, SA_3, 3)$		10	16	10362.30	10	16	557.20
$AP(TS, SA_3, 5)$		10	16	4998.70	10	16	1552.90
$AP(TS_r, SA_r, 3)$		10	16	6777.50	10	16	1481.50
$AP(TS_r, SA_r, 5)$	*	10	16	4555.60	10	16	1020.30
				6×6 S-box			
TS		20	32	7936.34	20	32	3365.81
$AP(TS, 3)$		20	32	6803.30	22	32	657451.30
$AP(TS, 5)$	*	20	32	6693.30	20	32	3355.80
TS_r		20	32	7342.95	20	32	3502.38
$AP(TS_r, 3)$		20	32	7455.60	22	32	1193029.70
$AP(TS_r, 5)$	*	20	32	3904.10	22	32	881951.40
SA_3	* (1)	20	32	3983.65	20	32	3961.48
$AP(SA_3, 3)$	(2)	20	32	4026.40	20	32	3973.60
$AP(SA_3, 5)$		20	32	4130.80	20	32	4052.50
SA_r		20	32	13223.55	20	32	2853.44
$AP(SA_r, 3)$		20	32	3807.80	20	32	2519.80
$AP(SA_r, 5)$	*	20	32	3799.30	20	32	2542.20
$AP(TS, SA_3, 3)$		20	32	4390.40	20	32	3844.70
$AP(TS, SA_3, 5)$	*	20	32	4224.80	20	32	3585.90
$AP(TS_r, SA_r, 3)$		20	32	4185.40	20	32	3031.50
$AP(TS_r, SA_r, 5)$		20	32	4136.00	20	32	2762.60

Table 3.6: Results of the APs for the 7×7 and 8×8 test problems.

Algorithm		Median			Best		
		NL	AC	Time (msec)	NL	AC	Time (msec)
7×7 S-box							
TS	* (1)	46	48	7332689.32	46	48	4540495.85
$AP(TS, 3)$		46	48	10670926.70	46	48	531649.50
$AP(TS, 5)$	*	46	56	404443.60	46	48	1725213.90
TS_r	*	46	56	2870727.66	46	48	2242817.24
$AP(TS_r, 3)$	*	46	48	6859680.70	46	48	640373.80
$AP(TS_r, 5)$	*	46	56	960809.10	46	48	4461868.70
SA_3		46	48	15404001.48	46	48	1639139.68
$AP(SA_3, 3)$	*	46	48	12653260.80	46	48	276356.10
$AP(SA_3, 5)$	* (1)	46	48	3155517.80	46	48	183949.40
SA_r		46	56	192213.00	46	48	4992287.41
$AP(SA_r, 3)$	*	46	56	33075.10	46	48	2417257.80
$AP(SA_r, 5)$	*	46	56	114760.80	46	48	624332.80
$AP(TS, SA_3, 3)$	*	46	56	102293.00	46	48	375019.10
$AP(TS, SA_3, 5)$	*	46	48	6029739.40	46	48	847927.20
$AP(TS_r, SA_r, 3)$	*	46	56	110495.10	46	48	5592137.80
$AP(TS_r, SA_r, 5)$		46	56	159311.90	46	48	3156178.30
8×8 S-box							
TS	*	98	80	78001050.97	100	80	61694764.99
$AP(TS, 3)$		98	88	11125639.50	98	80	18167071.70
$AP(TS, 5)$		98	88	13297265.00	98	80	15577934.10
TS_r		98	88	9287593.35	100	80	83956598.72
$AP(TS_r, 3)$	*	98	88	9266743.50	98	80	15573342.40
$AP(TS_r, 5)$		98	88	10818025.60	98	80	9497480.20
SA_3	(1)	100	88	2644293.84	100	80	2230341.29
$AP(SA_3, 3)$	*	100	88	910879.20	100	80	5821906.80
$AP(SA_3, 5)$	(2)	100	80	16348693.20	100	80	3554649.30
SA_r		100	88	9521700.16	100	80	10031840.50
$AP(SA_r, 3)$		100	88	5256191.20	100	80	19673499.00
$AP(SA_r, 5)$		100	88	7079210.10	100	80	4949492.80
$AP(TS, SA_3, 3)$	*	100	88	4023823.20	100	80	10780589.40
$AP(TS, SA_3, 5)$	*	100	88	2910635.10	100	80	2351611.20
$AP(TS_r, SA_r, 3)$		100	88	15498679.60	100	88	8189413.10
$AP(TS_r, SA_r, 5)$		100	88	5374450.40	100	80	5179593.80

However, even in this case, the AC of this solution was higher than the aforementioned approaches. Interestingly, the random-parameter variant SA_r closely followed in performance the fixed-parameter variants.

Pairwise Wilcoxon rank-sum tests were conducted among all SA variants. For each test problem, the dominant approaches in terms of wins are marked with a star "*" symbol in Table 3.4, and they have statistically indifferent performance among them. Overall, the SA_3 variant is the most prominent as the problem dimension increases. Also, it achieved very competitive running times, especially for its best solutions. This can be attributed to the $S_T = 2500$ setting that, combined with the rapid decrease of the temperature due to $\alpha = 0.5$, promoted the rejection of non-improving moves more rapidly than in the rest of the SA approaches. Thus, it resulted in fast transition of the algorithm's dynamic from global to local search. Overall, SA appeared to be more sensitive on its parameter setting than TS, while its performance was highly competitive to TS.

3.4.3.2 Results of algorithm portfolios

The second round of experiments focused on the proposed APs. Both fixed and random parameters were considered for the constituent TS and SA algorithms. Due to the large number of combinations, the experiments were restricted on APs comprising two variants of each algorithm. Thus, the promising TS, TS_r, SA_3, and SA_r algorithms were selected to form APs running on 3 and 5 nodes. The designed APs consisted of either a sole algorithm (homogeneous APs) or two different algorithms (heterogeneous APs).

For each test problem, 25 independent experiments were conducted per AP and they were statistically analyzed similarly to the sequential approaches. The obtained results are reported in Tables 3.5 and 3.6. Note that all the received solution values lie within the theoretical bounds reported in [20]. For each test problem, the APs are reported in blocks according to their constituent algorithm. The results of the baseline sequential algorithms, namely TS, TS_r, SA_3, and SA_r, are reproduced from Tables 3.3 and 3.4 to facilitate comparisons, and they are highlighted with light gray color in Tables 3.5 and 3.6.

For example, in the upper half of Table 3.5 (5×5 problem), the first block consists of the sequential TS algorithm and the two homogeneous TS-based APs, namely $AP(TS, 3)$ and $AP(TS, 5)$, with 3 and 5 nodes, respectively. Next comes the block

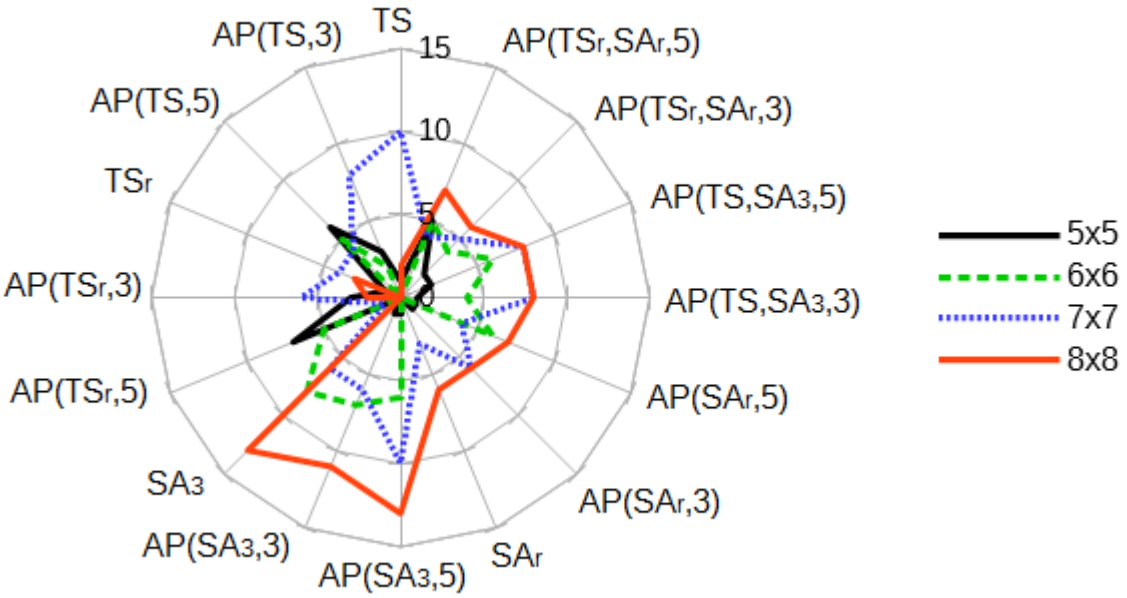


Figure 3.1: Number of wins per algorithm and test problem.

of TS_r and its related homogeneous APs, followed by the blocks of SA_3 and SA_r, along with the corresponding homogeneous APs. The last block for the 5×5 problem reports results for the heterogeneous APs that combine fixed-parameter and random-parameter variants of the constituent algorithms.

The dominating approaches in terms of wins in the pairwise Wilcoxon rank-sum tests are marked with a star "*" symbol in Tables 3.5 and 3.6. In addition, all approaches competed against each other and the two best-performing algorithms in terms of wins in the rank-sum tests are marked with the corresponding superscripts. For example, for the 5×5 problem in Table 3.5, $AP(TS_r, 5)$ was ranked first, while $AP(TS, 5)$ was ranked second among the best-performing approaches. Similarly, SA_3 and $AP(SA_3, 3)$ were the best algorithms for the 6×6 problem. Note that starred algorithms within the same block had equal numbers of wins among them. The total number of wins per algorithm and test problem are also graphically illustrated in Fig. 3.1.

Diverse conclusions can be derived from the reported results. In the 5×5 problem, homogeneous APs appear to be superior to their sequential counterparts in terms of the reported median solution values or time efficiency (in case of equal quality). The heterogeneous APs appear to consistently achieve high-quality median solutions and high time efficiency compared to the baseline algorithms. Similar conclusions are

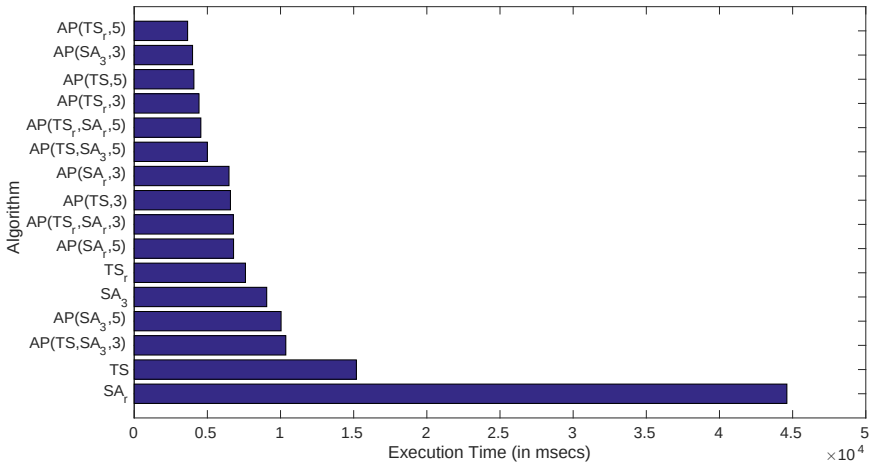
Table 3.7: Median numbers of function evaluations (FEV) required by the APs for all test problems.

Algorithm	5×5			6×6			7×7			8×8		
	NL	AC	FEV	NL	AC	FEV	NL	AC	FEV	NL	AC	FEV
$AP(TS, 3)$	10	16	27698	20	32	2172	46	48	641354	98	88	65297
$AP(TS, 5)$	10	16	16857	20	32	2030	46	56	16516	98	88	66930
$AP(TS_r, 3)$	10	16	19026	20	32	2112	46	48	471266	98	88	49958
$AP(TS_r, 5)$	10	16	14583	20	32	2006	46	56	40686	98	88	35425
$AP(SA_3, 3)$	10	16	16106	20	32	1090	46	48	878650	100	88	6032
$AP(SA_3, 5)$	10	16	43412	20	32	1048	46	48	237382	100	80	111113
$AP(SA_r, 3)$	10	16	28772	20	32	2375	46	56	1743	100	88	54970
$AP(SA_r, 5)$	10	16	30124	20	32	1575	46	56	7684	100	88	57190
$AP(TS, SA_3, 3)$	10	16	43742	20	32	1074	46	56	6019	100	88	36013
$AP(TS, SA_3, 5)$	10	16	21094	20	32	1024	46	48	351183	100	88	26208
$AP(TS_r, SA_r, 3)$	10	16	28423	20	32	2039	46	56	7626	100	88	131924
$AP(TS_r, SA_r, 5)$	10	16	19769	20	32	2010	46	56	11140	100	88	58931

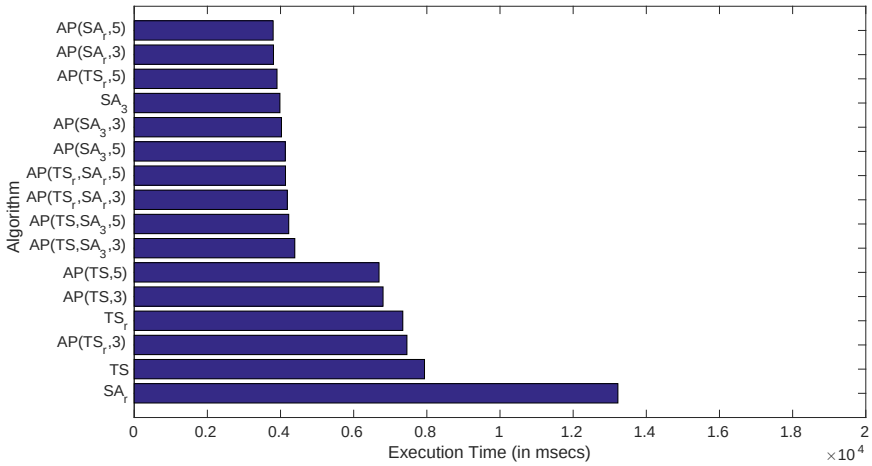
derived for the reported best solutions.

Another interesting observation is that APs with 5 nodes outperformed those with 3 nodes. This is an empirical confirmation of the rationale behind the use of multiple algorithms in APs. Moreover, the overall best AP in the experiments was the $AP(TS_r, 5)$, which consists of four instances of the TS algorithm with random parameters, followed by the homogeneous $AP(TS, 5)$ that consists of four fixed-parameter instances of TS. Similar performance was attained for the 6×6 problem. The APs habitually outperformed the sequential algorithms. However, in this case the SA_3 approach was ranked first, followed by $AP(SA_3, 5)$. Clearly, the previously neglected SA_3 algorithm appeared to form efficient AP schemes in this case.

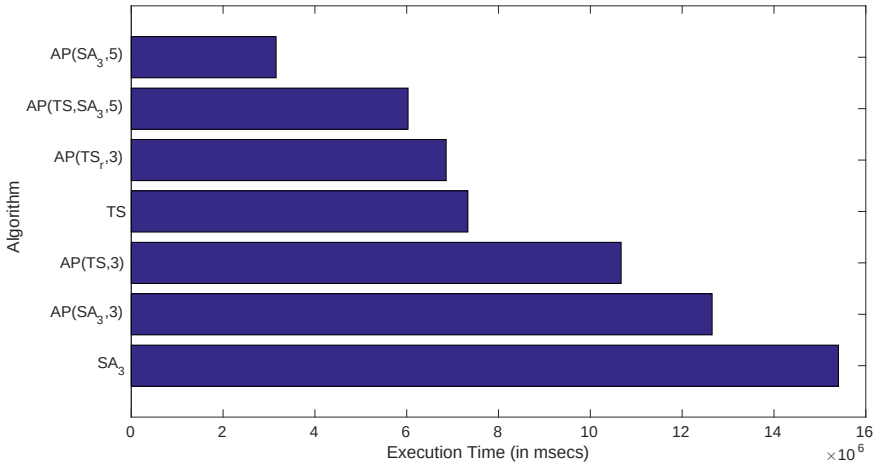
In the 7×7 and 8×8 problems reported in Table 3.6, enhanced performance was achieved for the TS-based APs with 3 nodes. This can be attributed to the exponentially increasing difficulty of the problem as the size of the S-box increases linearly, which augments the time requirements of the algorithms for the detection



(a) 5×5 test problem.

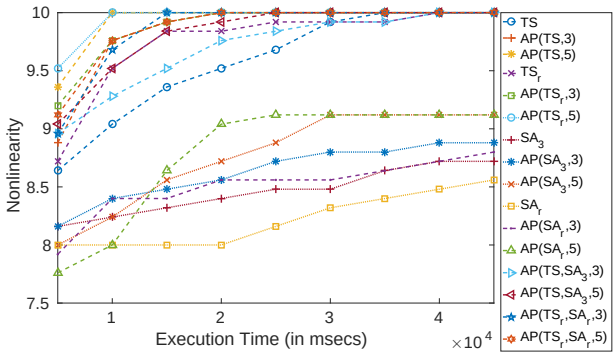


(b) 6×6 test problem.

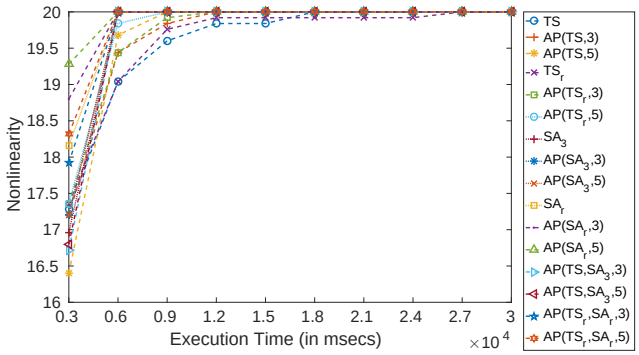


(c) 7×7 test problem.

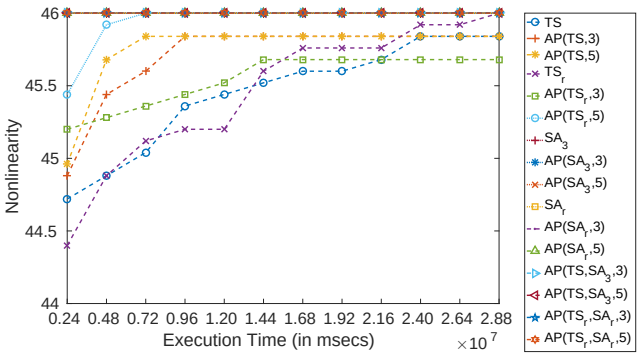
Figure 3.2: Required execution time to achieve the reported median solutions.



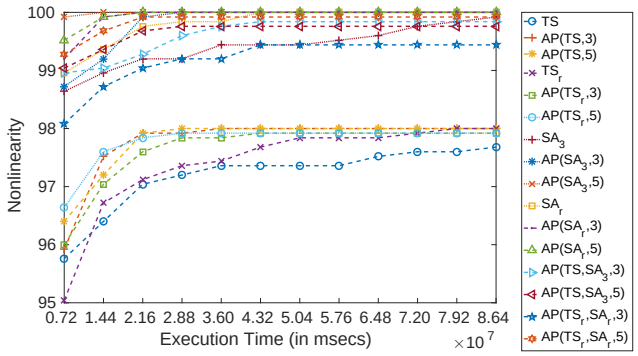
(a) 5×5 test problem.



(b) 6×6 test problem.

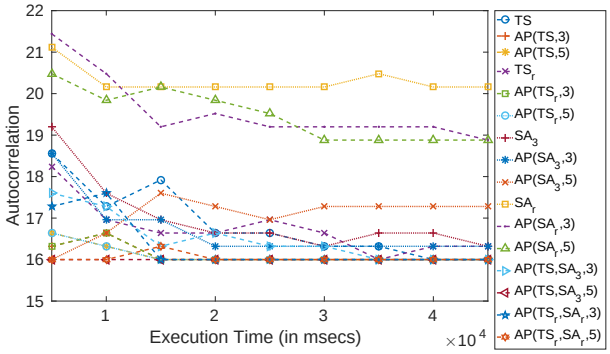


(c) 7×7 test problem.

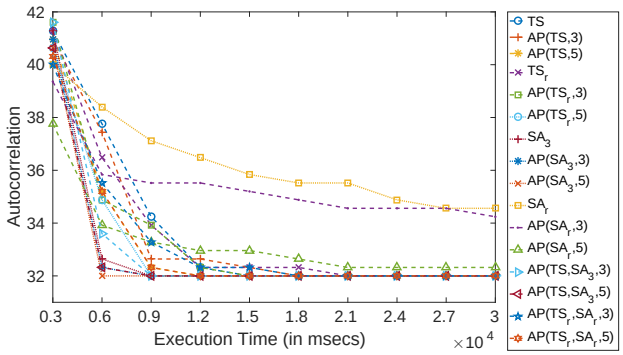


(d) 8×8 test problem.

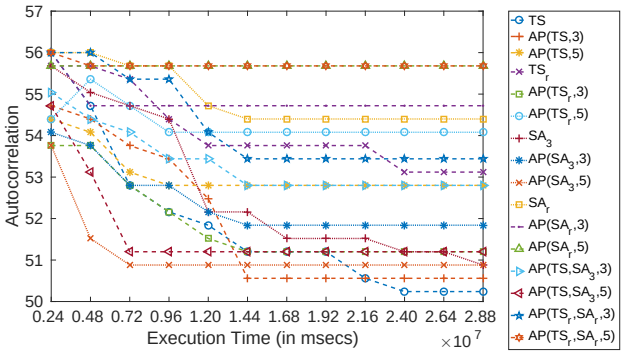
Figure 3.3: Nonlinearity of the best solution during the algorithms' execution.



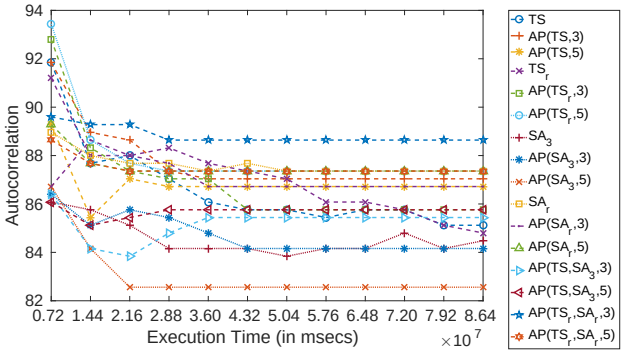
(a) 5×5 test problem.



(b) 6×6 test problem.



(c) 7×7 test problem.



(d) 8×8 test problem.

Figure 3.4: Autocorrelation of the best solution during the algorithms' execution.

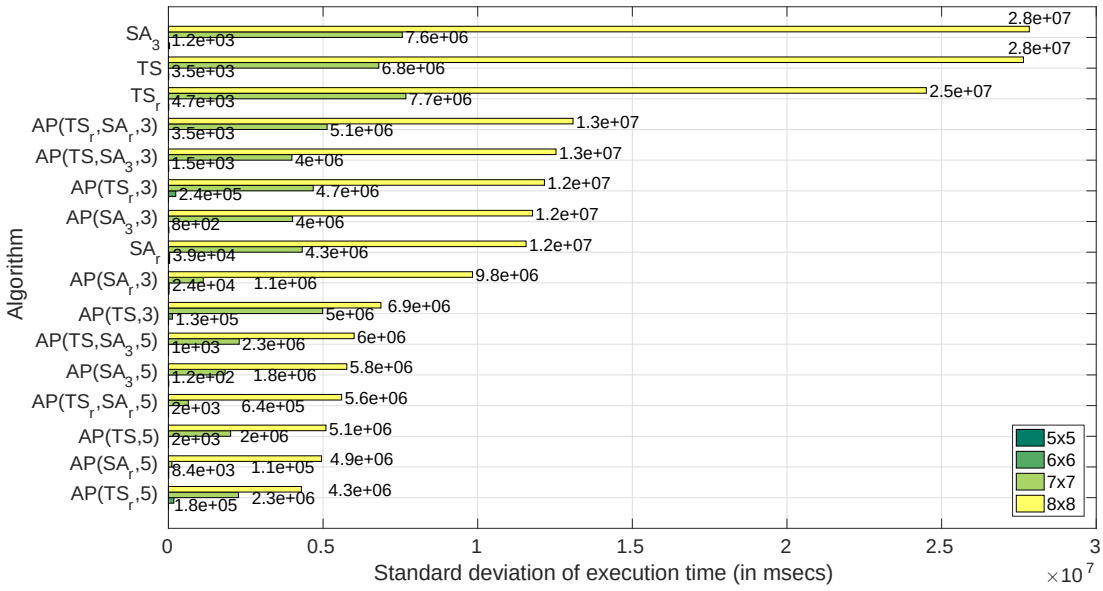


Figure 3.5: Standard deviation of running time for all algorithms and problems.

of high-quality solutions. The constituent algorithms of the APs with 3 nodes were assigned more time than the algorithms in APs with 5 nodes (recall that the total computational budget is the same for all APs). Thus, they could probe the search space more thoroughly, providing better solutions. This is a strong indication that the benefits from the use of APs decline if the running time per constituent algorithm is inadequate for deploying its dynamics.

In contrast to TS-based APs, the SA-based APs with 5 nodes outperformed the ones with the 3 nodes in most test problems. This suggests that SA-based APs exploited the additional nodes more efficiently. In particular, the inherent randomization of SA proved to enhance the exploration dynamics of the APs, which is beneficial in challenging problems. In fact, for the 8×8 test problem, $AP(SA_3, 5)$ was the best AP with marginal difference from the sequential SA_3 approach, which was ranked first mostly due to its rapid detection of the best solutions. This verifies the necessity of providing adequate running time to the algorithms when dimension increases. Note that $AP(SA_3, 5)$ was the variant that achieved the best median solution as can be seen in Table 3.6.

Overall, the considered APs exhibited competitive performance in all test problems. They were able to regularly achieve high-quality solutions and time efficiency. This is clearly communicated from Fig. 3.2, where the algorithms that achieved the best

Table 3.8: Maximum nonlinearity values achieved by the considered algorithms against other approaches in the literature.

Prob.	Curr. work	Picek *et al.* [96]	Laskari *et al.* [86]	Clark *et al.* [88]	Millan *et al.* [97]
5×5	10	10	10	10	10
6×6	22	22	20	22	20
7×7	46	48	46	48	46
8×8	100	104	98	102	100

medians per problem are ranked according to their running times. In all cases, APs occupy higher positions due to their high time-efficiency. The 8×8 test problem is excluded because only the $AP(SA_3, 5)$ variant achieved the best median solution. Another interesting observation is that APs based on SA_3 appear to be the most efficient. This implies that SA-based APs gain more benefits from fixed-parameter settings.

As previously mentioned, in parallel computing environments the running time of an algorithm constitutes an established measure of efficiency. However, for completeness purposes, the median of the number of function evaluations required by the APs for each test problem is reported in Table 3.7. In general, the observed performance with respect to function evaluations is aligned with the previous findings based on the running time of the algorithms.

For better understanding of the algorithms, the progress of their NL value during their execution, averaged over the 25 experiments, is depicted in Fig. 3.3. The corresponding AC values are illustrated in Fig. 3.4. Specifically, the running time was divided into equal segments and the achieved average NL and AC was plotted per segment. The curves yield apparent NL differences between the sequential algorithms and the APs. Also, it can be seen that AC is not monotonically decreasing but rather fluctuates. This is anticipated since AC is a lower-priority objective compelled to follow the changes in NL.

Another issue of interest in APs is the reduction of risk, which is defined in terms of the standard deviation of the AP's time efficiency. For the employed algorithms this can be interpreted as the standard deviation of the running time. Figure 3.5 reveals the standard deviations of running times for all algorithms and problems. Apparently APs achieved smaller standard deviations, especially for the harder problems, thereby

Table 3.9: Minimum autocorrelation values achieved by the considered algorithms against other approaches in the literature.

Problem	Current work	Laskari *et al.* [86]	Clark *et al.* [88]
5×5	16	16	16
6×6	32	32	32
7×7	48	56	48
8×8	80	80	80

revealing their risk-reduction properties.

Finally, for completeness purpose, Tables 3.8 and 3.9 report the best NL and AC values achieved by the proposed algorithms within the provided strict time budgets, against other computational methods from the literature. Specifically, the best results of Picek *et al.* [96], Laskari *et al.* [86], Clark *et al.* [88], and Millan *et al.* [97] are reported. Regarding the NL values, some differences mainly for the most difficult problems are witnessed. In particular, for the 7×7 and 8×8 problems, the cost functions proposed in [96] and [88] resulted in solutions of high NL values. The use of these cost functions constitutes an interesting direction for future research, as they can be incorporated in the proposed APs to boost their performance.

For the 8×8 test problem, some additional works can be found in the literature. In [98] the proposed approach was based on a finite field inversion method and managed to detect 8×8 S-boxes with NL value equal to 106. In [99], a reverse genetic algorithm with initial population of AES affine equivalent S-boxes was used, succeeding to detect solutions with NL value 112. Finally, in [100] the proposed approach exploited important theoretical findings on power mappings. The employed method managed to detect S-boxes with NL values equal to 112.

Regarding autocorrelation, Table 3.9 shows that the other approaches achieved similar results to the studied APs. The main difference is observed in the 7×7 problem, where the proposed approach achieved the best AC along with the approach of [88].

3.5 Application in Traffic Light Scheduling

Urban traffic planning is a fertile research area in smart cities to improve efficiency, environmental care, and safety, since traffic jams and congestion are one of the biggest

sources of pollution and noise. Traffic lights play an important role in solving these problems as they control the flow of the vehicular network in the city. These devices are positioned at road intersections, pedestrian crossings, and other locations to control conflicting flows of traffic and avoid possible accidents. At each intersection, all traffic lights are synchronized to carry out a sequence of valid phases periodically. Each phase consists of a combination of light color states and determines the time span that vehicles are allowed to use the roadway. The assignment of the time span for each phase in the phase sequence of all intersections at an urban area is called the *traffic light plan*.

Finding an optimal traffic light plan is crucial for reducing the number of stops for red lights, thereby minimizing the travel time of vehicles through the road network. Intuitive examples are the well known *green waves*, which facilitate a continuous traffic flow in one main direction. Reducing the travel time prevents drivers from time-loss and late arrivals, which induce economic impact. Also, it helps reducing the fuel consumption and CO_2 emissions while the vehicle is stopped at red lights.

The many obvious benefits of optimal traffic light scheduling have motivated a growing field of research related to automatic traffic control signals. A number of industrial solutions have been proposed for this problem, such as the Cross Zlín [101] and ATC [102]. These solutions focus on the real-time configuration of a single traffic light junction. Also, they require the existence of infrastructures that provide online information about the changing traffic situations. Here, a different direction is followed, because the increasing number of vehicles requires the transition from the local control of a single intersection to a holistic approach considering a large urban area, and because optimizing the existing traditional traffic lights rests still much unexplored.

This holistic approach is only possible by using advanced computational resources and techniques due to the complexity of the problem, which is twofold. Firstly, the problem usually has huge search spaces. For example, a simple intersection with 8 traffic light phases represents 55^8 (more than 8.3×10^{14}) possible solutions. Secondly, there is no closed mathematical formulation of the problem to assess the quality of candidate traffic lights configurations. Thus, the utilization of simulators is necessary. However, simulators are usually time-consuming, typically requiring from seconds up to a few minutes per simulation. Hence, new and efficient algorithmic tools become indispensable in real-world scenarios.

Metaheuristics have been widely used to tackle traffic light scheduling problems. Early attempts were mostly based on Genetic Algorithms (GAs). The first study appeared in [103] where a GA was employed to optimize the timing of the traffic light cycles of nine intersections located in the city of Chicago (IL), USA. The authors proposed further investigation of GAs on larger problem instances. In [104] the authors studied the reactions of drivers to changes of the traffic lights timing. Their approach used a GA and it was evaluated on a case study of the city of Chester, UK. A GA was used also in [105] to optimize traffic light cycle programs. In this work, the authors assumed that the traffic lights timing of each intersection works independently of other intersections. They tested their approach on a test case of a commercial area of the city of Santa Cruz, Spain. Another work involving the application of GAs on a traffic light scheduling problem appeared in [106]. The proposed approach tackled the problem of controlling the traffic lights timing for vehicles and pedestrians under a dynamic traffic load situation.

Recently, there has been a number of works focusing on the application of the Particle Swarm Optimization (PSO) algorithm on finding optimal traffic light schedules. In [107] PSO was employed to train a fuzzy logic controller installed at each intersection. Specifically, PSO was used to train the membership functions and the rules of the controller, targeting to detect the optimal duration of the green signal for each phase of the traffic lights. In [108] the authors proposed a PSO algorithm to discover isolation niches on a traffic light scheduling problem. The proposed approach was evaluated on a small problem instance consisting of an one-way road with two intersections. This work focused on the potential of the algorithm to maintain its diversity without trying to gain deep insight on the problem at hand.

A multi-objective PSO algorithm that employed a predictive model control strategy to optimize traffic light cycle schedules was studied in [109]. The proposed algorithm was evaluated on an urban network consisting of 16 intersections and 51 links. In [21, 50] PSO was proposed for computing the optimal traffic light cycle programs. The main objectives of these works were the maximization of the number of vehicles that reach their destinations, as well as the minimization of the total trip time of the vehicles. The evaluation of the cycle programs was based on the popular microscopic SUMO simulator. The proposed algorithm was assessed on small/medium urban areas located in the cities of Málaga and Sevilla (Spain), and in Bahía Blanca (Argentina).

More recently, PSO algorithms were used for detecting traffic light cycle programs,

aiming at the reduction of fuel consumption and vehicular emissions in metropolitan areas [110, 111]. These approaches followed a traffic emission model standardized by the European Union reference framework. The proposed algorithm achieved significant improvements in the considered objectives compared to traffic light cycle programs designed by experts.

This section focuses on various aspects of the traffic light scheduling problem. Firstly, an approach based on the established Differential Evolution algorithm [38] is proposed. Several variants of the algorithm are tested to distinguish the most competitive ones. Also, parallel APs are studied on realistic problem instances. Finally, the proposed approach is evaluated on two large, real-world urban scenarios for the cities of Málaga (Spain) and Paris (France). The latter involves the optimization of more than 375 traffic lights, while the largest instances tackled in previous works [21, 50] were restricted to cases of up to 190 traffic lights.

3.5.1 Problem Formulation

Here, the mathematical model presented in [21,50] is used for the traffic light scheduling problem. The considered problem has multiple objectives. The first objective is to maximize the number V_{R} of vehicles that reach their destination or, equivalently, minimize the number V_{NR} of vehicles that do not arrive at their destination during a given simulation time T_{sim}. A second objective is to minimize the total trip time T_{trip} of the vehicles, which is equal to the sum of the trip times of all vehicles. The trip time refers to the simulation time individually consumed by each vehicle to arrive at its destination. Evidently, vehicles that fail to reach their destination consume the whole simulation time.

A third objective is to minimize the sum of stop and wait times of all vehicles, denoted by T_{sw}. The stop and wait time refers to the overall time that each vehicle individually has to stop at those intersections that have traffic lights in red color, thereby delaying its trip. Another objective is the maximization of the ratio P of green and red colors in each phase state of all intersections, which is defined as follows,

$$P = \sum_{i=0}^{intr} \sum_{j=0}^{ph} d_{i,j} \frac{g_{i,j}}{r_{i,j}}, \tag{3.6}$$

where $intr$ denotes the number of all intersections; ph denotes the number of all phases; and $g_{i,j}$, $r_{i,j}$, denote the number of green and red signal colors, respectively,

at intersection i and phase state j, with duration $d_{i,j}$. The minimum value of $r_{i,j}$ is set to 1 in order to prevent division by zero.

The intuition behind Eq. (3.6) lies in the effort to promote green traffic signals at intersections overburdened by traffic flow, and red traffic signals at intersections where low traffic flow is observed. Traffic lights with extended times in red color may overwhelm not only the intersection where they are located, but also neighboring intersections, creating extensive traffic flow problems in the city.

All objectives are combined into a single-objective function formulated as follows,

$$f_{\text{obj}} = \frac{T_{\text{trip}} + T_{\text{sw}} + V_{\text{NR}}\, T_{\text{sim}}}{V_{\text{R}}^2 + P}. \tag{3.7}$$

It shall be noted that the quantities under minimization are placed in the numerator of Eq. (3.7), whereas the ones under maximization are placed in the denominator. Therefore, the problem is a global minimization task. The term V_{R} is squared to prioritize over all other terms as it represents the main (first) objective. Also, the number of non-arriving vehicles V_{NR} is multiplied by the simulation time T_{sim} to induce a penalization for this undersided scenario.

3.5.2 Employed algorithms

The considered problem is tackled through the Differential Evolution and Particle Swarm Optimization algorithms. Details about the algorithms can be found in Sections 2.6 and 2.7, respectively. Regarding the Particle Swarm Optimization algorithm, Eq. (2.16) is employed for updating the velocity of the particles. Additionally, the inertia weight changes linearly during the optimization according to Eq. (2.17) and the velocity is updated according to Eq. (2.18), which is used especially for combinatorial problems. Apart from the sequential algorithms, the Parallel AP model that is exposed in Section 3.3.2 is employed. In this section, the employed SUMO simulator is briefly described, and details are provided about the solution encoding employed by the algorithms.

3.5.2.1 SUMO: Simulator of Urban Mobility

As already mentioned, simulation plays a crucial role in the assessment of optimization algorithms for the traffic light scheduling problem. *Simulator of Urban MObility* (SUMO) [112] constitutes a well established tool for this purpose. SUMO is a popular,

open-source, highly-portable micro-simulator. It requires a number of input files in XML format that contain information about the road scenarios to be simulated.

Specifically, there is a network file `.net.xml` that stores information about the form of the map, namely nodes, edges, and connection links among them. A route file `.rou.xml` holds information about the journey of a vehicle from a starting point (starting vertex) to an ending point (destination vertex) as well as all intermediate points visited by the vehicle. The `.add.xml` files contain additional information about the map or the traffic lights. Finally, the `.tripinfo.xml` contains information used to evaluate traffic light cycle programs. For instance, it contains information about the vehicles' departure and arrival times that are used to compute each vehicle's total trip time.

A candidate solution vector (traffic light cycle program) is forwarded to SUMO, which in turn computes its objective value after a simulation procedure. SUMO starts the simulation after transforming the input vector and the information contained in its data files into real-world objects such as vehicles, intersections, traffic lights, etc. When the simulation is completed, SUMO returns all the information that is necessary for the computation of the objective function value of the specific traffic light cycle program for the city. Note that a single SUMO call is adequate for computing the corresponding function value because SUMO works in a deterministic way. Deterministic traffic simulators are preferable to stochastic ones as they acquire similar results at considerably lower computational cost [105].

3.5.2.2 Solution Enconding

Similarly to previous works [21,50], each direction component of the solution vector represents a phase duration of one state of the traffic lights of a particular intersection. Specifically, each state of each phase duration is encoded via an integer number, belonging to a simple vector of integers. The proposed encoding is desirable for various reasons. Firstly, the SUMO simulator itself employs integer numbers to represent the discrete time steps of the simulation procedure. Therefore, the mapping between the phase duration used by the data structures of SUMO and the solution vector is simplified. Secondly, the employed population-based algorithms can take into consideration the interdependence of variables, representing traffic lights of the same intersection as well as traffic lights of different intersections that exhibit high proximity.

Regarding the initialization of DE and PSO, the candidate solutions are initialized

Table 3.10: Málaga and Paris problem instances.

Problem instance	Number of traffic logics	Number of traffic lights	Number of vehicles
Málaga	56	190	1200
Paris	70	378	1200

Table 3.11: Parameters of the Algorithms.

Alg.	Param.	Description	Value(s)
DE	N	Population size	50
	F	Differential weight	{0.5, 0.7}
	CR	Crossover probability	{0.05, 0.1}
PSO	N	Population size	100
	c_1	Cognitive parameter	2.05
	c_2	Social parameter	2.05
	ω_{min}	Min. inertia weight	0.1
	ω_{max}	Max. inertia weight	0.5

in the range $[5, 60]$. Each value in this range represents the time units (in seconds) that the corresponding traffic light will keep the same signal color, in the case of red and green colors. As for the amber signal color, its time interval is set to a constant value (4 seconds), which remains unchanged during the optimization. The proposed interval is selected according to various real-world traffic light scenarios provided by the City Council of Málaga (Spain).

The two algorithms are typically used in continuous optimization, where solutions consist of real numbers. The considered problem belongs to the class of combinatorial optimization problems. In order to tackle it, both DE and PSO use a proper rounding of the solution vector at each iteration, thus converting it to a vector of integer values. This conversion is necessary also for implementation purposes, since the SUMO simulator admits only integer input values.

3.5.3 Experimental Results

The proposed parallel AP approach is particularly suitable for tackling real-world urban scenarios. For this reason, the proposed approach was evaluated on two close-

Table 3.12: Results of DE algorithm on the Málaga instance.

OP	F	CR	Mean	StD
DE1	0.5	0.05	0.4908	0.0098
	0.5	0.1	0.4895	0.0085
	0.7	0.05	0.5010	0.0073
	0.7	0.1	0.4992	0.0049
DE2	0.5	0.05	0.5015	0.0096
	0.5	0.1	0.5051	0.0079
	0.7	0.05	0.5141	0.0115
	0.7	0.1	0.5150	0.0070
DE3	0.5	0.05	0.5030	0.0098
	0.5	**0.1**	**0.4809**	**0.0080**
	0.7	0.05	0.4979	0.0062
	0.7	0.1	0.4979	0.0079
DE4	0.5	0.05	0.4988	0.0091
	0.5	0.1	0.4986	0.0114
	0.7	0.05	0.5058	0.0091
	0.7	0.1	0.5250	0.0134
DE5	0.5	0.05	0.5095	0.0138
	0.5	0.1	0.5178	0.0100
	0.7	0.05	0.5188	0.0101
	0.7	0.1	0.5408	0.0091

to-reality problem cases consisting of large metropolitan areas located in Málaga (Spain) and Paris (France). The considered problem cases were created by extracting information from real digital maps. The first problem instance involves the optimization of 190 traffic lights, whereas the second one contains 378 traffic lights. Previous works [21,50] tackled problem instances with no more than 190 traffic lights. The dimension of each problem case is equal to the number of the traffic lights it comprises. More details on the used instances can be found in Table 3.10.

The experimental evaluation was conducted on the `saw` cluster of the Sharcnet consortium. The parallel implementations were based on the OpenMPI project. Regarding the simulation procedure, each vehicle started its own trip from a starting

Table 3.13: Results of DE algorithm on the Paris instance.

OP	F	CR	Mean	StD
DE1	0.5	0.05	0.7420	0.0117
	0.5	0.1	0.7487	0.0096
	0.7	0.05	0.7502	0.0109
	0.7	**0.1**	**0.7332**	**0.0052**
DE2	0.5	0.05	0.7684	0.0117
	0.5	0.1	0.7681	0.0193
	0.7	0.05	0.7686	0.0097
	0.7	0.1	0.7603	0.0098
DE3	0.5	0.05	0.7764	0.0125
	0.5	0.1	0.7627	0.0600
	0.7	0.05	0.7602	0.0092
	0.7	0.1	0.7366	0.0111
DE4	0.5	0.05	0.7466	0.0111
	0.5	0.1	0.7344	0.0088
	0.7	0.05	0.7624	0.0063
	0.7	0.1	0.7521	0.0092
DE5	0.5	0.05	0.7732	0.0097
	0.5	0.1	0.7675	0.0105
	0.7	0.05	0.7759	0.0140
	0.7	0.1	0.7878	0.0085

point to a destination with a maximum speed of 50 km/h. This speed limit is typical in urban areas. The simulation time was set to 2200 seconds (iterations of microsimulation) for the Málaga instance and 3400 seconds for the Paris instance, as it consisted of a larger number of traffic lights. The simulation was conducted by executing the traffic simulator SUMO release 0.19.0 for Linux.

The first objective of the experiments was to compare the different variants of the DE algorithm on both problem instances. The population size of the algorithm was set to 50 individuals each one conducting 600 iterations, resulting in a total computational budget of 30000 function evaluations. Each function value was obtained by a single simulation run of SUMO. Two distinct values for the differential weight parameter

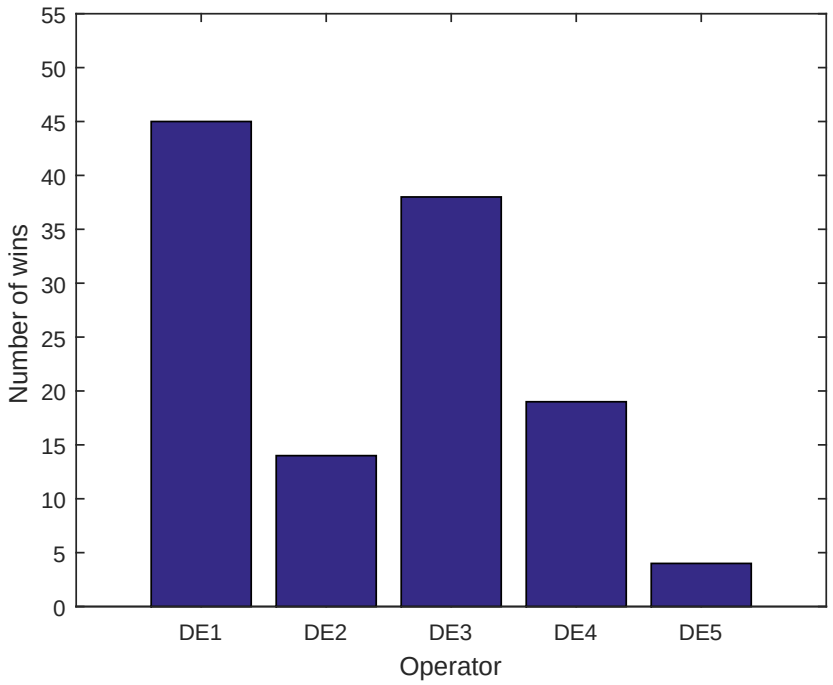


Figure 3.6: Comparison of DE operators on the Málaga instance.

$F \in \{0.5, 0.7\}$ are considered along with the crossover probability $CR \in \{0.05, 0.1\}$. Details for the parameterization of the DE algorithm are given in Table 3.11.

For each problem case and DE variant, 10 independent experiments were conducted and the best detected solution was recorded. Each independent experiment required 10 to 13 hours of simulation, and it was terminated as soon as the maximum number of function evaluations was attained. For each problem instance and algorithmic variant, the average of the obtained solution values and the standard deviation are reported in Tables 3.12 and 3.13. The best approach per problem case, i.e., the one with the lowest average function value is boldfaced.

For comparison purposes, pairwise Wilcoxon rank-sum tests were performed between all pairs of the considered approaches. For each variant, the number of *wins* was counted, i.e., the number of comparisons that it outperformed another variant (achieved lower mean function value) with significance level 95%.

The approaches that appear boldfaced in Tables 3.12 and 3.13 exhibited the highest number of wins. Specifically, the best approach on the Málaga instance, namely the one that used the DE3 operator and parameter values $F = 0.5$, $CR = 0.1$, won 18 different variants. The best approach on the Paris instance, namely the one that employed the DE1 operator and parameter values $F = 0.7$, $CR = 0.1$, won 17 dif-

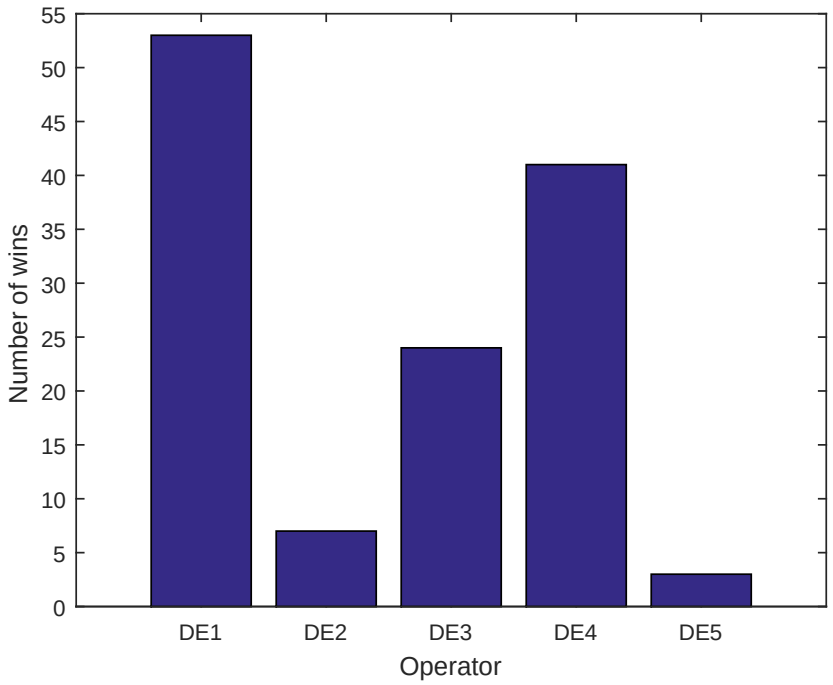


Figure 3.7: Comparison of DE operators on the Paris instance.

ferent variants. Notice that the total number of comparisons among the different DE variants is equal to 19.

The number of rank-sum wins per DE variant is graphically illustrated in Fig. 3.6 for the Málaga case and Fig. 3.7 for the Paris case. In both cases, there is an evident superiority of the DE1 operator, achieving 45 wins for the Málaga case and 53 wins for the Paris case. Also, DE3 and DE4 operators achieved high numbers of wins compared to DE2 and DE5 ones, which exhibited the worst performance. This fact is attributed to the global information incorporated into the three best-performing operators. In general, experimental evidence suggests that it is beneficial to use exploitation-oriented operators in order to rapidly achieve a suboptimal solution under restricted computational budget. Operators DE1, DE3, and DE4 fulfill this necessity since they take advantage of the globally best solution, which rapidly leads the population to suboptimal solutions.

In a second round of experiments, the best-performing DE variant for each problem competed against the global best model of the PSO algorithm. Details about the employed PSO were given in Section 2.7. Regarding the parameterization of PSO, the swarm size was set to 100 and the cognitive and social constants were equal to 2.05. The PSO algorithm used an initial inertia weight $\omega_{\max} = 0.5$ that was linearly

Table 3.14: Results of PSO algorithm on both problem instances.

Problem instance	Mean	StD
Málaga	0.5081	0.0122
Paris	0.7724	0.0083

decreased to the final value $\omega_{\text{min}} = 0.1$. Recall that the inertia weight adjusts the exploitation/exploration capabilities of PSO. The selected parameter values are also reported in Table 3.11.

The obtained average values and standard deviations of PSO are reported in Table 3.14. Evidently, the best-performing DE variant surmounts the specific PSO approach on each problem case. For the Málaga instance, the best DE approach achieved mean function value equal to 0.4809 in contrast to the value 0.5081 achieved by PSO. Similar performance is observed for the Paris instance where DE achieved an average value 0.7332 whereas PSO achieved 0.7724. Although the differences between these values seem marginal, it shall be underlined that the improvement in the function value from one iteration to another during the optimization was usually observed in the second or third decimal digit using both DE and PSO algorithms. Nevertheless, these small fitness differences correspond to important differences in the solution vectors.

In the third round of the experiments, the proposed parallel AP approach was evaluated. The best-performing algorithm per problem was selected and implemented according to the parallelization of population approach. Specifically, the DE variant was employed that uses DE3 with $F = 0.5$, $CR = 0.1$, for the Málaga problem and the one that uses DE1 with $F = 0.7$, $CR = 0.1$, for the Paris problem. The function evaluations were equally distributed among 2 and 3 nodes in an effort to speed up the computation process. Details about the proposed parallel AP model are given in Section 3.3.2. For each problem and algorithmic variant, the execution time required to achieve a targeted objective function value, f_{trg} was recorded. For the Málaga problem, this value was set to 0.50, while for the Paris problem it was equal to 0.74. The specific values of f_{trg} were achievable at each independent experiment. As soon as the algorithm reached the targeted function value or a better one, it terminated its execution.

A significant gain metric for parallel implementations is the *speedup*, which com-

Table 3.15: Results of the parallel approach.

Problem	Nodes	Time (hours)	Speedup
Málaga	2	1.90	–
	3	1.41	1.35
	4	1.10	1.73
Paris	2	9.28	–
	3	5.79	1.60
	4	5.29	1.75

putes the ratio between sequential and corresponding parallel execution times. Several speedup definitions are used in the literature. Here, the *weak speedup* definition [79] is used. The weak speedup s_m of a parallel algorithm using m processors is defined as follows,

$$s_m = \frac{T_{\text{seq}}}{T_m}, \tag{3.8}$$

where T_{seq} is the execution time of the sequential algorithm executed on a single processor, and T_m is the execution time of the parallel algorithm running on m processors.

Table 3.15 reports the execution time (in hours) and the achieved speedup for the best serial algorithm per problem parallelized across a number of nodes. Note that the reported number of nodes includes the master node, which runs the algorithm, and the slave nodes that conduct the function evaluations through parallel SUMO calls. Thus, 2 nodes refer to the serial execution of the algorithm (1 master and 1 slave) and it was used as a baseline for comparisons with the parallel AP approach that assumes higher number of nodes.

For the Málaga instance, is is observed that the sequential algorithm required 1.90 hours to achieve the considered f_{trg}. The approaches that exploited 3 and 4 nodes achieved lower execution times, namely 1.41 and 1.10 hours, respectively. In terms of speedup, Table 3.15 shows a value of 1.35 for 3 nodes. The speedup value was further increased when 4 nodes were used. This result was expected since the function evaluations were distributed among a larger number of nodes, thus enhancing the performance of the algorithm. However, is is noticed that the percentage of increase in speedup values from 2 to 3 nodes is lower than the one from 3 to 4 nodes.

For the Paris instance, it is observed that the parallel AP approach was even

more beneficial. Specifically, the sequential algorithm required 9.28 hours, whereas the parallel AP approach using 3 and 4 nodes needed 5.79 and 5.29 hours, respectively. The corresponding speedup values were 1.60 for the 3-node case and 1.75 for the 4-node one. The longer execution time needed for the Paris instance is attributed to its problem size, which is almost twice the Málaga one, thereby leading to higher simulation times.

Thus, the proposed parallel AP model has evidently boosted the performance of the employed algorithms. This finding is highly desirable in real-world traffic light scenarios as the considered one. Nevertheless, the effect of the simulation procedure on the acquired results is underlined, since high deviation in the running times needed by the SUMO simulator was noticed.

3.6 Synopsis

In this chapter, algorithm portfolios were introduced as powerful algorithmic schemes to tackle hard optimization problems. In the first part, a thorough literature review was presented related to sequential and parallel AP frameworks. Next, a simple yet efficient parallel AP framework was proposed where the algorithms were executed with no information exchange among them. The proposed AP was demonstrated on the design of bijective S-boxes of high nonlinearity and low autocorrelation. Initially two well studied trajectory methods, namely TS and SA, were applied on the considered problem. TS has never been used to design bijective S-boxes before. Next, the algorithms were incorporated into parallel AP frameworks that exploited the widely used master-slave model. The APs were constructed using either one algorithm (homogeneous case) or both TS and SA (heterogeneous case), and were thoroughly compared with each other and with their constituent algorithms.

Extensive experimental evaluation revealed that the proposed APs can provide S-boxes of better or equal quality with the sequential algorithms, although in significantly less time. SA-based APs that used fixed-parameter settings usually outperformed TS-based ones in harder test problems, while in smaller test problems TS-based APs surmounted SA-based ones. Additionally, TS-based APs achieved smaller standard deviations of running time compared to SA-based ones. In general, the proposed APs achieved smaller deviations in the time required to attain their best solution compared to their sequential versions, especially for the harder problems.

Future research will include the exploration of interactive AP frameworks. Also, more sophisticated formulations of the problem will be explored, along with different cost functions such as the spectrum-based cost function reported in the relevant literature.

Finally, a parallel AP for population-based metaheuristic algorithms was proposed. The AP employed a typical master-slave model and distributed the budget of function evaluations among the slaves. Its use was suggested for expensive optimization problems that spend high amounts of time in function evaluations. The parallel AP was applied on the traffic light scheduling problem. The proposed approach initially involved applying the sequential DE algorithm on the problem at hand. This was the first time that the potential of the DE algorithm was investigated on the considered problem. Next, the DE algorithm was incorporated into a parallel AP to tackle more realistic instances efficiently. The performance of the proposed approach was compared with the PSO algorithm in terms of solution quality. The proposed methods have been assessed on two real-world scenarios, consisting of large metropolitan areas located in Málaga (Spain) and Paris (France).

Chapter 4

New Parallel Trading-based Algorithm Portfolio

4.1 Introduction

In this chapter a novel *parallel trading-based algorithm portfolio* is proposed. The AP adopts a trading-based mechanism that dynamically modifies the allocation of the available execution time among the constituent algorithms. Thus, best-performing algorithms are assigned higher fractions of execution time than the rest, without modifying the AP's total execution time.

The rationale behind the development of APs is based on the concept of *investment*. This implies investment either on the algorithms that comprise the portfolio [12,13] or, in population-based approaches, on the individuals that compose the population [16]. The proposed AP framework introduces a different aspect of investment. Specifically,

its core idea is inspired by stock trading models and assumes a number of algorithms-investors that invest on elite solutions playing the role of stocks, while using execution time as the currency.

4.2 Proposed model

The proposed AP model assumes a number of M metaheuristics that interact with each other during the optimization process, acting as investors. The AP is allocated a fixed budget of total *running time* and employs a master-slave parallelization model where each algorithm is executed on a single slave node. The total running time is equally distributed among the algorithms. Each algorithm divides its own running time into *investment time* (denoted as T_{inv}) and *execution time* (denoted as T_{exec}). The execution time is devoted to the specific algorithm's execution solely. On the other hand, the investment time is consumed by each algorithm for buying solutions discovered by the other algorithms, if it fails to achieve progress.

The master node is responsible for two basic operations. Firstly, it retains an archive of M elite solutions, each one corresponding to the best solution discovered by one of the algorithms of the AP. Secondly, the master node prices each elite solution, assigning a cost (in terms of time) that is demanded for an algorithm to acquire it. For this purpose, the master sorts the solutions in descending order with respect to their objective values. Then, the price of each solution is defined in terms of its corresponding position ρ_i in the ranking, i.e.,

$$C_i = \frac{\rho_i \times BC}{M}, \tag{4.1}$$

where $BC = \beta T_{\text{inv}}$ is a base cost, and β is a constant that takes values in $[0, 1]$. The parameter β tunes the algorithm's elitism. Clearly, high values of β limit the number of elite solutions an algorithm can buy throughout the optimization process.

The slave nodes communicate with each other via the exchange of asynchronous messages through the master node. Specifically, if slave node i discovers a new elite solution $\mathbf{x}_{\text{new}}$, it sends it to the master node where it replaces its existing one $\mathbf{x}_i$. Interaction among the algorithms takes place when an algorithm cannot improve its own elite solution for an amount of time T_{noimp}. In this case, the master node acts as trading broker that applies a solution selection policy to help the algorithm that would like to make a purchase (buyer algorithm) make the most profitable

Algorithm 4.1 Trading-based algorithm portfolio: master node

Input: Execution time (T_{exec}), Investment time (T_{inv}), Constant β

Output: Best detected solution

```
solFound ← False
while (NOT solFound) do
    wait for requests from slaves
    if (slave i requests to send a new elite solution x_new) then
        replace x_i with x_new in the archive of elite solutions
    else if (slave i requests to buy) then
        sort elite solutions in descending order w.r.t. their objective values
        price elite solutions
        select elite solution x_j with cost C_j, j ≠ i with the maximum ROI
        increase T_exec of slave j by C_j
        decrease T_inv of slave i by C_j
        send elite solution x_j to slave i
    end if
    if (x_i is global optimum) then
        solFound ← True
    end if
end while
```

investment. In particular, the solution selection mechanism proposes elite solutions to the byer algorithm based on the *Return On Investment* (ROI) index. ROI comes from trading theory and it is defined as follows,

$$ROI_j = \frac{f - f_j}{C_j}, \quad j \in \{1, 2, \ldots, M\}, \tag{4.2}$$

where f denotes the objective value of the buyer's best solution, f_j denotes the objective value of the seller's elite solution, and C_j is the assigned price.

Among the possible elite solutions, the algorithm opts to buy the solution that maximizes the ROI index and has better objective value than its own one. If the buyer algorithm decides to acquire the j-th solution $\mathbf{x}_j$, then it rewards the seller algorithm with an amount of time equal to C_j. Specifically, the byer algorithm reduces its own investment time by C_j, whereas the seller algorithm extends its own execution time by C_j. This way the better-performing algorithms are expected to gain more running time than the rest as they sell solutions more often. Note that the total running time allocated to the AP at the beginning of the optimization remains constant. Detailed

Algorithm 4.2 Trading-based algorithm portfolio: slave nodes

Input: Execution time (T_{exec}), Investment time (T_{inv}), Constant β

Output: Best detected solution

initialize the AP with one algorithm per slave node
allocate to each algorithm execution time T_{exec} and investment time T_{inv}
solFound $\leftarrow$ False
while ($T_{\text{exec}} > 0$) AND (NOT solFound) **do**
    apply the algorithm for some iterations
    send to master node a new elite solution $\mathbf{x}_{\text{new}}$, if detected
    if ($\mathbf{x}_{\text{new}}$ is global optimum) **then**
        solFound $\leftarrow$ True
    else if (no improvement is achieved for T_{noimp} AND $T_{\text{inv}} > 0$) **then**
        request to buy an elite solution from the master node
        if (elite solution $\mathbf{x}_j$ is successfully bought) **then**
            incorporate the solution $\mathbf{x}_j$ in the algorithm
        end if
    end if
end while

pseudocodes of the the master node and the slave nodes procedures are summarized in Algorithm 4.1 and Algorithm 4.2, respectively.

4.3 Application in Circulant Weighing Matrices

Combinatorial matrices are special types of matrices with prescribed combinatorial properties. *Circulant weighing matrices* (CWMs) constitute an important class of combinatorial matrices that has been a fruitful research area for several decades. Their applications are numerous in diverse scientific fields spanning from coding theory where they are used to construct linear codes with good properties [113], to quantum information processing where they are used to improve quantum algorithms [114]. Applications can also be found in other research fields such as statistical experimentation and optical multiplexing [115].

Significant amount of research has been devoted to the investigation of existence of finite or infinite classes of circulant weighing matrices. For this purpose, a number of algebraic methodologies has been used to identify the necessary existence con-

ditions [22, 116–119]. Moreover, recent works have shed light on the classification of specific circulant weighing matrices [120–122]. Complementary to the theoretical approaches, computational methods have been used in cases where the first were unsuccessful. In such cases, the existence problem is transformed into an equivalent discrete minimization problem [123–127] such that the global minimizers of the involved objective function correspond to the desirable matrices.

4.3.1 Problem Formulation

A square $n \times n$ matrix $W = [w_{ij}]$ with entries,

$$w_{ij} \in \{-1, 0, 1\}, \quad i, j \in \{1, 2, \ldots, n\},$$

is called a *circulant weighing matrix of order n and weight k^2*, and denoted as,

$$\text{CW}(n, k^2),$$

if there exists a positive integer $k < n$ such that,

$$W\, W^t = k\, I_n, \tag{4.3}$$

where I_n is the identity matrix of size n, and W^t denotes the transpose matrix of W. Such a matrix has the property that, excluding the first row, each other row is a right cyclic shift of its preceding one. Thus, the matrix can be completely defined solely by its first row.

Various methodologies have been proposed for the systematic detection of circulant weighing matrices of various orders and weights [118, 120, 121, 128]. Beside the theoretical algebraic methods, computational optimization algorithms have been successfully applied. In these cases, the original problem is modeled as a permutation optimization problem where each global minimizer corresponds to a matrix of the desirable type, i.e., it defines the first row of the matrix. The objective function of the optimization problem is based on the *periodic autocorrelation function* [126]. Let,

$$T^n = \{(x_1, x_2, \ldots, x_n),\ x_i \in \{-1, 0, +1\} \text{ for all } i\}$$

be the set of all ternary sequences of length n, and let,

$$\mathbf{x} \in T^n,$$

be the first row that defines a CW(n, k^2) matrix. Then, its periodic autocorrelation function values are defined as,

$$\mathrm{PAF}_{\mathbf{x}}(s) = \sum_{i=1}^{n} x_i \, x_{i+s}, \quad s = 1, 2, \ldots, \left\lceil \frac{n}{2} \right\rceil, \tag{4.4}$$

where $i+s$ is taken modulo n when $i+s > n$. The property of Eq. (4.3) is equivalent to the system composed of the equations,

$$\mathrm{PAF}_{\mathbf{x}}(s) = 0, \quad \forall s.$$

Thus, the admissible sequences that define CW(n, k^2) matrices are the global minimizers of the combinatorial optimization problem,

$$\min_{\mathbf{x} \in T^n} \; f(\mathbf{x}) = \sum_{s=1}^{\left\lceil \frac{n}{2} \right\rceil} \left| \mathrm{PAF}_{\mathbf{x}}(s) \right|. \tag{4.5}$$

Moreover, it is proved that each admissible sequence has exactly,

(a) k^2 non-zero ($+1$ or -1) components,

(b) $k(k+1)/2$ components equal to $+1$, and

(c) $k(k-1)/2$ components equal to -1.

Taking into consideration the fixed number of appearances of 0, $+1$, and -1 in the sequence, the problem becomes a permutation optimization problem. Various approaches including metaheuristics have been extensively used for solving such problems [126, 127]. Experimental evidence has shown that the difficulty level of the problem increases with the length of sequence n and the weight k^2.

4.3.2 Employed Algorithms

Established parallel metaheuristics are used to tackle CWMs problems. The selected algorithms are prevailing in metaheuristics research. Specifically, two trajectory-based methods, namely Tabu Search (TS) [90] and Variable Neighborhood Search (VNS) [33], as well as two population-based methods, namely Differential Evolution (DE) [38] and Particle Swarm Optimization (PSO) [129] are considered. The employed algorithms are described in detail in Chapter 2. The parallelization of these approaches is straightforward in a multiple-trajectory or multiple-population framework, and allows

Table 4.1: Parameter setting for the considered algorithms.

Algorithm	Parameters and Values
TS	Tabu list size: $s_{TL} = 48$
VNS	Number of neighborhoods: $K = 2$
DE	Population size: $N = 100$; Mutation and crossover parameters: $F = 0.7$, $CR = 0.3$
PSO	Swarm size: $N = 100$; Parameters: $\chi = 0.729$, $c_1 = c_2 = 1.49$; Neighborhood: ring (radius 1)
Common	Migration period: $T_{\text{mig}} = 100$ iterations
AP	Investment time fraction: 0.3 (i.e., 30%); $\beta = 0.05$; $T_{\text{noimp}} = 5000$ non-improvement cycles

the use of communication strategies that promote cooperation in order to increase efficiency. Path Relinking (PR) [130] is also employed to enhance performance. Finally, homogeneous trading-based algorithm portfolios, which consist of combinations of the TS algorithm, are also used to tackle the considered problem.

PSO and DE were originally designed for real-valued search spaces. Thus, the question arises on how it can be applied on the studied permutation problems. This is addressed by using the *smallest position value* (SPV) representation scheme [131], which maps real-valued individuals into permutations of a predefined reference vector. Specifically, this scheme considers the real values as weights that determine the priority of the corresponding components of the reference vector [131].

4.3.2.1 Parallel Implementations

The employed metaheuristics can be easily parallelized with subsequent reduction in running time [79]. A parallelization framework is proposed that is based on a typical master-slave model that consists of M metaheuristics. Each slave node runs a copy of the same algorithm with identical or different parameterization. Communication among the algorithms is achieved via the exchange of messages through the master node. The communication is inherently asynchronous and exploits a migration scheme [79] that involves the periodic exchange of solutions.

The master node retains an external archive of M elite solutions, one solution detected by each algorithm. Whenever an algorithm discovers a new elite solution, it

forwards it to the master and the previous solution that was discovered by the same algorithm is replaced. Also, each algorithm requests periodically (with a period T_{mig}) to acquire the best elite solution that exists in the archive of the master. Next, the acquired elite solution is incorporated in the algorithm either as a new starting point (TS and VNS) or as a new population member (DE and PSO).

In addition to the aforementioned choices, the acquired elite solution can be used also for a *Path Relinking* (PR) phase [130]. In this context, PR is initiated with a user-defined probability ρ_{PR} to locate the best permutation between the slave's own best solution (starting point) and the acquired one (target point). This is achieved by iteratively permuting the current position such that a new one with the lowest Hamming distance from the target point is achieved. The best position found by this local search procedure is used as a new initial point or a new population member for the algorithm.

4.3.2.2 Algorithm Portfolio

The AP framework proposed in Section 4.2 is also adopted to tackle CWM problems. Additionally, a combination between the elite solution of the algorithm and the one it purchases takes place in order to further enhance the solution quality, The combination is attained by applying *crossover* and *mutation* between the solutions. Specifically, crossover is implemented by retaining the components that are equal in the two solutions. Mutation is then applied on the different components of two solutions and involves their random permutation. The resulting new solution initiates a new trajectory (in TS and VNS) or replaces the worst particle in the population (in DE and PSO). Therefore, stochasticity is infused in the algorithms boosting their exploration dynamics.

4.3.3 Experimental results

Parallel implementations of TS, VNS, DE, and PSO were considered, according to the settings provided in Chapter 2. Also, an AP was considered based on different variants of the TS, which was identified as the best-performing algorithm. Table 4.1 reports the parameter configuration of the employed algorithms. All experiments were conducted on the `glacier` cluster of the Sharcnet consortium, using 8 and 16 nodes (one master node and the rest were slave nodes). The implementation was based on

Table 4.2: Number of successes (suc) over 25 experiments (and the corresponding percentage) along with the number of unique solutions (uni) and the corresponding percentage with respect to the total number of solutions found by the algorithm.

			Number of CPUs	
Algorithm	Time (hrs.)		8	16
TS	12	suc	14 (56.0%)	22 (88.0%)
		uni	7 (50.0%)	5 (22.7%)
	24	suc	20 (80.0%)	24 (96.0%)
		uni	4 (18.2%)	6 (25.0%)
VNS	12	suc	3 (12.0%)	4 (16.0%)
		uni	1 (33.3%)	1 (25.0%)
	24	suc	5 (20.0%)	6 (24.0%)
		uni	2 (40.0%)	2 (33.3%)
DE	12	suc	8 (32.0%)	10 (40.0%)
		uni	1 (12.5%)	2 (20.0%)
	24	suc	15 (60.0%)	18 (72.0%)
		uni	3 (20.0%)	4 (22.2%)
PSO	12	suc	7 (28.0%)	11 (44.0%)
		uni	2 (28.6%)	3 (27.2%)
	24	suc	13 (52.0%)	17 (68.0%)
		uni	4 (30.8%)	5 (29.4%)
AP	12	suc	15 (60.0%)	25 (100.0%)
		uni	7 (46.7%)	10 (40.0%)
	24	suc	15 (60.0%)	25 (100.0%)
		uni	5 (33.3%)	9 (36.0%)

the OpenMPI project. Also, two different running time budgets, namely 12 and 24 hours, were considered for all algorithms. A number of 25 independent experiments were conducted for each algorithm and test case. Henceforth, the experimental configurations are denoted with their corresponding number of CPUs followed by the running time, i.e., 8/12, 8/24, 16/12, and 16/24, respectively.

The primary objective of the experiments was the detection of a hard CWM class, namely CW$(48, 36)$. A secondary objective was the comparison between the algo-

rithms. For this purpose, the number of successful experiments (out of 25) were recorded as well as the number of *unique solutions* detected by each algorithm, i.e., solutions that are not cyclic permutations of other solutions detected by the same algorithm. An experiment is considered to be *successful* if the discovered solution is a ternary sequence that is the first row of any CW$(48, 36)$ matrix and therefore minimizes the objective function defined in Eq. (4.5). Regarding the first objective, overall the algorithms detected the 22 unique solutions given below, with "-" denoting "-1", and "+" denoting "+1":

```
0+++-0-++-+0-0-+0+++-0+-0-+++0++++-0+0-+0--+-0--
0++0+-++0-00--+-0-+-++--0++0-+++0+00+-++0++----+
+++-0-+0-+++0+-0-0+-++-0-+++0--0-+--0+-0+0-++++0
++-000-+---+-0+--+++00+++-+000-+++++-0--+-++00-+
+00-++--000-+-+-+-0+---++00+++++000-++-++-0--+++
00-+-+--00++---++0+0-+-+00++--+-00+++++++0-0+--+
-0+0+-++00+++++-00-+--++-0-0-+++00-++---00-+++-+
0++--+++000-+-+-00+-----0++++-++000+--++00+-++-+
+0-+++00+0-++-0+-0+-++++-0+++-00-0-+-+0+-0----++
+0+--+00++-++-00++--+++0-0-+-+00-+----00++++-++0
00++----0-+++-+-000+-+++00++++-+0-+--++-000-+++-
---+00++-++-00+++-+++0-0++-+00-+----00++-+-++0+0
+-++000+++-+++0--++-+00-+---000++-+-++0+----+00+
-+0++++0++0-0+--+-0-+-+0--0++--0++0+0-+-++0++--0
00+++--++0+-++-+00++--+000++-++++0-----+00-+-+-0
-+-0-00+++-0-++-+--0-+0+++-0+00-+++0++++--+0-+0-
+-+--00+++--++0+0++-+00++--+-00++-++++0-0---+00-
0+0--++00+++++-00-++-++-0-0++++00-++---00-+-+-+-
-+0+-0+---+++0++++00+0-++-0+-0--++++-0-++-00-0-+
-+0++0-0---++0-+-+0-+0+++-0++0+0++-+-0++--0--0++
++-0-0-++--+-00-+++++00++-+0+0-+-+++-00---++-00+
-0++---0-0++0+-++0--0--++0-+-++0+0++0-+++0+-0+-+
```

Regarding the second objective, Table 4.2 reports the successes (both in number and percentage over 25 experiments) of each algorithm. Additionally, the number of unique solutions and the corresponding percentage with respect to the total number of solutions detected by the algorithm are reported. A first reading of the results clearly

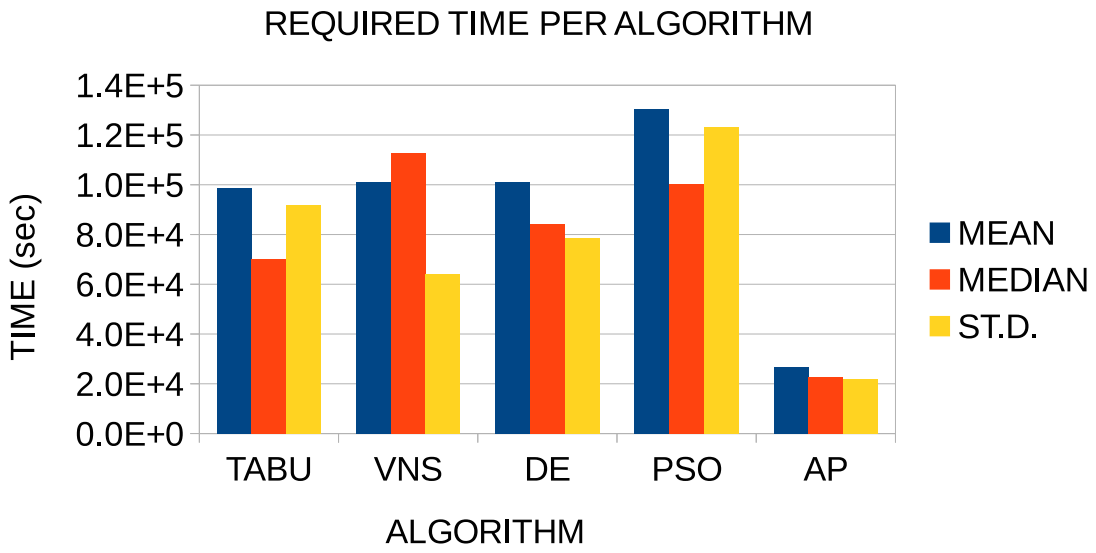


Figure 4.1: Statistics for the required running time (in seconds) per algorithm for the successful experiments over all problem configurations.

shows that TS outperforms the rest of the distinct algorithms by achieving significantly higher number of successes. This is verified also for the number of unique solutions found per case. Obviously, the exhaustive local neighborhood search along with the hill-climbing capabilities and the cooperation in the parallel scheme, equipped TS with satisfactory trade-off between exploration and exploitation.

However, even TS was outperformed by the AP in most test cases. In fact, the AP was the only approach that achieved 100% successes for some experimental configurations. Even in the 8/24 case, where TS outperformed AP in successes, its unique solutions were 4 in 20 successful experiments against 5 in 15 successful experiments of the AP. This evidence is a strong indication of the benefits gained from the special solution trading scheme implemented in the AP against the simple cooperation of the plain parallel models. The rest of the algorithms, especially DE and PSO, could easily bypass a solution. Nevertheless, the performance between DE and PSO was very similar (their success percentages differ by 6% at most).

Table 4.2 also reveals a consistent improvement of the four algorithms' results when the number of CPUs or the running time is doubled. Moreover, increasing the running time appears to be more effective than increasing the number of CPUs in most cases. However, this is not verified for the AP, where the number of CPUs seems to be of primary importance. At first sight, this observation seems to contradict

the previous one. However, it can be easily explained by the evidence illustrated in Fig. 4.1, which reports the required running time (in seconds) per algorithm only for the successful experiments over all problem configurations.

As can be seen, the AP approach requires only a small fraction of the running time required by the rest of the algorithms to detect a solution. Thus, providing additional time does not have a crucial impact. On the other hand, adding slave nodes increases the search capacity of the AP with a consequent surge in successes. The reported running times were further investigated by conducting Wilcoxon rank-sum tests for each pair of algorithms (only for the successful experiments) at significance level 95%. The tests revealed that only the AP had statistically significant differences in running time with the rest of the algorithms.

Overall, the AP approach was shown to be the most efficient and effective among the considered ones. Given that AP differs from the simple parallel TS only in the sophisticated solution trading scheme, it can be inferred that this AP framework can be highly beneficial.

4.4 Application in Production Scheduling

Recently, manufacturing companies have focused on reducing material waste by recovering a fraction of the used products. The reasons behind this action are mainly the increasing environmental care and the possible economical benefits. To this end, remanufacturing has been extensively used involving the transformation of used products into like-new ones. Specifically, the used product is first disassembled and then its problematic parts are either repaired or replaced with new ones. The final remanufactured products have the same quality as the new ones and usually are sold in competitive prices, thereby attracting the interest of potential buyers.

Economic Lot Sizing (ELS), i.e., planning manufacturing/production orders over a finite, discrete planning horizon where demand is dynamic and deterministic has been extensively studied in relevant literature. The ELS problem with remanufacturing options (ELSR) has recently received growing attention as an alternative to manufacturing. According to this, in every period over a finite, discrete planning horizon, a manufacturer faces a dynamic and deterministic demand for a single type of product, and has a deterministic number of returned used items at its disposal. In order to satisfy product demand, the manufacturer can either decide to manufacture

new items or remanufacture a number of returned items. Separate inventories are held for manufacturing and remanufacturing items. Also, fixed costs are incurred when ordering manufactured or remanufactured products. Additional, holding costs are incurred for storing serviceable and the returned product in inventory.

Different variants of the ELSR problem have been studied [132, 133]. In this section, the classical Wagner-Whitin model [134] with a remanufacturing process is considered. It employs the dynamic lot sizing model with separate manufacturing and remanufacturing setup costs as it was introduced in [23] and further studied in [135].

4.4.1 Problem Formulation

The problem assumes a manufacturer that sells a single type of product over a finite planning horizon of T time periods. In each time period $t = 1, 2, \ldots, T$, the consumers state their demand denoted by D_t, along with a number of used products that are returned to the manufacturer. The fraction R_t of returned products in period t that can be recovered and sold as new is stored at a recoverables inventory with a holding cost h^R per unit time. To satisfy the demand, a number of z_t^R and z_t^M products are remanufactured and manufactured, respectively, in period t and then brought to a serviceables inventory with a holding cost h^M per unit time. Naturally, the manufacturing and remanufacturing process incur setup costs denoted by K^R and K^M, respectively.

The target is to minimize the incurring setup and holding costs by determining the exact number of manufactured and remanufactured items per period under a number of constraints. The corresponding cost function is defined as follows [135],

$$C = \sum_{t=1}^{T} \left(K^R \gamma_t^R + K^M \gamma_t^M + h^R y_t^R + h^M y_t^M \right), \tag{4.6}$$

where γ_t^R and γ_t^M are binary variables denoting the initiation of a remanufacturing or manufacturing lot, respectively. The inventory levels of items that can be remanufactured or manufactured in period t are denoted by y_t^R and y_t^M, respectively. The operational constraints of the model are defined as follows:

$$y_t^R = y_{t-1}^R + R_t - z_t^R, \quad y_t^M = y_{t-1}^M + z_t^R + z_t^M - D_t, \quad t = 1, 2, \ldots, T, \tag{4.7}$$

$$z_t^R \leq Q\ \gamma_t^R, \quad z_t^M \leq Q\ \gamma_t^M, \quad t = 1, 2, \ldots, T, \tag{4.8}$$

Table 4.3: Parameters of the considered problem and the employed algorithms.

Problem parameter	Value(s)		Algorithm parameter	Value(s)
Dimension	n = 24	AP	Number of slave algorithms	M = 4
Setup costs	K^M, $K^R \in \{200, 500, 2000\}$		Per algorithm execution time	T_{tot} = 75000 msec.
Holding costs	h^M = 1, $h^R \in \{0.2, 0.5, 0.8\}$		Constants α, β	α = 0.1, β = 0.05
Demand for period t	$D_t \sim N(\mu_D, \sigma_D^2)$	PSO	Model	lbest (ring topology)
	$\mu_D = 100$		Swarm size	60
	$\sigma_D^2 = 10\%$ of μ_D (small variance)		Constriction coefficient	χ = 0.729
	$\sigma_D^2 = 20\%$ of μ_D (large variance)		Cognitive/social constants	$c_1 = c_2$ = 2.05
Returns for period t	$R_t \sim N(\mu_R, \sigma_R^2)$	DE	Population size	60
	$\mu_R \in \{30, 50, 70\}$		Operator	DE/rand/1
	$\sigma_R^2 = 10\%$ of μ_R (small variance)		Differential/crossover constants	F = 0.7, CR = 0.3
	$\sigma_R^2 = 20\%$ of μ_R (large variance)	TS	Size of tabu list	24

$$y_0^R = y_0^M = 0, \quad \gamma_t^R, \gamma_t^M \in \{0, 1\}, \quad y_t^R, y_t^M, z_t^R, z_t^M \geq 0, \quad t = 1, 2, \ldots, T. \tag{4.9}$$

Equation (4.7) guarantees the inventory balance, while Eq. (4.8) assures that fixed costs are paid whenever a new lot is initiated. In [135] the value of Q is suggested to be equal to the total demand of the planning horizon. Finally, Eq. (4.9) asserts that inventories are initially empty and determines the domain of each variable. The decision variables of the optimization problem are z_t^M and z_t^R for each period t. Thus, for a planning horizon of T periods the corresponding problem has dimension $n = 2\,T$. More details about the considered problem can be found in [23, 135, 136].

4.4.2 Employed Algorithms

Recently, modern population-based metaheuristics have been employed to effectively tackle the Wagner-Whitin and relevant inventory optimization problems [136–138]. Although, the employed population-based algorithms are primarily designed to tackle real-valued optimization problems, proper modifications related to the problem and the algorithms themselves can render them applicable to integer and mixed-integer optimization problems.

In order to enrich the algorithmic artillery for such types of optimization problems, the considered problem is tackled through a heterogeneous trading-based algorithm portfolio presented in Section 4.2. The employed algorithm portfolio consists of 4 metaheuristic algorithms, namely Particle Swarm Optimization (PSO) [129], Differential Evolution (DE) [38], Tabu Search (TS) [90], and Iterated Local Search (ILS) [139].

Table 4.4: Percentage error of the compared algorithms for different problem parameters.

	Alg.	Avg	StD	Max
All	SM_4^+	2.2	2.9	**24.3**
	PSO	4.3	4.5	49.8
	DE	3.3	5.1	31.9
	TS	51.6	33.4	255.5
	ILS	80.3	54.3	450.8
	AP	**1.9**	**2.8**	35.6
σ_D^2=10%	SM_4^+	2.1	2.8	**18.9**
	PSO	4.4	4.6	49.8
	DE	3.4	4.8	31.7
	TS	50.9	33.2	200.2
	ILS	79.7	54.2	450.8
	AP	**1.8**	**2.6**	26.8
σ_D^2=20%	SM_4^+	2.4	3.0	**24.3**
	PSO	4.1	4.5	48.3
	DE	3.3	5.2	31.9
	TS	52.4	33.5	255.5
	ILS	80.9	54.4	421.5
	AP	**2.0**	**2.9**	35.6

	Alg.	Avg	StD	Max
h^R=0.2	SM_4^+	1.7	2.5	**21.1**
	PSO	4.5	5.2	49.8
	DE	3.0	5.3	30.9
	TS	45.0	26.4	255.5
	ILS	94.8	67.2	450.8
	AP	**1.5**	**2.5**	35.6
h^R=0.5	SM_4^+	2.3	3.0	**24.3**
	PSO	4.3	4.5	45.5
	DE	3.3	5.0	31.9
	TS	50.8	32.1	202.1
	ILS	77.6	48.2	261.1
	AP	**1.9**	**2.8**	27.4
h^R=0.8	SM_4^+	2.8	3.0	**20.6**
	PSO	4.0	3.9	42.9
	DE	3.7	4.5	31.4
	TS	59.1	39.0	235.2
	ILS	68.4	40.8	211.4
	AP	**2.2**	**3.0**	21.6

	Alg.	Avg	StD	Max
μ_R=30	SM_4^+	**1.2**	**1.8**	**12.1**
	PSO	3.5	3.1	45.5
	DE	3.3	5.0	28.2
	TS	37.2	23.9	255.5
	ILS	70.6	46.5	336.8
	AP	1.6	2.5	25.7
μ_R=50	SM_4^+	2.3	2.7	**16.2**
	PSO	4.1	4.0	34.0
	DE	3.5	5.2	31.9
	TS	50.8	27.3	153.6
	ILS	83.7	53.0	364.2
	AP	**2.0**	**2.9**	27.4
μ_R=70	SM_4^+	3.3	3.5	**24.3**
	PSO	5.1	5.9	49.8
	DE	3.3	4.6	31.7
	TS	66.9	39.8	235.2
	ILS	86.5	61.1	450.8
	AP	**2.0**	**2.9**	35.6

	Alg.	Avg	StD	Max
K^M=200	SM_4^+	**2.3**	**2.6**	**13.5**
	PSO	4.0	3.1	45.5
	DE	3.2	3.9	24.0
	TS	39.1	27.3	255.5
	ILS	62.6	64.0	450.8
	AP	2.4	3.0	21.6
K^M=500	SM_4^+	2.1	2.5	**12.8**
	PSO	4.5	4.1	27.5
	DE	2.5	2.6	15.2
	TS	67.9	33.1	197.1
	ILS	62.0	40.6	278.1
	AP	**1.8**	**2.4**	17.6
K^M=2000	SM_4^+	2.3	3.4	**24.3**
	PSO	4.4	5.9	49.8
	DE	4.3	7.1	31.9
	TS	47.9	32.7	235.2
	ILS	116.3	34.2	260.4
	AP	**1.4**	**2.8**	35.6

	Alg.	Avg	StD	Max
K^R=200	SM_4^+	**1.9**	**2.1**	**11.8**
	PSO	5.7	5.5	49.8
	DE	3.8	4.0	24.0
	TS	75.2	38.0	203.3
	ILS	63.0	45.4	260.4
	AP	3.0	3.3	21.6
K^R=500	SM_4^+	3.4	3.2	19.1
	PSO	3.8	4.1	37.4
	DE	1.8	2.0	11.2
	TS	50.8	23.8	235.2
	ILS	62.4	37.8	244.8
	AP	**1.3**	**1.7**	**11.6**
K^R=2000	SM_4^+	1.4	2.9	**24.3**
	PSO	3.3	3.5	45.5
	DE	4.4	7.1	31.9
	TS	29.0	16.4	255.5
	ILS	115.5	59.2	450.8
	AP	**1.3**	**2.8**	35.6

	Alg.	Avg	StD	Max
σ_R^2=10%	SM_4^+	2.2	2.9	**21.1**
	PSO	4.3	4.6	46.7
	DE	3.4	5.0	31.4
	TS	52.1	34.3	233.8
	ILS	80.4	54.4	450.8
	AP	**1.8**	**2.7**	35.6
σ_R^2=20%	SM_4^+	2.3	2.9	**24.3**
	PSO	4.2	4.5	49.8
	DE	3.3	4.9	31.9
	TS	51.2	32.5	255.5
	ILS	80.1	54.2	399.1
	AP	**2.0**	**2.9**	25.6

4.4.3 Experimental Results

The proposed approach was evaluated on the established test suite used in [135]. It consists of a full factorial study of various problem instances with common planning horizon $T = 12$. Table 4.3 summarizes the configuration of the problem parameters as well as the employed algorithm parameters for the AP. Further details on the problem setting can be found in [135]. The proposed AP was compared against the best-performing variant (SM_4^+) of the state-of-the-art Silver-Meal heuristic [135], as well as against the serial versions of its constituent algorithms. The goal of the experiments was to achieve the lowest possible percentage error [135] from the global optimum within a predefined budget of total execution time T_{tot}. The global optimum per problem was computed by CPLEX and provided in the test suite.

Table 4.4 shows the average (Avg), standard deviation (StD), and maximum (Max) value of the percentage error for the different values of the problem parameters. A first inspection of the results reveals superiority of the proposed AP, which achieves the best overall mean percentage error (1.9%). The second lowest value was achieved by SM_4^+ (2.2%), followed by the sequential versions of DE (3.3%) and PSO (4.3%). Specifically, AP prevails in 14 out of 17 considered parameter cases, while in the rest 3 cases SM_4^+ is the dominant algorithm. The results of SM_4^+ and PSO were directly adopted from [135] and [136], respectively.

The results indicate that population-based algorithms (DE and PSO) outperform (by far) the trajectory-based ones (TS and ILS). Moreover, when all algorithms are integrated into the AP, the overall performance with respect to solution quality is further enhanced. This can be attributed to the dynamics of the trading among the algorithms. In particular, it was observed that the population-based algorithms were mainly the seller ones, using their exploration capability to discover high-quality solutions. On the other hand, trajectory-based algorithms employed their exploitation power to further fine-tune the vast number of acquired solutions. From this point of view, the employed algorithms of the AP exhibited complementarity, which is a desired property in APs [4,15]. Also, it was observed that between the two population-based algorithms, PSO acquired a higher number of solutions than DE during the optimization whereas the solutions of the latter were of better quality.

4.5 Application in Humanitarian Logistics

Humanitarian Logistics (HL) has attracted increasing interest over the last two decades due to the exponential surge in natural and man-made disasters [24]. From earthquakes to tsunamis, natural disasters have produced startling devastation with major death tolls and economical consequences.

HL plays a crucial role in addressing disaster relief operations problems. Quoting from [140], HL is responsible for "*planning, implementing and controlling the efficient, cost-effective flow and storage of goods and materials, as well as related information, from point of origin to point of consumption for the purpose of alleviating the suffering of vulnerable people*". In general, it is perceived that HL constitutes a powerful tool for managing disaster relief operations, making the difference between success and failure [141,142].

Even though HL is significant to prevent from consequences on people's health or life loss, the relevant literature is limited compared to commercial logistics, which aims at cost reduction [142]. Nevertheless, different aspects of HL have been addressed in several studies [143], including transportation and routing [144, 145], supply chain and procurement [146, 147], and distribution and supply location [148, 149].

In [150] a multi-period problem is studied, taking into account limited supply and transportation capacity that aims to minimize losses caused by (i) the mismatch between supplies and demand, and (ii) the transportation time due to logistics processes. In the next section, a model is considered where the objective is the minimization of losses caused by the mismatch between supply and demand of relief resources in the affected areas, taking into account the already existing quantities (if any) and the importance of the different resources. Furthermore, beyond constraints related to number, volume, and load capacity of vehicles, road capacity constraints are considered. The latter is the source of bottleneck in supply chain due to decrease in transportation capacity and unexpected increase of relief vehicles [151], and this maybe be defined by authorities.

4.5.1 Problem Formulation

In the studied model, a set J of affected areas (AAs) and a set I of dispatch centers (DCs) are considered. Relief resources (commodities) are transported from DCs to AAs through a number of vehicles of different type and mode. Also, ground and aerial vehicles of two sizes (big and small) are considered. Henceforth, the set of

Table 4.5: Notation used in the proposed model.

Model Variable	Description
T	Planning horizon
I	Set of Dispatch Centers (DCs)
J	Set of Affected Areas (AAs)
C	Set of commodities
M	Set of transportation modes
m	Index denoting the transportation mode (ground, air)
O_m	Set of vehicle types of transportation mode m
o	Index denoting the vehicle type (big vehicle, small vehicle)
b_{cj}	Importance weight of commodity c in AA j
w_c	Unit weight of commodity c
$volume_c$	Unit volume of commodity c
cap_{mo}	Capacity of type o, mode m vehicle
vol_{mo}	Volume capacity of type o, mode m vehicle
d^t_{cj}	Demand for commodity c in AA j at time period t
k^t_{ijm}	Traffic restriction for mode m vehicles traveling from DC i to AA j at time t
v^t_{imo}	Number of type o, mode m vehicles at DC i at time t
Decision Variable	**Description**
s^t_{cijm}	Delivered quantity of commodity c from DC i to AA j through transportation mode m at time t
v^t_{cijmo}	Number of type o, mode m vehicles used at period t to transport commodity c from DC i to AA j

commodities is denoted as C, the set of transportation modes is denoted as M , and the set of vehicles of mode $m \in M$ is denoted as O_m. The planning horizon is finite and denoted as T. The complete notation used in the studied model is presented in Table 4.5.

Based on this notation, the optimization problem lies in specifying the optimal delivered quantities s^t_{cijm} per commodity $c \in C$, from DC i to AA j using vehicles of transportation mode m, for each time period t. Moreover, the optimal number v^t_{cijmo} of type o, mode m vehicles is needed to be specified that are used to transport the

commodities at each time period t. All decision variables assume integer values. The corresponding minimization problem is defined as follows,

$$\min \sum_{t\in T}\sum_{j\in J}\sum_{c\in C} b_{cj}\left(d^t_{cj} - \sum_{i\in I}\sum_{m\in M} s^t_{cijm} - I^{t-1}_{cj}\right)^2, \tag{4.10}$$

where b_{cj} is a scalar importance weight of commodity c at AA j. The model is subject to the following constraints,

$$I^0_{cj} = Y_{cj}, \qquad \forall\, c \in C,\ \forall\, j \in J, \tag{4.11}$$

$$I^t_{cj} = \sum_{i\in I}\sum_{m\in M} s^t_{cijm} - d^t_{cj} + I^{t-1}_{cj}, \qquad \forall\, t \in T,\ \forall\, c \in C,\ \forall\, j \in J, \tag{4.12}$$

$$\sum_{c\in C}\sum_{j\in J} s^t_{cijm}\, w_c \leqslant \sum_{o\in O_m} v^t_{imo}\, cap_{mo}, \qquad \forall\, t \in T,\ \forall\, i \in I,\ \forall\, m \in M, \tag{4.13}$$

$$\sum_{c\in C}\sum_{j\in J} s^t_{cijm}\, vol_c \leqslant \sum_{o\in O_m} v^t_{imo}\, vol_{mo}, \qquad \forall\, t \in T,\ \forall\, i \in I,\ \forall m \in M, \tag{4.14}$$

$$s^t_{cijm} \leqslant \min\left\{\frac{\sum_{o\in O_m} v^t_{cijmo}\, cap_{mo}}{w_c},\ \frac{\sum_{o\in O_m} v^t_{cijmo}\, vol_{mo}}{volume_c}\right\}, \tag{4.15}$$

$$\sum_{c\in C}\sum_{o\in O_m} v^t_{cijmo} \leqslant k^t_{ijm}, \qquad \forall\, t \in T,\ \forall\, i \in I,\ \forall\, m \in M,\ \forall j \in J, \tag{4.16}$$

$$\sum_{c\in C}\sum_{j\in J} v^t_{cijmo} \leqslant v^t_{imo}, \qquad \forall\, t \in T,\ \forall\, i \in I,\ \forall\, m \in M,\ \forall\, o \in O_m. \tag{4.17}$$

Equation (4.11) accounts for the initial inventory level of commodity c pre-existing at DC j. Equation (4.12) determines the inventory balance, which takes into account the demand of the commodity c and the replenishment quantity. Equations (4.13) and (4.14) refer to capacity and volume constraints, respectively. Equation (4.15) defines upper limit of the delivered quantity s^t_{cijm}, which is useful for bounding the decision variables. Equation (4.16) stands for traffic flow restrictions expected in natural disasters, e.g., roads that are partially damaged or destroyed, reducing traffic capacity. Equation (4.17) ensures that the number of vehicles transporting the commodities in a particular AA does not exceed the total number of vehicles.

The squared error in Eq. (4.10) can be replaced by absolute error if metaheuristics are the employed solvers. Nevertheless, the quadratic form is chosen, in order to render the problem solvable by CPLEX.

4.5.2 Employed Algorithms

Humanitarian logistics models require efficient solvers that can produce satisfactory solutions within strict time constraints. Metaheuristics have been recognized as valuable optimization tools for this purpose. Such algorithms are able to offer solutions to difficult optimization problems within reasonable amount of time. However, this comes at the cost of dubious optimality of the detected solution. For more details about metaheuristics the reader is referred to Chapter 2. In literature, there is a significant amount of research studying the performance of metaheuristics in various problems in logistics, while recently several works appeared also in the growing area of HL [152, 153].

In this study, two prevailing population-based metaheuristic algorithms, namely DE and PSO are considered to address the humanitarian logistics problem. Details about these algorithms can be found in Section 2.6 and 2.7, respectively. Moreover, an enhanced DE (eDE) variant is considered, which is described in Section 2.6. Finally, homogeneous and heterogeneous trading-based APs, which consist of combinations of the aforementioned algorithms, are also used to tackle the considered problem.

4.5.2.1 Further Applicability Issues

Two main issues need to be addressed prior to the application of the presented metaheuristics on the problem of Section 4.5.1. The first one is related to the discrete nature of the search space, while the second one refers to constraint handling.

Regarding the first issue, simple rounding to the nearest integer is used. Specifically, the algorithms are applied on the corresponding real search space and, for the function evaluation, the vectors are rounded to the nearest integer ones. In DE and eDE, the rounded vectors are also retained in the population. In PSO, rounded vectors replace best positions. Rounding is a common approach, successfully applied in similar problems [137, 154].

The constraint handling problem is tackled with the widely used *penalty function* approach, combined with a set of preference rules between feasible and infeasible

Table 4.6: Capacity and volume information for vehicle types I (small) and II (big).

	Transportation Mode			
	Ground		Air	
	I	II	I	II
Load Capacity (ton)	3	10	4	9
Load Volume (m^3)	20	44	35	75

Table 4.7: Commodities information.

	Water	Medicines	Food
Importance weight	0.35	0.35	0.30
Unit Weight (kg)	650	20	200
Unit Volume (m^3)	1.44	0.125	0.60

solutions:

(i) Between two infeasible solutions, the one that violates fewer constraints is selected.

(ii) Between a feasible and an infeasible solution, the feasible one is preferred.

(iii) Between two feasible solutions, the one with the lowest objective value is preferred.

These rules have been previously used with PSO and DE [154]. The employed penalty function has a simple form,

$$P(\mathbf{x}) = f(\mathbf{x}) + \sum_{i \in VC(\mathbf{x})} |V(i)|\,, \tag{4.18}$$

where $f(\mathbf{x})$ is the actual objective value of $\mathbf{x}$; $V(i)$ is the violation magnitude of the i-th constraint; and $VC(\mathbf{x})$ is the set of constraints violated by $\mathbf{x}$. Note that the penalty for a violated constraint depends on the magnitude of violation. Apparently, in absence of violated constraints, the penalty function is equal to the original objective function.

4.5.3 Experimental Results

The main goal in the studied model is the minimization of losses caused by the mismatch between supply and demand, as well as the determination of the optimal

Table 4.8: Number of vehicles per DC.

	Transportation Mode			
	Ground		Air	
	I	II	I	II
DC_1	4	5	1	1
DC_2	4	5	1	1

Table 4.9: Mean, standard deviation, minimum, and maximum solution error values for all algorithms, averaged over all problems. Best values are boldfaced. The "+" symbol denotes AP approach constituting of the corresponding algorithms.

Algorithm	Mean	St.D.	Min	Max
PSO	513.80	235.85	197.00	2442.20
DE	63.31	40.45	26.97	160.21
eDE	3.54	3.42	0.29	11.80
PSO+DE	52.28	31.11	27.01	129.42
PSO+eDE	4.14	3.99	0.16	13.77
DE+DE	59.65	55.36	21.15	193.81
DE+eDE	0.76	0.91	0.00	2.91
eDE+eDE	**0.75**	**0.85**	**0.00**	**2.27**
PSO+DE+eDE	0.84	1.18	0.00	3.74

number of vehicles for the transportation of relief resources to the stricken areas. In the experiments, three life-essential commodities were considered, namely *water*, *medicines*, and *food*. Among them, the first two were assumed to have slightly higher importance weights than the third one.

Moreover, the existence of two DCs responsible to supply two AAs was assumed as well as two modes of transportation, ground and aerial, using trucks and helicopters, respectively. For each transportation mode, two vehicle types were considered, namely small and big vehicles, henceforth denoted as type I and II, respectively. Tables 4.6 and 4.7 report the relevant information for vehicles and commodities, respectively. Note that the reported data are based on real-world values (e.g., palettes of water bottles, typical transportation boxes for medication etc). Also, Table 4.8 reports the number of available vehicles per DC.

In the context of the proposed model, a test suite of 10 benchmark problems with diverse characteristics was initially generated and solved to optimality with the commercial CPLEX solver. The problems are henceforth denoted as $P1 - P10$. In a second phase, extensive experiments were conducted with the following algorithms: PSO, DE, eDE, AP with PSO+DE, AP with PSO+eDE, AP with DE+DE, AP with DE+eDE, AP with eDE+eDE, and AP with PSO+DE+eDE. The five basic DE and eDE mutation operators of Eqs. (2.2)-(2.6) were considered, along with all combinations of their parameters $F \in [0, 2]$ and $CR \in [0, 1]$, with step size 0.05.

Preliminary experiments provided clear evidence that DE2 with,

$$F = F_1 = F_2 = 0.4, \qquad CR = 0.05,$$

was the most promising setting. The PSO algorithm was considered in its lbest model with ring topology of radius $r = 1$, and the default parameter set,

$$\chi = 0.729, \qquad c_1 = c_2 = 2.05.$$

The population size for all algorithms was set to $N = 150$, since the corresponding optimization problem's dimension was $n = 144$. The boundaries for the decision variables were the ones imposed by the given data (for the vehicles) and the constraints of Section 4.5.3 (for the delivered quantities).

In order to statistically validate each algorithm, 30 independent experiments were performed per problem instance. The experiments were conducted on Intel® i7 servers with 8GB RAM. The running time for each experiment was set to 10 minutes in order to be comparable with that of CPLEX. For each algorithm and experiment, the best solution $\mathbf{x}^*_{\text{alg}}$ and its value f^*_{alg} were recorded, along with the absolute solution error from the global minimum detected by CPLEX, i.e.,

$$solution \;\; error = \left| f^*_{\text{cplex}} - f^*_{\text{alg}} \right|.$$

Average values of solution error over the 30 experiments, along with standard deviation, minimum, and maximum values, were also recorded for performance comparison purpose.

4.5.3.1 Results and Discussion

A summary of all the recorded results is reported in Table 4.9, where the best-performing approach is boldfaced. Also, the results are graphically illustrated to facilitate visual comparisons. The average solution error from the global minimum is

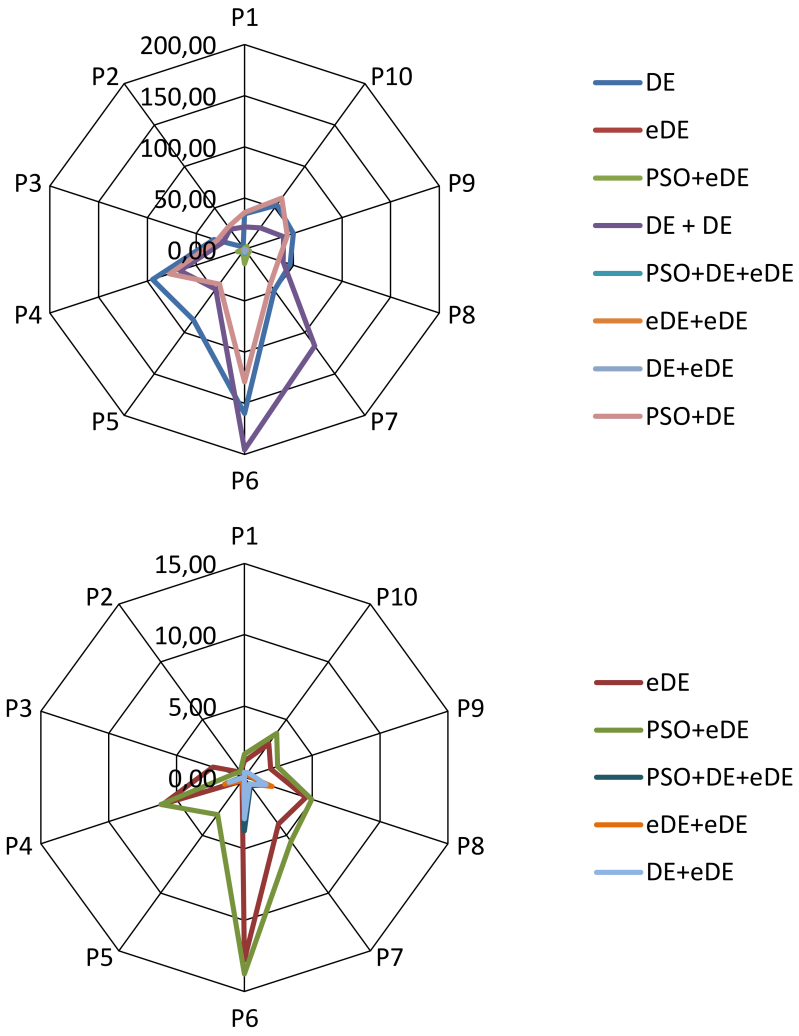


Figure 4.2: Averaged solution error per algorithm and problem (upper part) and zoom in center area (lower part).

presented in the upper part of Fig. 4.2, per problem and algorithm. In the lower part of Fig. 4.2, the central region around the origin is zoomed, exposing the corresponding curves of the most competitive algorithms. Similarly, in the upper and lower part of Fig. 4.3, the averaged standard deviation per problem and algorithm is illustrated. Note that in all figures the results of PSO are excluded, due to scaling reasons.

Furthermore, the success rate per algorithm was also recorded, i.e., the percentage of experiments where it succeeded to reach the optimal solution within the available execution time. Figure 4.4 presents the resulted success rates per problem instance for the most promising algorithms. Finally, the boxplots of Fig. 4.5 illustrate the distribution of the obtained solution error values in all experiments.

The reported results offer several conclusions. Firstly, it can be easily seen that

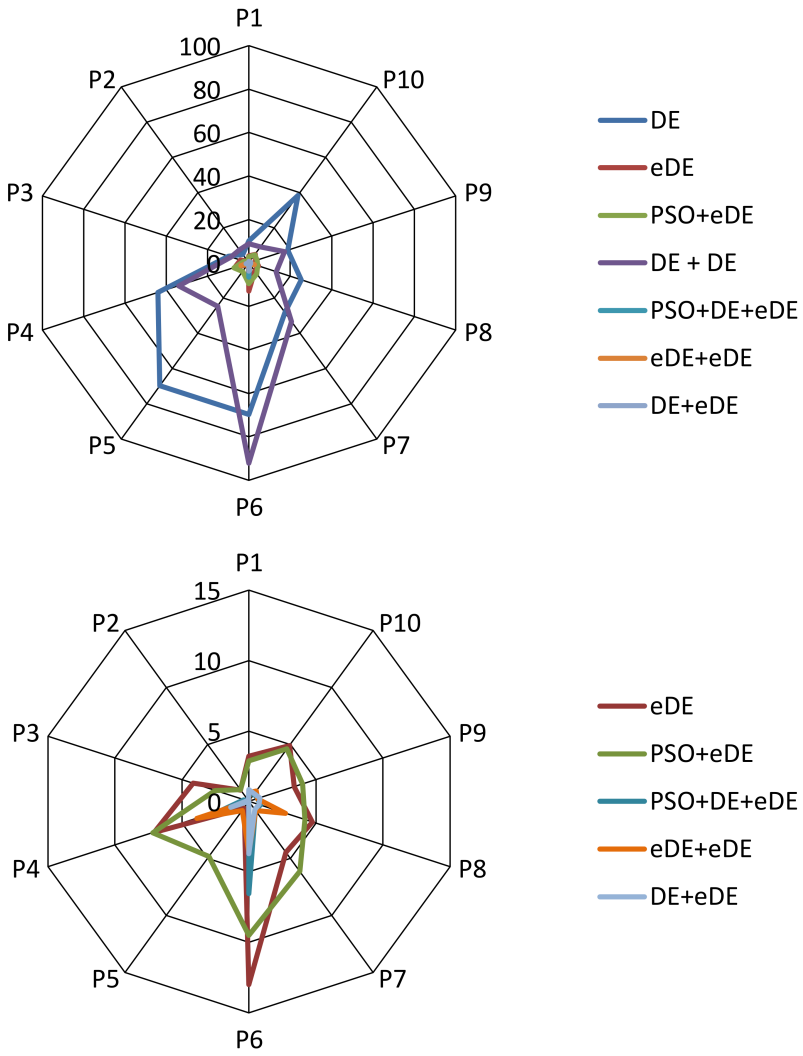


Figure 4.3: Standard deviation of the solution error per algorithm and problem (upper part) and zoom in center area (lower part).

the homogeneous AP approach eDE+eDE as well as the heterogeneous AP with PSO+DE+eDE, outperformed the rest of the algorithms, yielding higher success rates. Also, these two approaches exhibited almost equivalent performance. However, in problems P6-P8, which were proved to be the most difficult ones with respect to the success rates of the algorithms, the eDE+eDE approach dominated in terms of efficiency.

In order to quantitatively study this behavior, the solution purchases between the algorithms of the AP approaches were further analyzed. The analysis verified that, especially for these problems, the number of purchases between the algorithms was remarkably high. This leads to the conclusion that, due to the complexity of

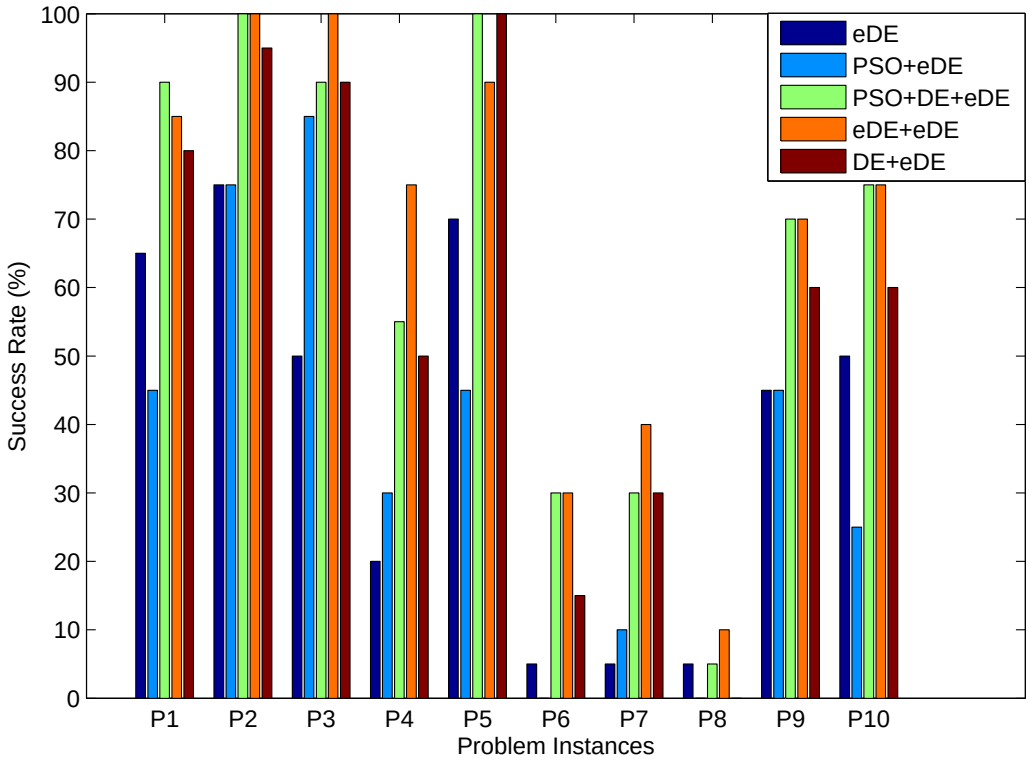


Figure 4.4: Success rates of the most promising algorithms per problem.

these problems, the constituent algorithms of the AP experienced severe difficulties in reaching the optimal solution. Therefore, they were forced to exchange information in order to improve their performance. Also, in the case of PSO+DE+eDE, the assigned execution time per algorithm was shorter than that of each eDE instance in eDE+eDE, because in the first case the total time of the AP is divided by 3, while in the latter it is divided in 2 equal parts. Since PSO was proved to be less efficient than eDE, the assigned time in PSO+DE+eDE was naturally proved to be insufficient.

Regarding the standalone algorithms, eDE was clearly the dominant one, exhibiting undoubtful advantages against the rest. This can also explain the superiority of the eDE-based AP approaches. Obviously, the special probabilistic operator of eDE as well as the restart mechanism with mild perturbations (see Section 2.6) were beneficial for the algorithm. Experimental evidence suggested that this can be attributed to the alleviation of search stagnation, caused by the rounding of the real-valued vectors to the nearest integers. Moreover, this can be related also to the domination of DE2 operator, which offers the necessary diversity to avoid stagnation.

Although there is clear advantage of some algorithms against the rest, there are marginal differences among the most promising approaches. In order to investigate

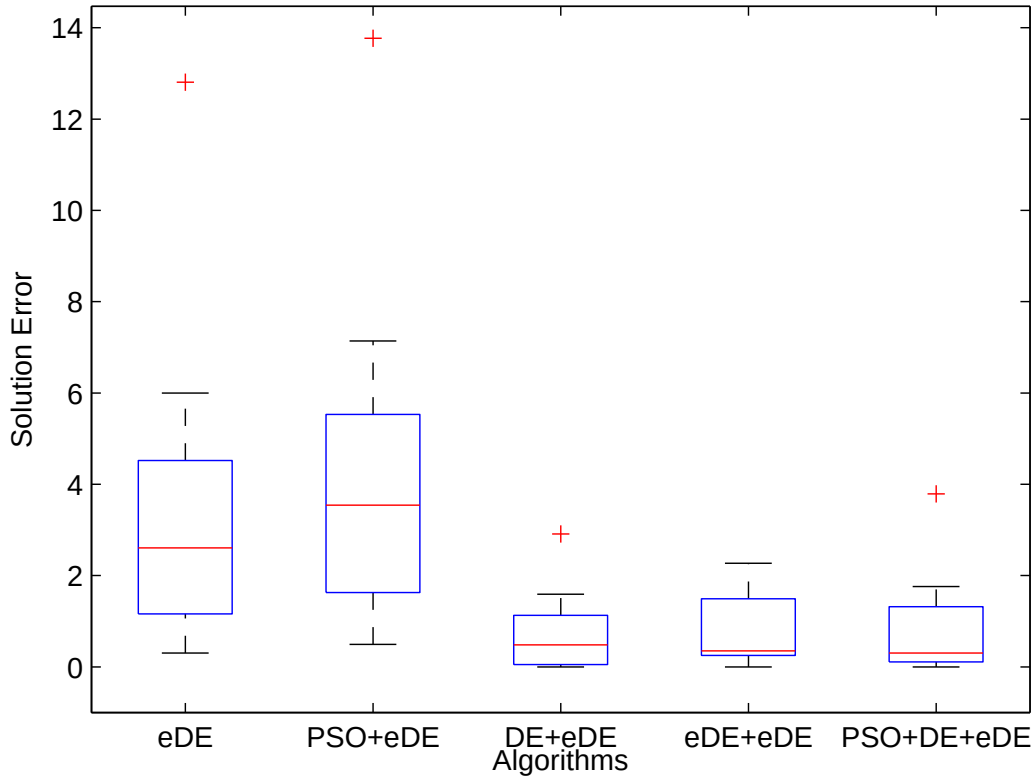


Figure 4.5: Solution error distribution of the most promising algorithms for all test problems.

whether these differences were the outcome of random fluctuations, statistical significance tests among the most competitive algorithms were conducted. Specifically, pairwise comparisons of the algorithms were conducted using the Wilcoxon rank-sum tests at 95% confidence level, for all problems. Whenever an algorithm was statistically superior to another, it was counted as *win* of the algorithm. On the other hand, if it was statistically inferior, it was counted as *loss*. The lack of statistical significance was counted as *draw* for both algorithms.

The results concerning wins/losses/draws are presented in Table 4.10 and the corresponding graphical illustration is given in Fig. 4.6 for all problem instances. The superiority of DE+eDE, eDE+eDE, and PSO+DE+eDE was anew confirmed. In almost all comparisons, these approaches were prevalent against the rest. Yet, most of the comparisons among them resulted in draws, despite the marginal differences reported in Table 4.10. Especially for DE+eDE and eDE+eDE, no losses were reported.

Table 4.10: Wins/losses/draws of row versus column algorithms for all problem instances.

	eDE	PSO+eDE	DE+eDE	eDE+eDE	PSO+DE+eDE
eDE	-	1 / 1 / 8	0 / 5 / 5	0 / 5 / 5	0 / 8 / 2
PSO+eDE		-	0 / 7 / 3	0 / 9 / 1	1 / 7 / 2
DE+eDE			-	0 / 0 / 10	0 / 0 / 10
eDE+eDE				-	0 / 0 / 10
PSO+DE+eDE					-

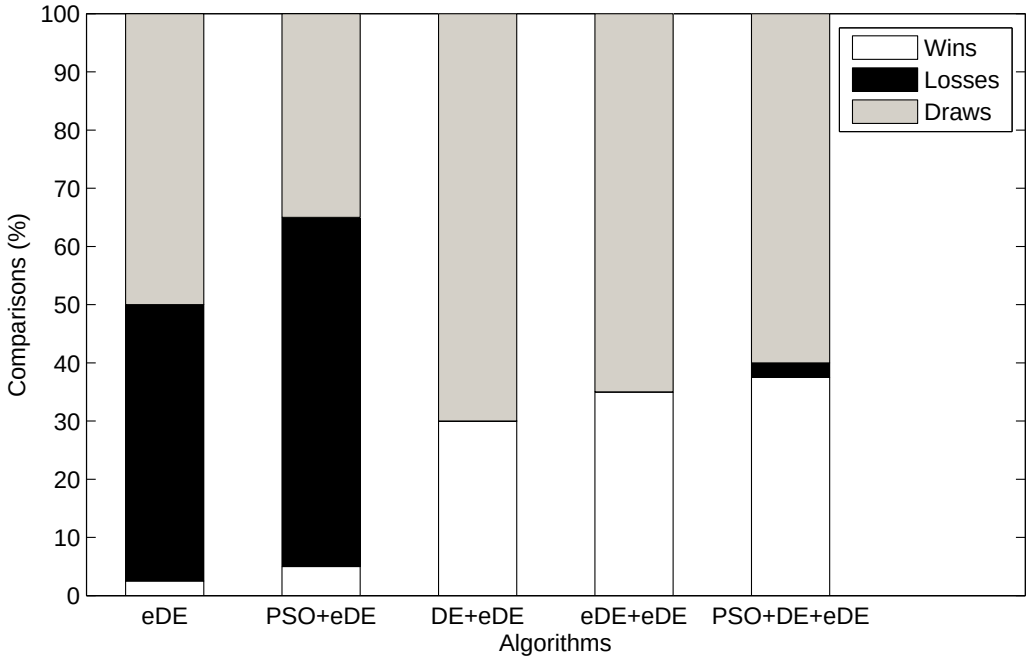


Figure 4.6: Results of the pairwise statistical comparisons among the most competitive algorithms for all test problems.

Thus, the initial assumption regarding the superiority of eDE-based approaches was corroborated by the statistical evidence, placing these AP approaches in a salient position among the most promising solvers.

4.6 Synopsis

In this chapter, a new trading-based algorithm portfolio framework was proposed. The AP comprises of a number of metaheuristic algorithms that operate in parallel and exploit a sophisticated trading-based time allocation mechanism. The mechanism distributes the available computational resources irregularly among the algorithms and indirectly through the trading of solutions. The algorithms sell and buy solutions when needed using execution time as currency. In this way, the AP favors best-performing algorithms with more execution time than the rest, as they sell solutions more frequently. Also, it combines the exploration/exploitation dynamics of each individual constituent algorithm in an efficient way.

The proposed AP was evaluated on a challenging CWM problem. First, parallel versions of TS, VNS, DE and PSO are employed that exploited a typical master-slave parallelization model. Particularly, each slave node executes a copy of the same algorithm and exchanges information with the rest through the master node. All slave nodes assume identical copies of the algorithm selected after preliminary experimentation. Among them, the TS-based approach exhibited the best performance in terms of quality and uniqueness of acquired solutions. The next step involved constructing homogeneous APs based on the TS algorithm. The experimental results revealed the superiority of the TS-based AP framework against the parallel frameworks based on TS or other algorithms. This offers motivation for further research on the application of APs on combinatorial matrices problems.

Additionally, a significant operations research problem, namely the single-item lot sizing problem with returns and remanufacturing, was used as a benchmark for the proposed AP. Initially, the application of well studied metaheuristic algorithms, namely PSO, DE, TS, and ILS was considered. Next, a heterogeneous AP consisted of all four algorithms outperformed its constituent algorithms with respect to quality of acquired solutions. This can be attributed to the dynamics of the trading among the algorithms as well as the synergy among the population-based and the trajectory-based algorithms. In particular, it was observed that the population-based algorithms were mainly the seller ones, using their exploration capability to discover high-quality solutions. On the other hand, trajectory-based algorithms employed their exploitation power to further fine-tune the acquired solutions. Overall, the results indicated that the AP is highly competitive against its constituent algorithms, individually, as well

as against a state-of-the-art algorithm of the considered problem.

Finally, a humanitarian logistics problem complemented the set of applications. For this problem, a model was first introduced that aims at minimizing the losses caused by the mismatch between supply and demand, while concurrently determining the number of different types of vehicles used to transport relief commodities from dispatch centers to stricken areas. A number of test problems with diverse characteristics was generated for the proposed model and solved to optimality using CPLEX.

Next, a number of prevalent modern metaheuristics was studied in solving the considered humanitarian logistics problem. The proposed approach was based on DE, eDE, PSO, and heterogeneous/homogeneous APs consisting of combinations of these algorithms. Proper modifications and refinements were introduced to tackle the special requirements of the test problems. From the obtained results, it was concluded that APs based on eDE offer remarkable performance efficiency and solution quality. Also, it became evident that APs can offer crucial insight in gathering information regarding the most appropriate metaheuristic for the problem at hand. Future work will enrich the study of APs by employing larger and more diverse collections of metaheuristics, in order to efficiently deal with problems of higher complexity.

Chapter 5

New Parallel Forecasting-based Algorithm Portfolio

5.1 Introduction

An essential issue in designing efficient algorithm portfolios is the resources allocation plan. In the previous chapter, this issue was investigated by proposing a trading-based mechanism for the online distribution of the available computational budget among the constituent algorithms. In this chapter, the resource allocation issue is investigated again, although in a different manner. Specifically, a new *parallel forecasting-based algorithm portfolio* is introduced, which employs a time series forecasting mechanism in order to predict the performance of the constituent algorithms and properly allocate the available computational resources. Three essential forecasting models, namely the simple exponential smoothing, the linear exponential smoothing, and the moving average are employed by the forecasting mechanism. The inspiration behind the pro-

posed model emanates from stock market environments where stockbrokers employ forecasting models that exploit the past behavior of the stocks to predict their future values.

5.2 Time Series Forecasting

Time series forecasting [155, 156] is concerned with the prediction of a model based on its historical observations. The predictions are the outcome of diverse methodologies ranging from empirical rules and Monte Carlo simulations to sophisticated statistical procedures. The main struggle in these approaches lies in the detection of statistical patterns in the available data. The forecasted variable can be considered as the aggregation of pure signal and noise, where the signal can be predicted while the noise introduces distortion to its values. A forecasting model shall be capable of capturing the underlying signal and extrapolating it in time. Naturally, this is far from easy task for complex signals.

Among the plethora of available models for time series forecasting, *exponential smoothing* [157, 158] is distinguished as a simple and straightforward approach. It exploits prior assumptions and observations to predict variables that can be either random processes or deterministic processes contaminated by noise. Central role in exponential smoothing plays the concept of *window functions*, which determines the scheme that assigns significance weights on the available observations.

The *simple moving average* [159, 160] is the simplest approach for data smoothing. If f_t denotes the observed quantity at time moment t, then this approach predicts the next value of f at time $t+1$ as the average of the k most recent observations,

$$\hat{f}_{t+1} = \frac{1}{k} \sum_{i=0}^{k-1} f_{t-i}. \tag{5.1}$$

In this model each one of the k observations has equal weight. Thus, increasing k results in declining impact of the most recent observations, thereby filtering more period-to-period noise. In turn, this produces smoother forecast series. The special case for $k = 1$ is also called the *random walk* model, which assumes that the coming value can be predicted solely by the last observed value,

$$\hat{f}_{t+1} = f_t. \tag{5.2}$$

This model has the property of following the exact path of the predicted variable but

with 1-period lag.

Alternatively to the simple moving average model, the *simple exponential smoothing* [159,161,162] model can be used. This is also known as the exponentially weighted moving average model and its main difference from the previous one lies in the assumption of gradually decreasing weights for older observations. According to this model, the next forecasted value is given as,

$$\hat{f}_{t+1} = \alpha\, f_t + (1-\alpha)\, \hat{f}_t, \tag{5.3}$$

with $\alpha \in [0,1]$ being a smoothing constant. Smaller values of α produce smoother forecast series. On the other hand, higher values assume that each observation introduces significant changes in the level of the series. The simple exponential smoothing model is among the most popular ones in business applications due to its simplicity and efficiency under a variety of conditions.

The aforementioned models assume that the time series does not have a distinguishable trend. Some of their variants can also admit a constant linear trend. Although this is adequate for 1-step-ahead predictions, it may be inefficient for predicting variables with varying growth rates. This deficiency is tackled through the *linear exponential smoothing* [159] model, which takes into consideration both level and trend of the series. Let l_t and r_t denote the local estimates of level and trend, respectively. Then, the level is computed according to the Holt's formula as follows,

$$l_t = \alpha\, f_t + (1-\alpha)\, (l_{t-1} - r_{t-1}), \tag{5.4}$$

while the trend is computed as,

$$r_t = \beta\, (l_t - l_{t-1}) + (1-\beta)\, r_{t-1}, \tag{5.5}$$

where $\alpha, \beta \in [0,1]$ are weighing factors. Eventually, the forecast for k steps ahead is computed as,

$$\hat{f}_{t+k} = l_t + k\, r_t. \tag{5.6}$$

Higher values of β assume rapid changes in the trend of the data, while smaller values better match slower changes.

5.3 Proposed model

A metaheuristic optimization algorithm can be considered as a stochastic system that evolves through time. Its state signal during a run can be defined as the function

value of the best solution achieved at each time moment or iteration. This signal can be tracked in order to predict the algorithm's performance through extrapolation. Randomized fluctuations due to stochasticity or adaptive changes in the algorithm's dynamic introduce noise to the performance signal. The noise can be detrimental for the quality of predictions based on simple observation or explicit if-then-else rules.

The main idea in the proposed approach lies in monitoring the performance signal of each constituent algorithm of an algorithm portfolio in order to predict its forthcoming performance, and use these predictions to allocate the available computational resources by favoring the most promising algorithms. Since the focus is mostly on high-performance computation environments, the number of processing units occupied by each algorithm of the portfolio constitutes the shared computational resources under consideration.

Putting it formally, let AP denote an algorithm portfolio consisting of M algorithms or different parameterizations of the same algorithm,

$$\text{AP} = \{a_1, a_2, \ldots, a_M\}.$$

Let $C \geqslant M$ denote the number of available processing units. These can be either physical cores or processing threads. Also, let $e_{\max}$ denote the maximum number of function evaluations available to the portfolio. This means that the portfolio is terminated when the total number of function evaluations spent by all its constituent algorithms exceeds $e_{\max}$.

In this model the computational budget is assigned to the portfolio in *batches*. The user specifies the total number, $b_{\max}$, of batches. At each batch, a fixed number of function evaluations,

$$e_{\text{batch}} = \frac{e_{\max}}{b_{\max}}, \tag{5.7}$$

is allocated to the portfolio and shall be completely consumed before the next batch assignment. These function evaluations are equally shared among the C processing units. Thus, the computational budget allocated to each processing unit at each batch is fixed and equal to,

$$e_{\text{proc}} = \frac{e_{\text{batch}}}{C}. \tag{5.8}$$

Each processing unit executes one of the algorithms of the portfolio during each batch. Thus, the main decision issue is the allocation of algorithms to processing units, i.e., how many instances of each algorithm are going to be allocated on the available

processing units. Let p_b^i denote the number of processing units running algorithm a_i during batch b. This means that each one of these processing units executes the specific algorithm independently of the rest. Obviously, it shall hold that,

$$C = \sum_{i=1}^{M} p_b^i, \quad b \in \{1, 2, \ldots, b_{\max}\}. \tag{5.9}$$

At the end of batch b, the best performance achieved by each algorithm is recorded. This consists of the best solution value achieved by any instance of the algorithm from the beginning of the portfolio's execution. Using this information, forecasting methods are used to predict the performance of each algorithm in the forthcoming $b+1$ batch. According to the predicted performance, a new number of processing units that will be occupied by each algorithm in the next batch is determined.

Specifically, let f_b^i denote the overall best solution value detected by algorithm a_i at the end of batch b, regardless of the number of processing units it has occupied so far. The objective value is assumed to be strictly non-negative, which is the case in the minimization problems in this chapter. Also, let $\hat{f}_b^i$ be the corresponding predicted value of the algorithm regardless of the employed forecasting model. The sets,

$$H_b^i = \left\{ f_1^i, \hat{f}_2^i, f_2^i, \hat{f}_3^i, f_3^i, \ldots, \hat{f}_b^i, f_b^i \right\}, \quad i \in \{1, 2, \ldots, M\},$$

contain all the available (actual and forecasted) performance data achieved by each algorithm a_i up to batch b. Additional information may be also included in these sets depending on the selected forecasting method, e.g., trend values for the linear exponential smoothing model. Then, this information is used to predict the performance of each algorithm a_i in the next batch $b+1$. Let,

$$\hat{f}_{b+1}^i = \texttt{Forecast}\left(H_b^i\right), \quad i \in \{1, 2, \ldots, M\},$$

be the forecasted solution values of the algorithms for the next batch using any of the forecasting models. Then, the fraction of processing units that will be occupied by algorithm a_i in the next batch $b+1$ is given by the normalized inverse of its predicted performance value,

$$\eta_{b+1}^i = \frac{1/\hat{f}_{b+1}^i}{\sum_{j=1}^{M} 1/\hat{f}_{b+1}^j}, \quad i \in \{1, 2, \ldots, M\}. \tag{5.10}$$

Proportionally to these fractions, the actual number of processing units that will host each algorithm in the next batch is determined through a resource allocation plan,

$$p_{b+1}^i = \texttt{Allocate}\left(\eta_{b+1}^i, C\right), \quad i \in \{1, 2, \ldots, M\}, \tag{5.11}$$

Algorithm 5.1 Forecasting-based algorithm portfolio

Input: Algorithms $a_1, \ldots, a_M$, number of processing units C, number of batches $b_{\max}$, computational budget $e_{\max}$

Output: Overall best solution

set e_{proc} according to Eq. (5.8) and $b \leftarrow 0$
/ processing units equally shared in 1st batch */*
set $\eta_{b+1}^i \leftarrow 1/M$ for all $i = 1, \ldots, M$
$p_{b+1}^i \leftarrow \texttt{Allocate}\left(\eta_{b+1}^i, C\right)$ for all $i = 1, \ldots, M$
$H_b^i \leftarrow \emptyset$ for all $i = 1, \ldots, M$
/ loop on the number of batches */*
for $(b = 1 \ldots b_{\max})$ **do**
  / parallel execution */*
  $\texttt{ExecuteAlgorithm}\left(a_i, p_b^i, e_{\text{proc}}\right)$ for all $i = 1, \ldots, M$
  for $(i = 1 \ldots M)$ **do**
    $\texttt{UpdateBest}\left(f_b^i\right)$
    $H_b^i \leftarrow H_{b-1}^i \cup \{f_b^i\}$
    $\hat{f}_{b+1}^i \leftarrow \texttt{Forecast}\left(H_b^i\right)$
    $H_b^i \leftarrow H_b^i \cup \{\hat{f}_{b+1}^i\}$
  end for
  compute η_{b+1}^i and p_{b+1}^i according to Eqs. (5.10) and (5.11) for all $i = 1, \ldots, M$
end for
return overall best solution

taking care that Eq. (5.9) holds. For example, a simple allocation procedure may directly set p_{b+1}^i as the decimal value $\eta_{b+1}^i C$ rounded to the nearest integer. However, correction may be needed since the outcome of the rounded quantities is not guaranteed to be equal to C (it can smaller or larger). In the implementation, the quantities,

$$\left\lfloor \eta_{b+1}^i C \right\rfloor,$$

were alternatively used to make a first assignment of algorithms to processing units. This approach always leaves spare processing units, which are then additionally assigned to the algorithms. Specifically, the algorithm with the best fraction value gets the first spare processing unit, the second best algorithm gets the second spare pro-

cessing unit, and this is continued in the same manner until all spare processing units have been equipped with an algorithm.

The proposed procedure is given in the pseudocode of Algorithm 5.1. In a parallel master-slave environment, better work division is attained when the master processing unit is devoted to bookkeeping, forecasting, and resources allocation, while the rest of the processing units (slave units) are devoted to algorithm execution. In this framework, the pseudocode can be executed on the master unit except step 9, which can be concurrently executed on each slave.

Another issue of interest is the minimum number of processing units occupied by each algorithm. Specifically, it is widely perceived that the efficiency of algorithm portfolios stems from the inclusion of algorithms of different characteristics. Complementarity of the constituent algorithms has been widely recognized as performance booster of the portfolio since weaknesses of one approach are addressed by another. Thus, the `Allocate`( ) function in Eq. (5.11) shall preserve that p_{b+1}^{i} does not vanish for any algorithm, rendering it inactive. In order to avoid this potential deficiency, it is recommended to firstly assign each one of the M algorithm to one processing unit regardless of its predicted performance, and then allocate the rest $C - M$ processing units to the algorithms according to the proposed procedure described above.

It shall be noticed that the proposed algorithm portfolio model does not exceed the computational budget of function evaluations provided by the user. It rather exploits it more efficiently by favoring the best-performing algorithms, which occupy more processing units at each batch and, eventually, perform higher number of function evaluations than the rest. Nevertheless, the total computational cost of the portfolio remains fixed to the predefined value of the user. This is a significant property for algorithms executed in parallel environments where execution time shall be predetermined, while exceeding it may impose penalties or additional charges.

5.4 Application in Circulant Weighing Matrices

In this section, the forecasting-based algorithm portfolio is applied on a challenging combinatorial problem, namely the detection of circulant weighing matrices [22]. Details about the considered problem are presented in Section 4.3.

The experimental analysis was conducted in two stages. In the first stage, proof of concept was the main goal. For this purpose, a reasonable experimental configuration

Table 5.1: Experimental configuration.

Description	Notation	Value(s)
Test problems	CW(n, k^2)	CW$(48, 36)$, CW$(52, 36)$, CW$(57, 49)$ CW$(62, 16)$, CW$(84, 16)$, CW$(112, 16)$
Tabu search algorithms	a_1	$s_{TL} = 2$ (tabu list size), $T_{\text{noimp}} = 100$ (iterations before restarting)
	a_2	$s_{TL} = 10$, $T_{\text{noimp}} = 100$
	a_3	$s_{TL} = 2$, $T_{\text{noimp}} = 1000$
	a_4	$s_{TL} = 10$, $T_{\text{noimp}} = 1000$
Algorithm portfolios	PL	No forecasting, equally shared resources (plain portfolio)
	RW	Simple moving average, $k = 1$
	M.3	Simple moving average, $k = 3$
	M.10	Simple moving average, $k = 10$
	S.3	Simple exp. smoothing, $\alpha = 0.3$
	S.8	Simple exp. smoothing, $\alpha = 0.8$
	L.3.3	Linear exp. smoothing, $\alpha = 0.3$, $\beta = 0.3$
	L.3.8	Linear exp. smoothing, $\alpha = 0.3$, $\beta = 0.8$
	L.8.3	Linear exp. smoothing, $\alpha = 0.8$, $\beta = 0.3$
	L.8.8	Linear exp. smoothing, $\alpha = 0.8$, $\beta = 0.8$
Num. of batches	$b_{\max}$	500 (1st stage), 100, 500, 1000 (2nd stage)
Num. of processing units	C	41 (1st stage), 21, 41, 61 (2nd stage)
Num. of function evaluations	$e_{\max}$	10^{11}
Num. of experiments per case	g	100

was set and the proposed algorithm portfolios were applied under various parameterizations of the forecasting models. The two best-performing models distinguished in the first stage were further studied in a second stage of experiments with respect to their sensitivity on the number of batches used by the forecasting mechanism and the number of available processing units.

The complete experimental configuration is reported in Table 5.1. Specifically, six representative test problems were selected for experimentation with sequence lengths ranging from 48 up to 112 and various weights ranging from 4 to 7. All problems were selected from the provided list in [22]. Taking into consideration the findings from previous works [123] and the evidence reported in Section 4.3.3 of Chapter 4, the considered algorithm portfolios consisted solely of the efficient tabu search algorithm. Specifically, the four parameter combinations were considered that are produced by

Table 5.2: Maximum and minimum correlation coefficients between the portfolios' samples of function evaluations per test problem.

Problem	Max	Alg. Portfolios			Min	Alg. Portfolios		
CW$(48, 36)$	0.230	L.3.8	-	L.8.8	-0.210	RW	-	L.8.8
CW$(52, 36)$	0.237	PL	-	L.3.3	-0.196	RW	-	S.3
CW$(57, 49)$	0.362	PL	-	L.3.3	-0.220	M.3	-	M.10
CW$(62, 16)$	0.162	L.3.3	-	L.8.8	-0.157	S.3	-	L.3.8
CW$(84, 16)$	0.264	RW	-	S.8	-0.176	S.3	-	S.8
CW$(112, 16)$	0.354	L.3.8	-	L.8.3	-0.183	RW	-	L.8.3

tabu list size $s_{TL} \in \{2, 10\}$, and number of non-improving iterations before restarting, $T_{\text{noimp}} \in \{100, 1000\}$. The four constituent algorithms are denoted as a_1, a_2, a_3, and a_4, according to the notation reported in Table 5.1.

The three forecasting models described in Section 5.2 were employed in the algorithm portfolios under various parameters. Specifically, the simple moving average model was applied by averaging $k \in \{1, 3, 10\}$ previous observations, with the special case $k = 1$ corresponding to the random walk model. These models are denoted as RW, M.3, and M.10, respectively. The simple exponential model was considered with a low and a high level of its parameter, namely $\alpha \in \{0.3, 0.8\}$, denoted as SES.3 and SES.8, respectively. The linear exponential model was considered with four combination of its parameters, namely $\alpha, \beta \in \{0.3, 0.8\}$, denoted as L.3.3, L.3.8, L.8.3, and L.8.8 as reported in Table 5.1. Finally, the plain portfolio without any forecasting, which allocates equal amount of computational resources to its constituent algorithms, was considered as the baseline approach and it is henceforth denoted as PL.

In the experiments each portfolio was executed on each test problem until the maximum computational budget of $e_{\text{max}} = 10^{11}$ function evaluations was exceeded or an optimal solution was found. An optimal solution is a sequence $\mathbf{x}$ where the objective function $f(\mathbf{x})$ of Eq. (4.5) vanishes. The number of batches for forecasting and resources allocation was set to $b_{\text{max}} = 500$, and the number of processing units was equal to $C = 41$ in the 1st stage of experimentation. This choice allowed the portfolios to conduct an adequate number of function evaluations before revising their resources allocation schedule. Note that one of the processing units plays the role of the master where bookkeeping procedures are executed, while the rest of the processing units

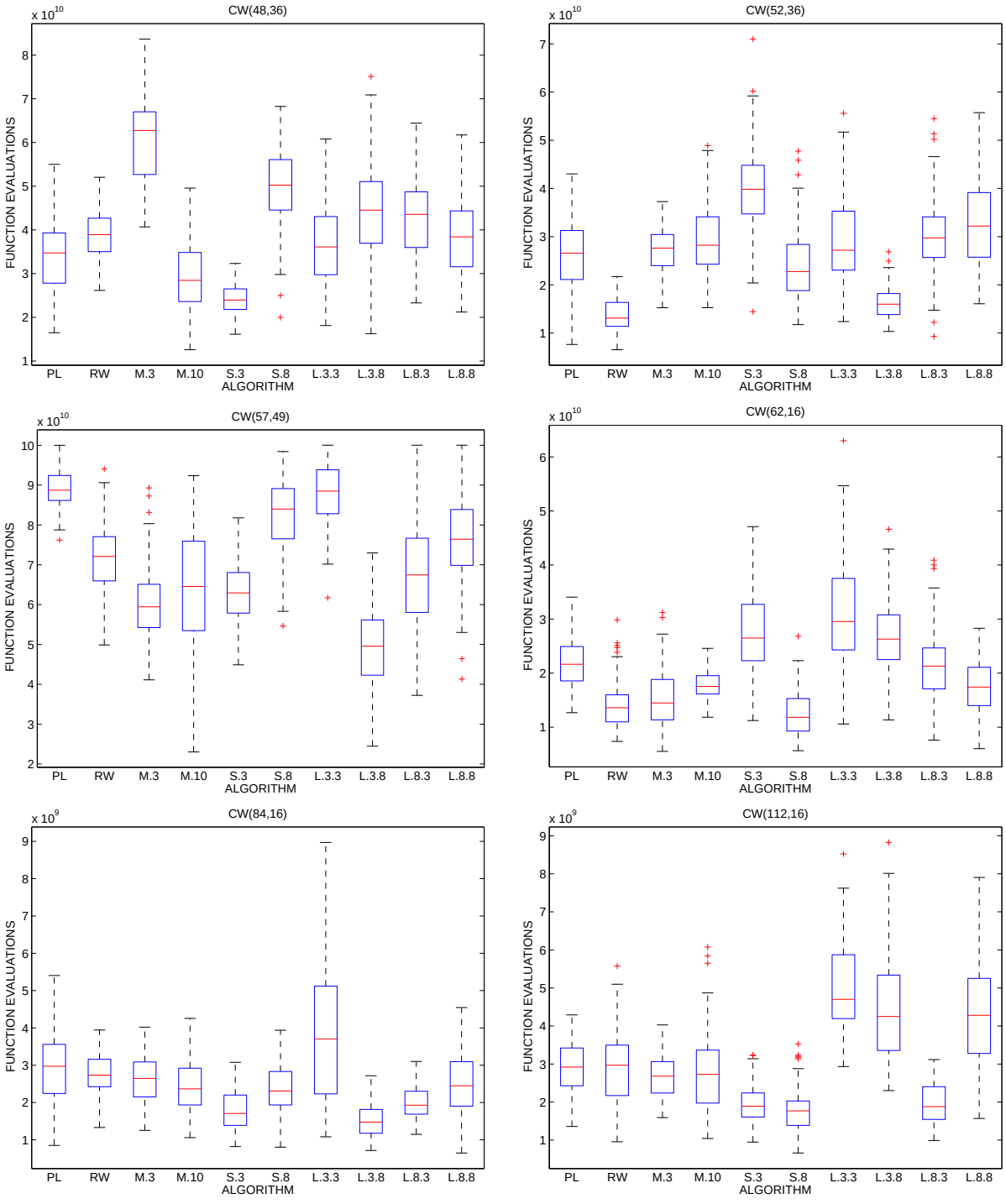


Figure 5.1: Boxplots of the number of function evaluations per problem and algorithm portfolio.

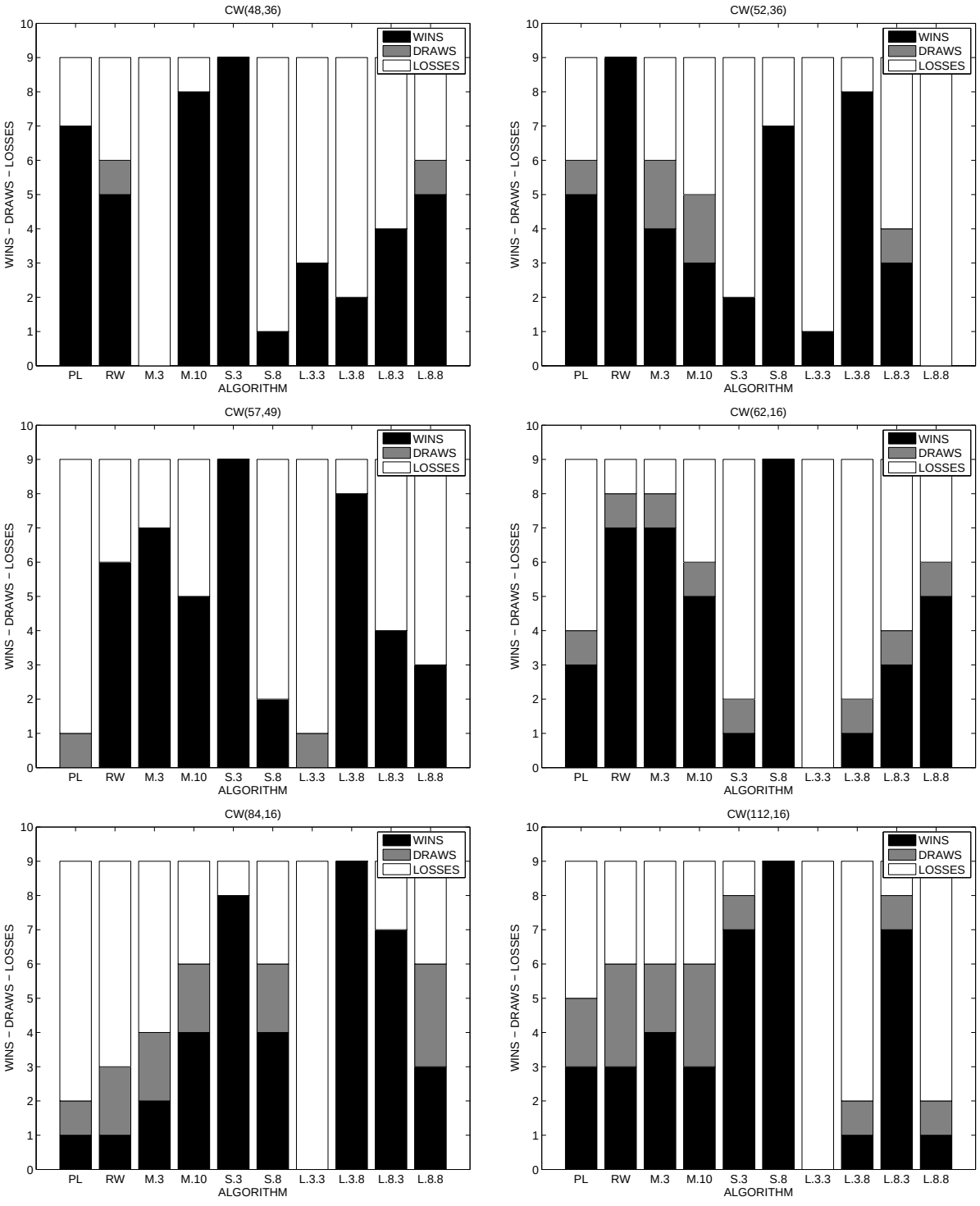


Figure 5.2: Number of wins, draws, and losses per problem and algorithm portfolio.

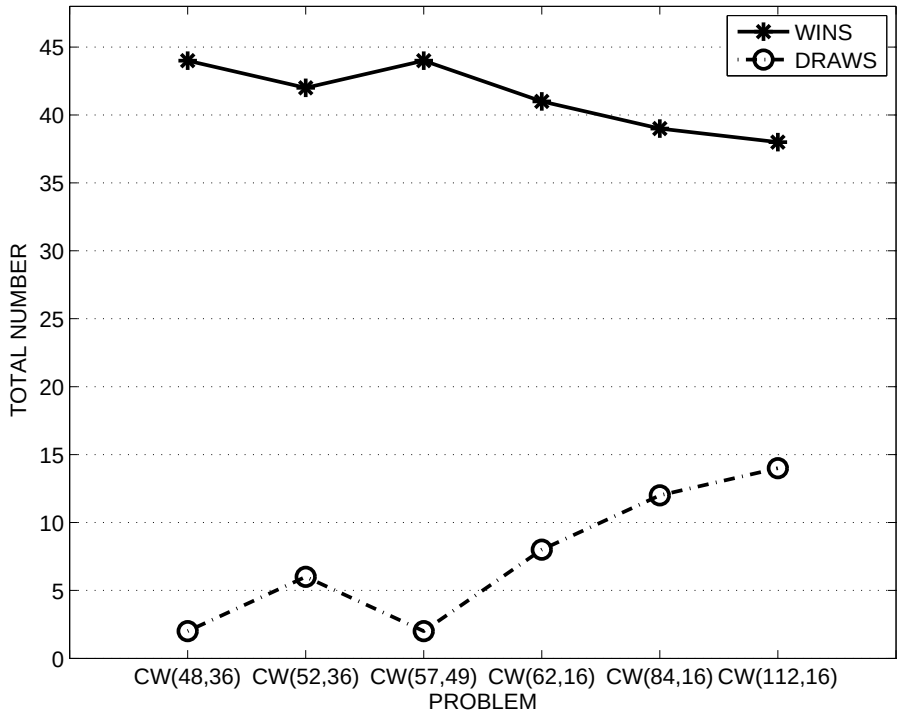


Figure 5.3: Total number of wins and draws of all algorithm portfolios per test problem.

simply execute their assigned algorithms. In the 2nd stage of experiments, the two variables assumed different values, namely $b_{\max} \in \{10, 500, 1000\}$ and $C \in \{21, 41, 61\}$, in order to explore the impact of these choices on the portfolio's performance. All runs were conducted in a mixed environment consisting of the `saw` cluster of the Sharcnet consortium along with a number of multi-core servers consisting of Intel© i7 processors.

A number of $g = 100$ independent experiments per algorithm portfolio and test problem were conducted, while statistical hypothesis testing was applied for head-to-head comparisons of the portfolios. The quantities of interest were the number of successful experiments, which reflects the effectiveness of the portfolio, and the average number of function evaluations spent, which reflects its efficiency. In order to facilitate comparisons between the algorithms, pairwise Wilcoxon rank-sum tests were conducted between each pair of algorithm portfolios on each test problem. Specifically, two portfolios were initially compared according to their number of successful runs. When the statistical test revealed significant difference between them, the most successful algorithm was awarded a win and the other one assumed a loss.

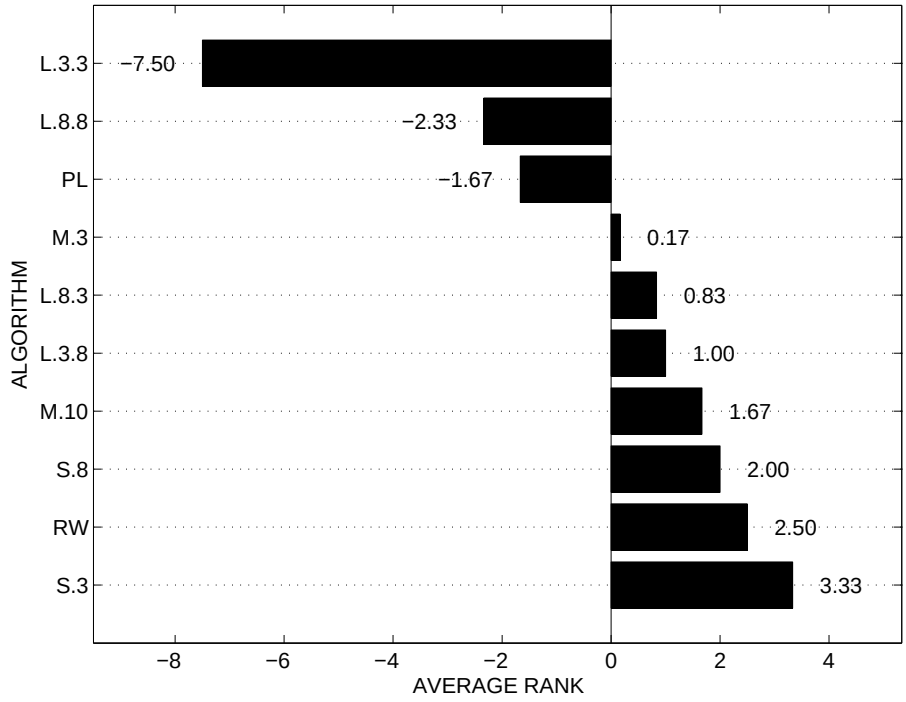


Figure 5.4: Average ranks of the algorithm portfolios over all test problems.

In case where this test revealed no significant difference, the corresponding test was conducted for the number of function evaluations. Again, a win was awarded to the winning algorithm and a loss to the other one. In case of insignificant differences in both tests, both portfolios assumed a draw. Let μ_w denote the wins, μ_l denote the losses, and μ_d denote the draws of an algorithm in the statistical tests. Since there are 10 distinct algorithm portfolios in the experiments, it shall clearly hold that,

$$\mu_w + \mu_l + \mu_d = 9.$$

In order to quantify the relevant differences between the portfolios, a *rank* is assigned to each one based on its wins and losses. The rank of a portfolio AP is defined as follows,

$$R(\text{AP}) = \mu_w - \mu_l, \tag{5.12}$$

i.e., it is the difference between its number of wins and losses and lies in the range $[-9, 9]$ for each portfolio. This way, the portfolios can be sorted according to their ranks in order to identify the best one for each test problem individually. Also, the average rank over all test problems can offer insight on the overall performance of each portfolio.

Table 5.3: Number of successes, mean and standard deviation of function evaluations, as well as wins, draws, losses, and rank per algorithm portfolio for test problems CW$(48, 36)$, CW$(52, 36)$, and CW$(57, 49)$.

Problem	AP	Suc	Mean	St.Dev.	Wins	Draws	Losses	Rank
CW$(48, 36)$	PL	100	$3.40e+10$	$2.34e+10$	7	0	2	5
	RW	100	$3.91e+10$	$1.80e+10$	5	1	3	2
	M.3	82	$6.15e+10$	$2.90e+10$	0	0	9	−9
	M.10	100	$2.93e+10$	$2.19e+10$	8	0	1	7
	S.3	100	$2.43e+10$	$1.18e+10$	9	0	0	9
	S.8	79	$4.99e+10$	$3.17e+10$	1	0	8	−7
	L.3.3	90	$3.67e+10$	$3.18e+10$	3	0	6	−3
	L.3.8	90	$4.41e+10$	$3.66e+10$	2	0	7	−5
	L.8.3	100	$4.29e+10$	$3.04e+10$	4	0	5	−1
	L.8.8	100	$3.84e+10$	$2.67e+10$	5	1	3	2
CW$(52, 36)$	PL	100	$2.63e+10$	$2.18e+10$	5	1	3	2
	RW	100	$1.37e+10$	$9.53e+09$	9	0	0	9
	M.3	100	$2.74e+10$	$1.52e+10$	4	2	3	1
	M.10	100	$2.95e+10$	$1.94e+10$	3	2	4	−1
	S.3	100	$4.00e+10$	$2.73e+10$	2	0	7	−5
	S.8	100	$2.40e+10$	$1.96e+10$	7	0	2	5
	L.3.3	89	$2.91e+10$	$2.49e+10$	1	0	8	−7
	L.3.8	100	$1.62e+10$	$1.03e+10$	8	0	1	7
	L.8.3	100	$3.00e+10$	$2.59e+10$	3	1	5	−2
	L.8.8	91	$3.26e+10$	$2.55e+10$	0	0	9	−9
CW$(57, 49)$	PL	28	$8.87e+10$	$1.79e+10$	0	1	8	−8
	RW	67	$7.19e+10$	$2.57e+10$	6	0	3	3
	M.3	69	$6.00e+10$	$3.25e+10$	7	0	2	5
	M.10	49	$6.46e+10$	$4.04e+10$	5	0	4	1
	S.3	90	$6.30e+10$	$2.55e+10$	9	0	0	9
	S.8	36	$8.22e+10$	$2.55e+10$	2	0	7	−5
	L.3.3	29	$8.79e+10$	$2.34e+10$	0	1	8	−8
	L.3.8	70	$4.90e+10$	$3.26e+10$	8	0	1	7
	L.8.3	39	$6.73e+10$	$4.27e+10$	4	0	5	−1
	L.8.8	39	$7.64e+10$	$3.30e+10$	3	0	6	−3

Table 5.4: Number of successes, mean and standard deviation of function evaluations, as well as wins, draws, losses, and rank per algorithm portfolio for test problems CW$(62, 16)$, CW$(84, 16)$, and CW$(112, 16)$.

Problem	AP	Suc	Mean	St.Dev.	Wins	Draws	Losses	Rank
CW$(62, 16)$	PL	100	$2.15e+10$	$1.22e+10$	3	1	5	-2
	RW	100	$1.41e+10$	$1.15e+10$	7	1	1	6
	M.3	100	$1.53e+10$	$1.69e+10$	7	1	1	6
	M.10	100	$1.76e+10$	$7.62e+09$	5	1	3	2
	S.3	100	$2.73e+10$	$2.46e+10$	1	1	7	-6
	S.8	100	$1.25e+10$	$1.26e+10$	9	0	0	9
	L.3.3	100	$3.14e+10$	$2.88e+10$	0	0	9	-9
	L.3.8	100	$2.71e+10$	$1.87e+10$	1	1	7	-6
	L.8.3	100	$2.17e+10$	$2.04e+10$	3	1	5	-2
	L.8.8	100	$1.75e+10$	$1.43e+10$	5	1	3	2
CW$(84, 16)$	PL	100	$2.93e+09$	$2.42e+09$	1	1	7	-6
	RW	100	$2.74e+09$	$1.87e+09$	1	2	6	-5
	M.3	100	$2.63e+09$	$2.07e+09$	2	2	5	-3
	M.10	100	$2.45e+09$	$1.99e+09$	4	2	3	1
	S.3	100	$1.79e+09$	$1.66e+09$	8	0	1	7
	S.8	100	$2.36e+09$	$1.96e+09$	4	2	3	1
	L.3.3	100	$3.84e+09$	$4.77e+09$	0	0	9	-9
	L.3.8	100	$1.52e+09$	$1.26e+09$	9	0	0	9
	L.8.3	100	$2.02e+09$	$1.30e+09$	7	0	2	5
	L.8.8	100	$2.47e+09$	$2.87e+09$	3	3	3	0
CW$(112, 16)$	PL	100	$2.90e+09$	$2.37e+09$	3	2	4	-1
	RW	100	$2.93e+09$	$2.97e+09$	3	3	3	0
	M.3	100	$2.67e+09$	$1.94e+09$	4	2	3	1
	M.10	100	$2.79e+09$	$3.00e+09$	3	3	3	0
	S.3	100	$1.94e+09$	$1.60e+09$	7	1	1	6
	S.8	100	$1.79e+09$	$2.01e+09$	9	0	0	9
	L.3.3	100	$5.01e+09$	$3.62e+09$	0	0	9	-9
	L.3.8	100	$4.43e+09$	$4.39e+09$	1	1	7	-6
	L.8.3	100	$1.98e+09$	$1.64e+09$	7	1	1	6
	L.8.8	100	$4.30e+09$	$3.70e+09$	1	1	7	-6

5.4.1 First stage of experiments: proof of concept

The 1st stage of experiments was conducted according to the experimental setting presented above. Figure 5.1 graphically depicts in boxplots the number of function evaluations spent by each algorithm portfolio for each test problem in 100 experiments. The number of successful runs, the mean and standard deviation of the required function evaluations, along with the wins, draws, losses, and the rank of each portfolio for each test problem are reported in Tables 5.3 and 5.4. The latter information is depicted also in Fig. 5.2. Figure 5.3 shows the total number of wins and draws of all algorithms observed per test problem. Finally, the average ranks of the portfolios over all test problems are illustrated in Fig. 5.4.

Some interesting observations can be made from the reported data. Figure 5.1 demonstrates significant variability on each portfolio's performance, depending on the corresponding test problem. Also, there seems to be no apparent correlation of performance between any two algorithms. Indeed, Table 5.2 reports the maximum and minimum values of the correlation coefficients computed on the samples of functions evaluations of the algorithm portfolios for each test problem. As can be seen, there are no crucial positive or negative correlations. The highest value is 0.362 between PL and L.3.3 in CW$(57, 49)$, while the lowest value is -0.220 between M.3 and M.10 for the same problem. Moreover, no pair of portfolios appears persistently in Table 5.2. These observations suggest that the portfolios' performance exhibits strong dependence on the corresponding problem and it is rather unrelated among them.

This conclusion can be verified also in Fig. 5.2. As can be seen there are no observable correlations between the algorithm portfolios with respect of their wins, draws, and losses in the statistical performance comparisons. Also, it can be clearly observed that even the highly-performing portfolios exhibit variable performance trends in all problems. However, it can be observed that the number of draws apparently increases with the problem's dimension. This is illustrated in Fig. 5.3, which shows the total number of wins and draws of all algorithms per test problem. Clearly, as dimension increases, the total number of draws exhibits a monotonically upward trend. This reveals the increasing difficulty of the problems, which is responsible for the trimmed performance differences among the portfolios.

Most interestingly, Fig. 5.2 depicts the average ranks of the algorithm portfolios, which quantifies the average performance of the portfolios over all test problems.

Firstly, it can be seen that the plain algorithm portfolio without forecasting, denoted as PL, was outperformed by the majority of the forecasting-based portfolios. This is a strong indication that performance forecasting can enhance the portfolios in terms of effectiveness and efficiency. Secondly, it can be clearly observed the superiority of simple exponential smoothing and simple moving average approaches against the linear exponential smoothing. Since the main difference of the latter is the ability of capturing short-term trends (varying rate of growth or cyclical patterns), it is reasonable to assume that the obtained portfolios performance does not exhibit such behavior. Instead, the ability of capturing linear trends and making accurate 1-step-ahead forecasts of the other two forecasting models resulted in their observed dominance.

Moreover, it can be seen that the two most successful approaches are the ones that use the least number of observations in the simple moving average model (RW case), and the smaller value of α in the simple exponential smoothing model (SES.3 case). In the first case, using less observations results in a model that filters less noise in the signal but responds rapidly to performance changes. In the second case, smaller values of α yield smoother forecast series at the cost of slower responses in performance changes, but it also alleviates abrupt changes in the forecasted values due to temporal variations of the observed performance.

This reveals the two essential but conflicting properties that a forecasting-based algorithm portfolio shall possess, namely,

(a) the ability to timely respond to new achievements of the constituent algorithms, and

(b) the alleviation of instant changes in the portfolios' dynamic due to possibly temporal achievements.

Appropriate trade-off between the two can offer tangible benefits to the portfolio. Evidently, the obtained results in the experiments show that property (b) can offer performance boost since SES.3 is 33% better than RW in terms of its average rank as can be seen in Fig. 5.2.

The two best-performing portfolios were subjected to further experimentation in order to probe their sensitivity with respect to two parameters, namely the number of batches and the number of processing units.

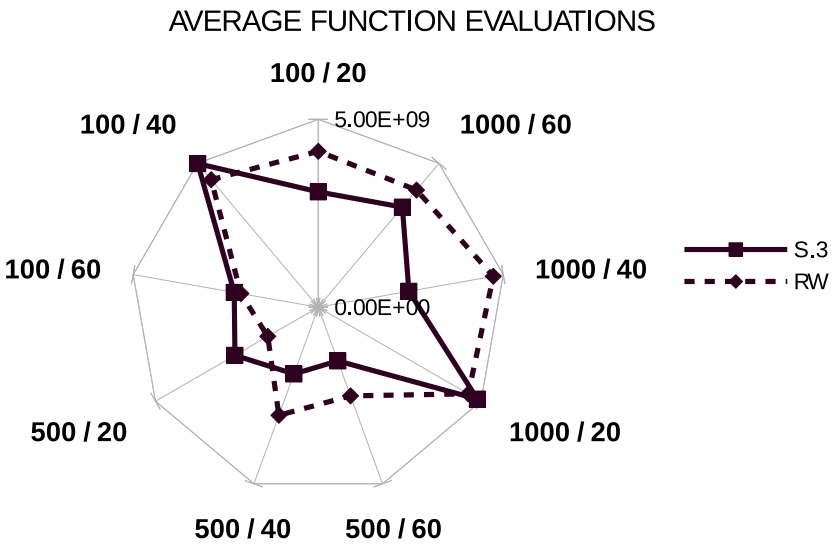


Figure 5.5: Average number of function evaluations per parameter setting.

5.4.2 Second stage of experiments: sensitivity analysis

In the second stage of experiments, the distinguished SES.3 and RW algorithm portfolios were considered for further investigation. The test problem of highest dimension, namely CW$(112, 16)$, was selected as the testbed for the sensitivity analysis of the portfolios with respect to the number of batches, $b_{\max}$, and the number of processing units, C. As reported in Table 5.1, three levels were considered for each variable, namely,

$$b_{\max} \in \{100, 500, 1000\}, \quad C \in \{21, 41, 61\}.$$

Note that C contains also the master processing unit, thus the actual number of the available computational resources allocated to the constituent algorithms of the portfolios are 20, 40, and 60, respectively. Henceforth, the notation

$$\mathrm{AP}/b_{\max}/C$$

is used, where $\mathrm{AP} \in \{\mathrm{SES.3}, \mathrm{RW}\}$, to denote the corresponding portfolio with the specific setting of $b_{\max}$ batches and C processing units. The total number of function evaluations was equal to the one of the 1st stage of experiments for consistency reason.

The analysis of the portfolios was similar to the one in the previous section. A total number of 100 runs were conducted for each portfolio and parameters combination, extracting the same statistical information and relative performance data, namely

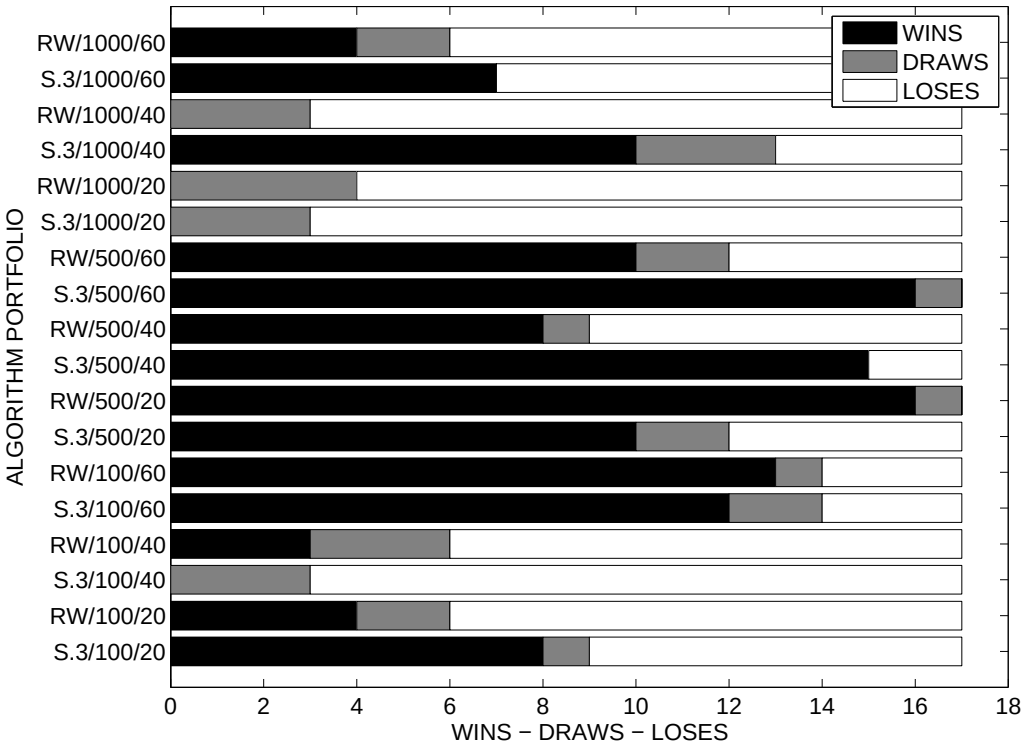


Figure 5.6: Number of wins, draws, and losses per parameter setting.

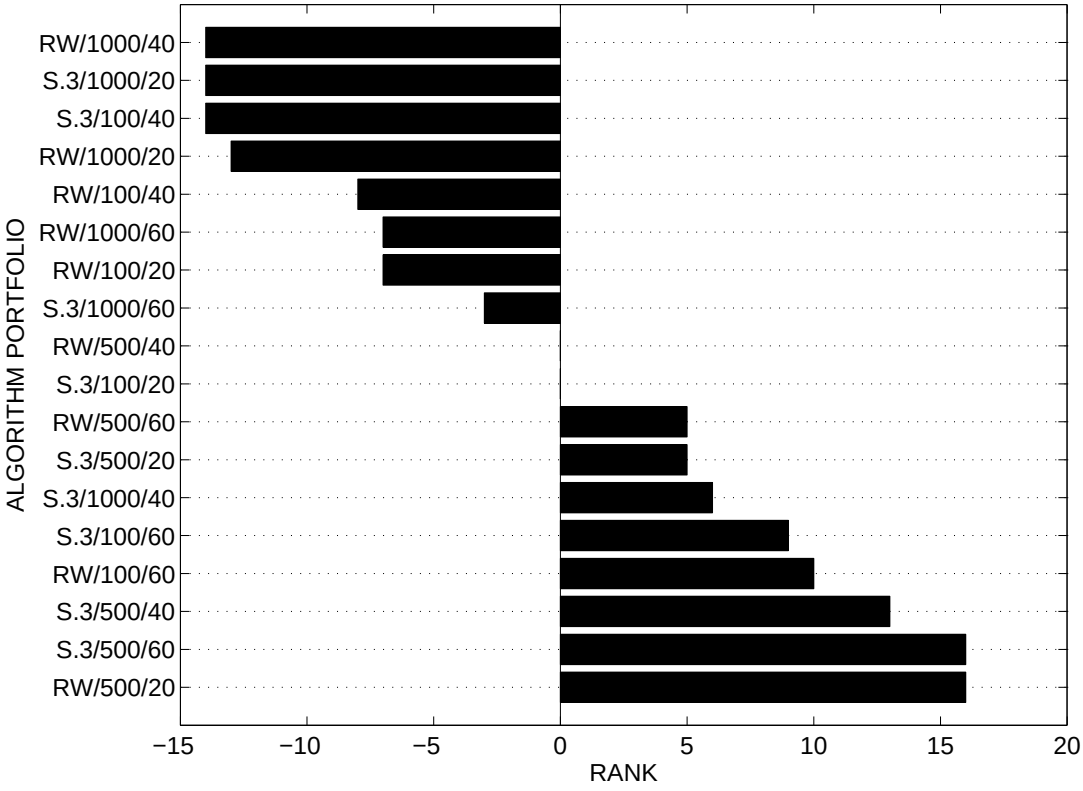


Figure 5.7: Ranks of the algorithm portfolios per parameter setting.

Table 5.5: Average number of function evaluations allocated to each processing unit per batch.

		SES.3	modif.	RW	modif.
Best 8 AP	(rank > 0)	4716981.13		6521739.27	
Worst 8 AP	(rank < 0)	3571428.57	(-24.3%)	3968253.96	(-39.2%)

Table 5.6: Two-way ANOVA table for SES.3 and RW. Processing units correspond to rows and number of batches correspond to columns.

AP	Source	SS	df	MS	F	p-value
SES.3	Columns	$4.7552e+20$	2	$2.3776e+20$	295.4936	0
	Rows	$1.8116e+20$	2	$9.0581e+19$	112.5764	0
	Interaction	$5.6473e+20$	4	$1.4118e+20$	175.4647	0
	Error	$7.1691e+20$	891	$8.0462e+17$		
	Total	$1.9383e+21$	899			
RW	Columns	$6.6276e+20$	2	$3.3138e+20$	408.2137	0
	Rows	$2.1044e+20$	2	$1.0522e+20$	129.6189	0
	Interaction	$2.5925e+20$	4	$6.4812e+19$	79.8390	0
	Error	$7.2330e+20$	891	$8.1178e+17$		
	Total	$1.8557e+21$	899			

wins, losses, draws, and rank. Figure 5.5 illustrates the average number of function evaluations per algorithm portfolio and parameter setting. Figures 5.6 and 5.7 report the corresponding number of wins, draws, losses, and the ranks of each portfolio, respectively. Table 5.5 reports the average number of function evaluations allocated to the best-performing SES.3 and RW portfolios. Eventually, Table 5.6 reports the statistical information of two-way analysis of variance (ANOVA) for each portfolio with respect to the values of batches and processing units, which aims at revealing the impact of each variable and their possible interactions.

The results offer interesting evidence. As can be seen in Fig. 5.5, SES.3 was proved to be more efficient in most cases, requiring smaller average numbers of function evaluations. This is observed especially for higher number of batches and higher number of processing units. Note that, according to Eqs. (5.7) and (5.8) under fixed computational budget, an increase in the number of batches or processing units results

in fewer number of function evaluations assigned to each processing unit at each resources allocation cycle. Thus, it seems that SES.3 gained more benefits from RW when frequent reallocations occur. This is reasonable, since RW is expected to closely follow changes in the performance of constituent algorithms, thereby rapidly changing the allocated resources from one batch to the next. This property can introduce fluctuations in the observed performance signals of the algorithms since their total exploration dynamic becomes highly variable between two consequent batches. On the other hand, the smoother transition from one allocation state to another in SES.3 evidently avoids that deficiency.

Figure 5.6 and, especially, Fig. 5.7 are pretty revealing regarding the most dominant approaches. As can be seen in Fig. 5.7, the 8 best-performing approaches with positive ranks (more wins than losses) consist of 5 instances of SES.3 and just 3 of RW, while exactly the opposite holds for the worst 8 approaches with negative ranks (more losses than wins). Table 5.5 reports the average number of function evaluations that is allocated to each processing unit for the SES.3 and RW portfolios that belong to the best 8 and, respectively, to the worst 8 portfolios depicted in Fig. 5.7. As can be seen, the best SES.3 approaches have an average that is around 24% higher than the less successful ones. Similarly, the successful RW approaches have an average that is approximately 39% higher than their worst counterparts. This evidence corroborates the previous conjecture.

In order to detect possible performance similarities and interactions between the different parameter settings, two-way ANOVA was conducted for SES.3 and RW individually. Table 5.6 reports the complete ANOVA table for both cases. In this table, sum of squares is denoted as SS, degrees of freedom is denoted as df, MS stands for mean squares and F is the F ratio. As can be seen, all the obtained p-values were equal to zero. This means that there were no statistical similarities of performance among different settings of either the number of batches or the number of processing units. Also, there was no observable interaction among them. This shows that the selected parameterization for these two variables can result in quite distinct performance profiles of the portfolios.

Nevertheless, SES.3 with 500 batches and 60 processing units shared the best position in Fig. 5.7 with the RW approach with 500 batches and 20 processing units. Both achieved almost 19% increased rank values than the next best approach. This is observed also in Fig. 5.6, where it can be seen that these two portfolios are the only

ones without losses, while they exhibit only one draw from the comparison between them.

5.5 Synopsis

A new forecasting-based algorithm portfolio model was proposed. Time series forecasting was employed to predict the performance of the portfolio's constituent algorithms. Then, computational resources (processing units) were allocated to the algorithms accordingly.

The challenging problem of detecting circulant weighing matrices was selected as the testbed for the empirical analysis of the proposed approach. Six problems of various sequence lengths and weights were used for this purpose. Moving average and exponential smoothing techniques were used with heterogeneous parallel algorithm portfolios based on the tabu search algorithm to detect matrices of various sizes and weights.

The experimental evidence and statistical analysis verified the ability of the proposed model to outperform standard algorithm portfolios. Also, sensitivity analysis provided useful insight regarding the impact of variables related to the model (number of batches) and the computation environment (number of processing units) on the portfolio's performance.

Further work will consider the application of the proposed approaches on open circulant weighing matrices problems. Also, the online adaptation of the employed forecasting model as well as its parameters will significantly enhance its performance and broaden its applicability.

Chapter 6

Concluding Remarks and Future Work

6.1 Concluding Remarks

Algorithm Portfolios define schemes that combine multiple algorithms into a joint framework while sharing the computational resources. The goals of the present thesis were the justification for the use of algorithm portfolios of metaheuristic algorithms in demanding optimization problems and the development of new parallel algorithm portfolio models. First, the necessity for appropriate computational resources allocation in modern metaheuristics was identified. Standard parallel algorithm portfolio models were introduced and applied on hard optimization problems such as the design of S-boxes in cryptography and the traffic light scheduling in smart cities environments.

Moreover, two new parallel algorithm portfolios with sophisticated resources allocation mechanisms were proposed. The first model introduces a trading-based mechanism that distributes the computational resources through a solution trading procedure. It was assessed on three demanding problems, namely the detection of circulant weighing matrices, the lot-sizing planning in production systems with returns and remanufacturing, and the transportation of commodities in humanitarian logistics. The second proposed model employs time series forecasting techniques to predict

the performance of the constituent algorithms of the portfolio and allocate the computational resources, accordingly. Its potential was demonstrated on the detection of circulant weighing matrices.

The experimental results offered strong evidence on the efficiency and effectiveness of the proposed algorithm portfolios on the studied discrete and mixed-integer problems. The portfolios usually achieved solutions of equal or higher quality than their constituent algorithms in significantly less time. Also, the experiments showed that it is highly desirable to construct portfolios with complementary algorithms. For example, it seems beneficial to form portfolios that harness both population-based and local-based algorithms to address specific types of problems. Additionally, the potential of the proposed budget allocation mechanisms in algorithm portfolios has been fully revealed. Specifically, the mechanisms can offer remarkable performance enhancement with respect to solution quality. Finally, the experimental results offered guidelines for the parameter setting of the portfolios.

6.2 Future work

It is highly interesting to investigate the synergy among the constituent algorithms of the portfolio and the impact of their complementarity on performance. Another important topic is the number of constituent algorithms. It is questionable whether adding algorithms to the algorithm portfolio results in subsequent performance improvement. Furthermore, it would be interesting to apply algorithm portfolios on optimization problems with many objectives. Additionally, it would be compelling to construct algorithm portfolios that also combine gradient-based optimization algorithms as there are no relevant works in literature. Finally, the performance of the algorithm portfolios could be further investigated on modern computing infrastructures using GPU-based techniques and tools.

Bibliography

[1] K. Voglis, K. E. Parsopoulos, and I. E. Lagaris, "Particle swarm optimization with deliberate loss of information," *Soft Computing*, vol. 16, no. 8, pp. 1373–1392, 2012.

[2] F. Glover and G. Kochenberger, *Handbook of metaheuristics*, vol. 57. Springer Science & Business Media, 2006.

[3] D. H. Wolpert and W. G. Macready, "No free lunch theorem for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 67–82, 1997.

[4] F. Peng, K. Tang, G. Chen, and X. Yao, "Population-based algorithm portfolios for numerical optimization," *IEEE Trans. on Evol. Comp.*, vol. 14, no. 5, pp. 782–800, 2010.

[5] A. Almakhlafi and J. Knowles, "Systematic construction of algorithm portfolios for a maintenance scheduling problem," in *Proceedings CEC 2013*, pp. 245–252, 2013.

[6] J. R. Rice, "The algorithm selection problem," *Advances in Computers*, vol. 15, pp. 65–118, 1976.

[7] M. Muñoz, M. Kirley, and S. Halgamuge, *The Algorithm Selection Problem on the Continuous Optimization Domain*. Springer, 2013.

[8] M. Črepinšek, S.-H. Liu, and M. Mernik, "Exploration and exploitation in evolutionary algorithms: A survey," *ACM Comput. Surv.*, vol. 45, no. 3, pp. 35:1–35:33, 2013.

[9] B. A. Huberman, R. M. Lukose, and T. Hogg, "An economics approach to hard computational problems," *Science*, vol. 27, pp. 51–53, 1997.

[10] C. P. Gomes and B. Selman, "Algorithm portfolio design: Theory vs. practice," in *Proceedings Thirteenth conference on Uncertainty in artificial intelligence*, pp. 190–197, 1997.

[11] A. S. Fukunaga, "Genetic algorithm portfolios," in *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, vol. 2, pp. 1304–1311, 2000.

[12] K. Tang, F. Peng, G. Chen, and X. Yao, "Population-based algorithm portfolios with automated constituent algorithms selection," *Information Sciences*, vol. 279, pp. 94–104, 2014.

[13] S. Yuen, C. Chow, X. Zhang, and Y. Lou, "Which algorithm should I choose: An evolutionary algorithm portfolio approach," *Applied Soft Computing*, vol. 40, pp. 654 – 673, 2016.

[14] N. Shukla, Y. Dashora, M. Tiwari, F. Chan, and T. Wong, "Introducing algorithm portfolios to a class of vehicle routing and scheduling problem," in *Proceedings OSCM 2007*, pp. 1015–1026, 2007.

[15] J. A. Vrugt, B. A. Robinson, and J. M. Hyman, "Self-adaptive multimethod search for global optimization in real-parameter spaces," *IEEE Trans. on Evol. Comp.*, vol. 13, no. 2, pp. 243–259, 2009.

[16] I. Yevseyeva, A. P. Guerreiro, M. T. M. Emmerich, and C. M. Fonseca, "A portfolio optimization approach to selection in multiobjective evolutionary algorithms," in *Proceedings PPSN 2014*, pp. 672–681, 2014.

[17] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Networks*, vol. IV, (Piscataway, NJ), pp. 1942–1948, IEEE Service Center, 1995.

[18] K. E. Parsopoulos and M. N. Vrahatis, *Particle Swarm Optimization and Intelligence: Advances and Applications*. Information Science Publishing (IGI Global), 2010.

[19] C. Carlet, "On the higher order nonlinearities of boolean functions and S-boxes, and their generalizations," in *Proceedings SETA 2008*, pp. 345–367, 2008.

[20] C. Carlet, "Vectorial boolean functions for cryptography," *Boolean models and methods in mathematics, computer science, and engineering*, vol. 134, pp. 398–469, 2010.

[21] J. García-Nieto, A. Carolina Olivera, and E. Alba, "Optimal cycle program of traffic lights with particle swarm optimization," *IEEE Trans. on Evol. Comp.*, vol. 17, no. 6, pp. 823–839, 2013.

[22] K. Arasu and A. Gutman, "Circulant weighing matrices," *Cryptogr. Commun.*, vol. 2, pp. 155–171, 2010.

[23] R. H. Teunter, Z. P. Bayindir, and W. Van den Heuvel, "Dynamic lot sizing with product returns and remanufacturing," *International Journal of Production Research*, vol. 44, no. 20, pp. 4377–4400, 2006.

[24] L. Özdamar, E. Ekinci, and B. Kucukyazici, "Emergency logistics planning in natural disasters," *Annals of Operations Research*, vol. 129, no. 1-4, pp. 217–245, 2004.

[25] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[26] A. G. Nikolaev and S. H. Jacobson, "Simulated annealing," in *Handbook of Metaheuristics* (M. Gendreau and J.-Y. Potvin, eds.), pp. 1–40, Springer, 2010.

[27] F. Glover, "Tabu search - part I," *ORSA Journal on Computing*, vol. 1, pp. 190–206, 1989.

[28] F. Glover, "Tabu search - part II," *ORSA Journal on Computing*, vol. 2, pp. 4–32, 1990.

[29] M. Gendreau and J.-Y. Potvin, "Tabu search," in *Handbook of Metaheuristics* (M. Gendreau and J.-Y. Potvin, eds.), pp. 41–59, Springer, 2010.

[30] F. Glover and M. Laguna, *Tabu Search*. Norwell, MA, USA: Kluwer Academic Publishers, 1997.

[31] D. Pham and D. Karaboga, *Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*. London: Springer-Verlag, 2000.

[32] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search: Framework and applications," in *Handbook of Metaheuristics* (M. Gendreau and J.-Y. Potvin, eds.), pp. 363–397, Springer, 2010.

[33] N. Mladenovic and P. Hansen, "Variable neighborhood search," *Computers and Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[34] P. Hansen, N. Mladenović, J. Brimberg, and J. A. Moreno Pérez, "Variable neighborhood search," in *Handbook of Metaheuristics* (M. Gendreau and J.-Y. Potvin, eds.), vol. 146, ch. 3, Springer, 2010.

[35] J. Brimberg and N. Mladenović, "A variable neighborhood algorithm for solving the continuous location-allocation problem.," *Stud. Locat. Anal.*, no. 10, pp. 1–12, 1996.

[36] K. Fleszar and K. Hindi, "Solving the resource-constrained project scheduling problem by a variable neighbourhood search.," *European Journal of Operational Research*, vol. 155, no. 2, pp. 402 – 413, 2004.

[37] P. Hansen, J. Brimberg, D. Urošević, and N. Mladenović, "Primal-dual variable neighborhood search for the simple plant-location problem.," *INFORMS Journal on Computing*, vol. 19, no. 4, pp. 552–564, 2007.

[38] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optimization*, vol. 11, pp. 341–359, 1997.

[39] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 4–31, 2011.

[40] A. W. Mohamed, "RDEL: Restart differential evolution algorithm with local search mutation for global numerical optimization," *Egyptian Informatics Journal*, vol. 15, no. 3, pp. 175–188, 2014.

[41] W. Ma, M. Wang, and X. Zhu, "Improved particle swarm optimization based approach for bilevel programming problem-an application on supply chain model," *International Journal of Machine Learning and Cybernetics*, vol. 5, no. 2, pp. 281–292, 2014.

[42] R. Poli, "An analysis of publications on particle swarm optimisation applications," Tech. Rep. CSM-649, University of Essex, Department of Computer Science, UK, 2007.

[43] S. Rana, S. Jasola, and R. Kumar, "A boundary restricted adaptive particle swarm optimization for data clustering," *International Journal of Machine Learning and Cybernetics*, vol. 4, no. 4, pp. 391–400, 2013.

[44] N. Tian and C.-H. Lai, "Parallel quantum-behaved particle swarm optimization," *International Journal of Machine Learning and Cybernetics*, vol. 5, no. 2, pp. 309–318, 2014.

[45] X. Wang, Y. He, L. Dong, and H. Zhao, "Particle swarm optimization for determining fuzzy measures from data," *Information Sciences*, vol. 181, no. 19, pp. 4230–4252, 2011.

[46] J. Kennedy, "Small worlds and mega–minds: Effects of neighborhood topology on particle swarm performance," in *Proc. IEEE Congr. Evol. Comput.*, (Washington, D.C., USA), pp. 1931–1938, IEEE Press, 1999.

[47] P. N. Suganthan, "Particle swarm optimizer with neighborhood operator," in *Proc. IEEE Congr. Evol. Comput.*, (Washington, D.C., USA), pp. 1958–1961, 1999.

[48] M. Clerc and J. Kennedy, "The particle swarm–explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. Evol. Comput.*, vol. 6, no. 1, pp. 58–73, 2002.

[49] I. C. Trelea, "The particle swarm optimization algorithm: Convergence analysis and parameter selection," *Information Processing Letters*, vol. 85, pp. 317–325, 2003.

[50] J. García-Nieto, E. Alba, and A. Carolina Olivera, "Swarm intelligence for traffic light scheduling: Application to real urban areas," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 2, pp. 274–283, 2012.

[51] C. h. Chen and L. H. Lee, *Stochastic simulation optimization: an optimal computing budget allocation*, vol. 1. World scientific, 2011.

[52] T. Bartz-Beielstein, D. Blum, and J. Branke, "Particle swarm optimization and sequential sampling in noisy environments," in *In Metaheuristics, Operations Research/Computer Science Interfaces Series*, vol. 39, pp. 261–273, Springer, 2007.

[53] H. Pan, L. Wang, and B. Liu, "Particle swarm optimization for function optimization in noisy environment," *Applied Mathematics and Computation*, vol. 181, no. 2, pp. 908–919, 2006.

[54] J. Rada-Vilela, M. Zhang, and M. Johnston, "Optimal computing budget allocation in particle swarm optimization," in *Proc. 2013 Genetic and Evolutionary Computation Conference (GECCO'13)*, (Amsterdam, Netherlands), pp. 81–88, 2013.

[55] S. Zhang, P. Chen, L. H. Lee, C. E. Peng, and C.-H. Chen, "Simulation optimization using the particle swarm optimization with optimal computing budget allocation," in *Proceedings of the 2011 Winter Simulation Conference*, pp. 4298–4309, 2011.

[56] P.-Y. Yin and J.-Y. Wang, "Optimal multiple-objective resource allocation using hybrid particle swarm optimization and adaptive resource bounds technique," *Journal of Computational and Applied Mathematics*, vol. 216, no. 1, pp. 73 – 86, 2008.

[57] R. Akbari and K. Ziarati, "A rank based particle swarm optimization algorithm with dynamic adaptation," *J. Comp. App. Math.*, vol. 235, no. 8, pp. 2694–2714, 2011.

[58] L.-Y. Wan and W. Li, "An improved particle swarm optimization algorithm with rank-based selection.," in *Proceedings of IEEE International Conference of Machine Learning and Cybernetics*, vol. 7, pp. 4090–4095, 2008.

[59] T. Bäck, D. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation*. New York: IOP Publishing and Oxford University Press, 1997.

[60] Y. Jin, M. Olhofer, and B. Sendhoff, "Evolutionary dynamic weighted aggregation for multiobjective optimization: Why does it work and how?," in *Proceedings GECCO 2001 Conference*, (San Francisco, CA), pp. 1042–1049, 2001.

[61] K. E. Parsopoulos and M. N. Vrahatis, "Particle swarm optimization method in multiobjective problems," in *Proceedins of the ACM 2002 Symposium on Applied Computing (SAC 2002)*, (Madrid, Spain), pp. 603–607, ACM Press, 2002.

[62] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont, *Evolutionary Algorithms for Solving Multi–Objective Problems*. New York: Kluwer, 2002.

[63] M. Lozano, D. Molina, and F. Herrera, "Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems," *Soft Computing*, vol. 15, no. 11, pp. 2085–2087, 2011.

[64] D. Whitley, M. Lunacek, and J. Knight, "Ruffled by ridges: How evolutionary algorithms can fail," in *Lecture Notes in Computer Science (LNCS)* (K. e. a. Deb, ed.), vol. 3103, pp. 294–306, Springer, 2004.

[65] J. Eshelman, "The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination," *Foundations of Genetic Algorithms*, pp. 265–283, 1991.

[66] A. Auger and N. Hansen, "A restart CMA evolution strategy with increasing population size," in *Proc. IEEE Congress On Evolutionary Computation*, (Edinburgh, UK), pp. 1769–1776, 2005.

[67] A. Duarte, R. Mart, and F. Gortazar, "Path relinking for large-scale global optimization," *Soft Comput.*, vol. 15, no. 11, pp. 2257–2273, 2011.

[68] M. Zambrano-Bigiarini, M. Clerc, and R. Rojas, "Standard particle swarm optimisation 2011 at cec-2013: A baseline for future pso improvements," in *Proceedings of the IEEE 2013 Congress on Evolutionary Computation*, (Mexico), pp. 2337–2344, 2013.

[69] P. Yadav, R. Kumar, S. K. Panda, and C. S. Chang, "An intelligent tuned harmony search algorithm for optimisation," *Information Sciences*, vol. 196, pp. 47–72, 2012.

[70] M. Omran and M. Mahdavi, "Global-best harmony search," *Applied Mathematics and Computation*, vol. 198, no. 2, pp. 643–656, 2008.

[71] R. Kumar, "Directed bee colony optimization algorithm," *Swarm and Evolutionary Computation*, vol. 17, no. 0, pp. 60–73, 2014.

[72] R. Akay, A. Basturk, A. Kalinli, and X. Yao, "Parallel population-based algorithm portfolios: An empirical study," *Neurocomputing*, 2017. In press.

[73] N. Shukla, A. Choudhary, P. Prakash, K. Fernandes, and M. Tiwari, "Algorithm portfolios for logistics optimization considering stochastic demands and mobility allowance," *International Journal of Production Economics*, vol. 141, pp. 146–166, 2013.

[74] M.-L. Cauwet, J. Liu, B. Rozière, and O. Teytaud, "Algorithm portfolios for noisy optimization," *Annals of Mathematics and Artificial Intelligence*, vol. 76, no. 1, pp. 143–172, 2016.

[75] J. F. Calderín, A. D. Masegosa, and D. A. Pelta, "An algorithm portfolio for the dynamic maximal covering location problem," *Memetic Computing*, pp. 1–11, 2016.

[76] S. Y. Yuen and X. Zhang, "On composing an algorithm portfolio," *Memetic Computing*, vol. 7, pp. 203–214, 2015.

[77] M. Misir, S. Handoko, and H. Lau, "Building algorithm portfolios for memetic algorithms," in *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO Comp '14, (New York, NY, USA), pp. 197–198, ACM, 2014.

[78] S. Yuen and X. Zhang, "Multiobjective evolutionary algorithm portfolio: Choosing suitable algorithm for multiobjective optimization problem," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1967–1973, 2014.

[79] E. Alba, *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience, 2005.

[80] C. Adams and S. Tavares, "The structured design of cryptographically good S-boxes," *Journal of Cryptology*, vol. 3, no. 1, pp. 27–41, 1990.

[81] M. Matsui, *On correlation between the order of S-boxes and the strength of DES*. Springer Berlin Heidelberg, 1995.

[82] E. F. Brickell, J. H. Moore, and M. R. Purtill, "Structure in the S-boxes of the des," in *Proceedings on Advances in cryptology—CRYPTO '86*, pp. 3–8, 1987.

[83] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2013.

[84] M. Matsui and A. Yamagishi, "A new method for known plaintext attack of feal cipher," in *Workshop on the Theory and Application of of Cryptographic Techniques*, pp. 81–91, 1992.

[85] E. Biham and A. Shamir, "Differential cryptanalysis of DES-like cryptosystems," *Journal of Cryptology*, vol. 4, no. 1, pp. 3–72, 1991.

[86] E. C. Laskari, C. M. Meletiou, and M. N. Vrahatis, "Utilizing evolutionary computation methods for the design of S-boxes," in *Proceedings International Conference on Computational Intelligence and Security 2006*, pp. 1299–1302, 2006.

[87] W. Millan, L. Burnett, G. Carter, A. Clark, and E. Dawson, "Evolutionary heuristics for finding cryptographically strong S-boxes," *Lecture Notes in Computer Science*, vol. 1726, pp. 263–274, 1999.

[88] J. A. Clark, J. L. Jacob, and S. Stepney, "The design of S-boxes by simulated annealing," *New Generation Computing*, vol. 23, no. 3, pp. 219–231, 2005.

[89] W. Millan, "Smart hill climbing finds better boolean functions," in *Proceedings Boolean Functions Workshop on Selected Areas on Cryptography, SAC 97*, pp. 50–63, 1997.

[90] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers and Operations Research*, vol. 13, no. 5, pp. 533–549, 1986.

[91] S. Picek, J. F. Miller, D. Jakobovic, and L. Batina, "Cartesian genetic programming approach for generating substitution boxes of different sizes," in *Proceedings GECCO 2015 (Companion Volume)*, pp. 1457–1458, 2015.

[92] K. Nyberg, "Perfect nonlinear S-boxes," in *Proceedings Workshop on the Theory and Application of Cryptographic Techniques*, pp. 378–386, 1991.

[93] C. Carlet and D. Cunsheng, "Nonlinearities of S-boxes," *Finite Fields and Their Applications*, vol. 13, no. 1, pp. 121–135, 2007.

[94] J. A. Clark, J. L. Jacob, and S. Stepney, "Searching for cost functions," in *Proceedings CEC 2004*, pp. 1517–1524, 2004.

[95] E. Alba, G. Luque, and S. Nesmachnow, "Parallel metaheuristics: recent advances and new trends," *International Transactions in Operational Research*, vol. 20, no. 1, pp. 1–48, 2013.

[96] S. Picek, M. Cupic, and L. Rotim, "A new cost function for evolution of S-boxes," *Evolutionary Computation*, vol. 24, no. 4, pp. 695–718, 2016.

[97] W. Millan, "How to improve the non-linearity of bijective S-boxes," in *Proceedings 3nd Australian Conference on Information Security and Privacy*, pp. 181–192, 1998.

[98] J. Fuller and W. Millan, "Linear redundancy in S-boxes," in *Proceedings FSE 2003*, pp. 74–86, 2003.

[99] G. Ivanov, N. Nikolov, and S. Nikova, "Reversed genetic algorithms for generation of bijective S-boxes with good cryptographic properties," *Cryptography and Communications*, vol. 8, no. 2, pp. 247–276, 2016.

[100] Y. Nawaz, K. C. Gupta, and G. Gong, "Algebraic immunity of S-boxes based on power mappings: Analysis and construction," *IEEE Transactions on Information Theory*, vol. 55, pp. 4263–4273, Sept 2009.

[101] "Cross zlín, edaptiva: Full-featured urban traffic management center, Technical report," 2015.

[102] "Aldridge traffic controllers, SCATS, Technical report," 2015.

[103] N. M. Rouphail, B. B. Park, and J. Sacks, "Direct signal timing optimization: strategy development and results. technical report," in *XI Pan American conference in Traffic and Transportation Engineering*, 2000.

[104] F. Teklu, A. Sumalee, and D. Watling, "A genetic algorithm approach for optimizing traffic control signals considering routing," *Comput. Aided Civil and Infrastruct. Eng.*, vol. 22, no. 1, pp. 31–43, 2007.

[105] J. Sánchez, M. Galán, and E. Rubio, "Applying a traffic lights evolutionary optimization technique to a real case: "las ramblas" area in santa cruz de tenerife," *IEEE Trans. Evol. Comput.*, vol. 12, no. 1, pp. 25–40, 2008.

[106] A. M. Turky, M. S. Ahmad, M. Z. M. Yusoff, and B. T. Hammad, "Using genetic algorithm for traffic light control system with a pedestrian crossing," in *Proceedings Fourth International Conference on Rough Sets and Knowledge Technology*, pp. 512–519, 2009.

[107] J. Chen and L. Xu, "Road-junction traffic signal timing optimization by an adaptive particle swarm algorithm," in *Proceedings 9th International Conference on Control, Automation, Robotics and Vision*, pp. 1–7, 2006.

[108] L. Peng, M. H. Wang, J. P. Du, and G. Luo, "Isolation niches particle swarm optimization applied to traffic lights controlling," in *Proceedings of the 48th IEEE Decision and Control Chinese Conference CDC/CCC*, pp. 3318–3322, 2009.

[109] S. Kachroudi and N. Bhouri, "A multimodal traffic responsive strategy using particle swarm optimization," in *Proceedings of the 12th IFAC Symposium on Transportation Systems*, 2009.

[110] J. García-Nieto, J. Ferrer, and E. Alba, "Optimising traffic lights with metaheuristics: Reduction of car emissions and consumption," in *IJCNN*, pp. 48–54, 2014.

[111] A. Carolina Olivera, J. García-Nieto, and E. Alba, "Reducing vehicle emissions and fuel consumption in the city by using particle swarm optimization," *Applied Intelligence*, vol. 42, no. 3, pp. 389–405, 2015.

[112] D. Krajzewicz, M. Bonert, and P. Wagner, "The open source traffic simulation package sumo," *RoboCup 2006*, 2006.

[113] K. Arasu and T. Gulliver, "Self-dual codes over fp and weighing matrices," *IEEE Transactions on Information Theory*, vol. 47, no. 5, pp. 2051–2055, 2001.

[114] W. van Dam, "Quantum algorithms for weighing matrices and quadratic residues," *Algorithmica*, vol. 34, pp. 413–428, 2002.

[115] C. Koukouvinos and J. Seberry, "Weighing matrices and their applications," *Journal of Statistical planning and inference*, vol. 62, no. 1, pp. 91–101, 1997.

[116] K. Arasu, J. Dillon, D. Jungnickel, and A. Pott, "The solution of the waterloo problem," *J. Comb. Theory, Ser. A*, vol. 71, pp. 316–331, 1995.

[117] P. Eades, *On the existence of orthogonal designs*. PhD thesis, Australian National University, Canberra, 1997.

[118] P. Eades and R. Hain, "On circulant weighing matrices," *Ars Comb.*, vol. 2, pp. 265–284, 1976.

[119] A. Geramita and J. Seberry, "Orthogonical designs: Quadratic forms and hadamard matrices," *Lecture Notes in Pure and Applied Mathematics*, 1979.

[120] M. Ang, K. Arasu, S. Ma, and Y. Strassler, "Study of proper circulant weighing matrices with weigh 9," *Discrete Math.*, vol. 308, pp. 2802–2809, 2008.

[121] K. Arasu, K. Leung, S. Ma, A. Nabavi, and D. Ray-Chaudhuri, "Determination of all possible orders of weight 16 circulant weighing matrices," *Finite Fields Appl.*, vol. 12, pp. 498–538, 2006.

[122] B. Schmidt and K. W. Smith, "Circulant weighing matrices whose order and weight are products of powers of 2 and 3," *Journal of Combinatorial Theory, Series A*, vol. 120, no. 1, pp. 275–287, 2013.

[123] M. Chiarandini, I. Kotsireas, C. Koukouvinos, and L. Paquete, "Heuristic algorithms for hadamard matrices with two circulant cores," *Theoretical Computer Science*, vol. 407, no. 1-3, pp. 274–277, 2008.

[124] J. Cousineau, I. Kotsireas, and C. Koukouvinos, "Genetic algorithms for orthogonal designs," *Australasian Journal of Combinatorics*, vol. 35, pp. 263–272, 2006.

[125] I. Kotsireas, K. Parsopoulos, G. Piperagkas, and M. Vrahatis, "Ant-based approaches for solving autocorrelation problems," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7461 LNCS, pp. 220–227, 2012.

[126] I. Kotsireas, "Algorithms and metaheuristics for combinatorial matrices," in *Handbook of Combinatorial Optimization* (P. Pardalos, D.-Z. Du, and R. L. Graham, eds.), pp. 283–309, Springer New York, 2013.

[127] I. Kotsireas, C. Koukouvinos, P. Pardalos, and D. Simos, "Competent genetic algorithms for weighing matrices," *Journal of Combinatorial Optimization*, vol. 24, no. 4, pp. 508–525, 2012.

[128] Y. Strassler, *The classification of circulant weighing matrices of weight 9.* PhD thesis, Bar-Ilan University, 1997.

[129] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings Sixth Symposium on Micro Machine and Human Science*, (Piscataway, NJ), pp. 39–43, 1995.

[130] C. Ribeiro and M. Resende, "Path-relinking intensification methods for stochastic local search algorithms," *Journal of Heuristics*, vol. 18, no. 2, pp. 193–214, 2012.

[131] F. Tasgetiren, A. Chen, G. Gencyilmaz, and S. Gattoufi, "Smallest position value approach," *Studies in Computational Intelligence*, vol. 175, pp. 121–138, 2009.

[132] B. Golany, J. Yang, and G. Yu, "Economic lot-sizing with remanufacturing options," *IIE Transactions*, vol. 33, no. 11, pp. 995–1004, 2001.

[133] P. Piñeyro and O. Viera, "The economic lot-sizing problem with remanufacturing: analysis and an improved algorithm," *Journal of Remanufacturing*, vol. 5, no. 1, p. 12, 2015.

[134] H. M. Wagner and T. M. Whitin, "Dynamic version of the economic lot size model," *Management Science*, vol. 5, no. 1, pp. 88—96, 1958.

[135] T. Schulz, "A new silver–-meal based heuristic for the single–-item dynamic lot sizing problem with returns and remanufacturing," *International Journal of Production Research*, vol. 49, no. 9, pp. 2519—2533, 2011.

[136] E. Moustaki, K. E. Parsopoulos, I. Konstantaras, K. Skouri, and I. Ganas, "A first study of particle swarm optimization on the dynamic lot sizing problem with product returns," in *Proceedings BALCOR 2013*, pp. 348–356, 2013.

[137] G. S. Piperagkas, I. Konstantaras, K. Skouri, and K. E. Parsopoulos, "Solving the stochastic dynamic lot-sizing problem through nature-inspired heuristics," *Computers & Operations Research*, vol. 39, no. 7, pp. 1555–1565, 2012.

[138] G. S. Piperagkas, C. Voglis, V. A. Tatsis, K. E. Parsopoulos, and K. Skouri, "Applying PSO and DE on multi-item inventory problem with supplier selection," in *The 9th Metaheuristics International Conference (MIC 2011)*, (Udine, Italy), pp. 359–368, 2011.

[139] H. R. Lourenco, "Job-shop scheduling: Computational study of local search and large-step optimization methods," *European Journal of Operational Research*, vol. 83, no. 2, pp. 347–364, 1995.

[140] A. Thomas and L. Kopczak, *From Logistics to Supply Chain Management - The Path Forward to the Humanitarian Sector*. Fritz Institute, 2005.

[141] A. Cozzolino, S. Rossi, and A. Conforti, "Agile and lean principles in the humanitarian supply chain," *Journal of Humanitarian Logistics and Supply Chain Management*, vol. 2, no. 1, pp. 16–33, 2012.

[142] L. N. Van Wassenhove, "Humanitarian aid logistics: Supply chain management in high gear," *Journal of the Operational Research Society*, vol. 57, no. 5, pp. 475–489, 2006.

[143] G. Galindo and R. Batta, "Review of recent developments in OR/MS research in disaster operations management," *European Journal of Operational Research*, vol. 230, no. 2, pp. 201–211, 2013.

[144] Y. Han, X. Guan, and L. Shi, "Optimization based method for supply location selection and routing in large-scale emergency material delivery," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 4, pp. 683–693, 2011.

[145] M. Huang, K. Smilowitz, and B. Balcik, "A continuous approximation approach for assessment routing in disaster relief," *Transportation Research Part B: Methodological*, vol. 50, pp. 20–41, 2013.

[146] A. Clark and B. Culkin, "A network transshipment model for planning humanitarian relief operations after a natural disaster," in *Decision Aid Models for Disaster Management and Emergencies*, vol. 7 of *Atlantis Computational Intelligence Systems*, pp. 233–257, Atlantis Press, 2013.

[147] M. Peng and H. Chen, "System dynamics analysis for the impact of dynamic transport and information delay to disaster relief supplies," in *2011 International Conference on Management Science and Engineering (ICMSE 2011)*, pp. 93–98, 2011.

[148] P. Van Hentenryck, R. Bent, and C. Coffrin, "Strategic planning for disaster recovery with stochastic last mile distribution," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, vol. 6140 of *Lecture Notes in Computer Science*, pp. 318–333, Springer Berlin Heidelberg, 2010.

[149] B. Vitoriano, M. Ortuño, G. Tirado, and J. Montero, "A multi-criteria optimization model for humanitarian aid distribution," *Journal of Global Optimization*, vol. 51, no. 2, pp. 189–208, 2011.

[150] N. Liu and Y. Ye, "Humanitarian logistics planning for natural disaster response with bayesian information updates," *Journal of Industrial and Management Optimization*, vol. 10, no. 3, pp. 665–689, 2014.

[151] M. Besiou, O. Stapleton, and L. N. Van Wassenhove, "System dynamics for humanitarian operations," *Journal of Humanitarian Logistics and Supply Chain Management*, vol. 1, no. 1, pp. 78–103, 2011.

[152] S. Yan and Y.-L. Shih, "An ant colony system-based hybrid algorithm for an emergency roadway repair time-space network flow problem," *Transportmetrica*, vol. 8, no. 5, pp. 361–386, 2012.

[153] Y.-J. Zheng, H.-F. Ling, J.-Y. Xue, and S.-Y. Chen, "Population classification in fire evacuation: A multiobjective particle swarm optimization approach," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 1, pp. 70–81, 2014.

[154] K. E. Parsopoulos, I. Konstantaras, and K. Skouri, "Metaheuristic optimization for the single-item dynamic lot sizing problem with returns and remanufacturing," *Computers & Industrial Engineering*, vol. 83, pp. 307–315, 2015.

[155] C. Chatfield, *Time-series forecasting*. CRC Press, 2000.

[156] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[157] E. S. Gardner, "Exponential smoothing: The state of the art," *Journal of forecasting*, vol. 4, no. 1, pp. 1–28, 1985.

[158] E. S. Gardner, "Exponential smoothing: The state of the art–part II," *International journal of forecasting*, vol. 22, no. 4, pp. 637–666, 2006.

[159] S. Makridakis, S. C. Wheelwright, and R. J. Hyndman, *Forecasting methods and applications*. John Wiley & Sons, 2008.

[160] R. J. Hyndman, *Moving Averages*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[161] S. W. Roberts, "Control chart tests based on geometric moving averages," *Technometrics*, vol. 1, no. 3, pp. 239–250, 1959.

[162] J. M. Lucas and M. S. Saccucci, "Exponentially weighted moving average control schemes: Properties and enhancements," *Technometrics*, vol. 32, no. 1, pp. 1–12, 1990.

[163] C. Grosan and A. Abraham, "A new approach for solving nonlinear equations systems," *IEEE Transactions on Systems, Man, and Cybernetics–Part A: Systems and Humans*, vol. 38, no. 3, pp. 698–714, 2008.

Appendix A

Appendix

A.1 Test Problems

A.1.1 Standard Test Suite

The standard test suite consists of the following problems:

Test Problem 0 (TP0 - Sphere) [18]. This is a separable n-dimensional problem, defined as:

$$f(x) = \sum_{i=1}^{n} x_i^2, \tag{A.1}$$

and it has a single global minimizer, $x^* = (0, 0, \ldots, 0)^\top$, with $f(x^*) = 0$.

Test Problem 1 (TP1 - Generalized Rosenbrock) [18]. This is a non-separable n-dimensional problem, defined as:

$$f(x) = \sum_{i=1}^{n-1} \left(100 \left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right), \tag{A.2}$$

and it has a global minimizer, $x^* = (1, 1, \ldots, 1)^\top$, with $f(x^*) = 0$.

Test Problem 2 (TP2 - Rastrigin) [18]. This is a separable n-dimensional problem, defined as:

$$f(x) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right), \tag{A.3}$$

and it has a global minimizer, $x^* = (0, 0, \ldots, 0)^\top$, with $f(x^*) = 0$.

Test Problem 3 (TP3 - Griewank) [18]. This is a non-separable n-dimensional problem, defined as:

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \tag{A.4}$$

and it has a global minimizer, $x^* = (0, 0, \ldots, 0)^\top$, with $f(x^*) = 0$.

Test Problem 4 (TP4 - Ackley) [18]. This is a non-separable n-dimensional problem, defined as:

$$\begin{aligned} f(x) &= 20 + \exp(1) - 20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) \\ &\quad - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right), \end{aligned} \tag{A.5}$$

and it has a global minimizer, $x^* = (0, 0, \ldots, 0)^\top$, with $f(x^*) = 0$.

A.1.2 Nonlinear Systems

This test set consists of six real-application problems, which are modeled as systems of nonlinear equations. Computing a solution of a nonlinear system is a very challenging task and it has received the ongoing attention of the scientific community. A common methodology for solving such systems is their transformation to an equivalent global optimization problem, which allows the use of a wide range of optimization tools. The transformation produces a single objective function by aggregating all the system's equations, such that the solutions of the original system are exactly the same with that of the derived optimization problem.

Consider the system of nonlinear equations:

$$\begin{cases} f_1(x) &= 0, \\ f_2(x) &= 0, \\ &\vdots \\ f_m(x) &= 0, \end{cases}$$

with $x \in S \subset \mathbb{R}^n$. Then, the objective function:

$$f(x) = \sum_{i=1}^{m} |f_i(x)|, \tag{A.6}$$

defines an equivalent optimization problem. Obviously, if x^* with $f(x^*) = 0$ is a global minimizer of the objective function, then x^* is also a solution of the corresponding nonlinear system and vice versa.

In our experiments, we considered the following nonlinear systems, previously employed by Grosan and Abraham [163] to justify the usefulness of evolutionary approaches as efficient solvers of nonlinear systems:

Test Problem 5 (TP5 - Interval Arithmetic Benchmark) [163] This problem consists of the following system:

$$\begin{cases} x_1 - 0.25428722 - 0.18324757\ x_4x_3x_9 &= 0, \\ x_2 - 0.37842197 - 0.16275449\ x_1x_{10}x_6 &= 0, \\ x_3 - 0.27162577 - 0.16955071\ x_1x_2x_{10} &= 0, \\ x_4 - 0.19807914 - 0.15585316\ x_7x_1x_6 &= 0, \\ x_5 - 0.44166728 - 0.19950920\ x_7x_6x_3 &= 0, \\ x_6 - 0.14654113 - 0.18922793\ x_8x_5x_{10} &= 0, \\ x_7 - 0.42937161 - 0.21180486\ x_2x_5x_8 &= 0, \\ x_8 - 0.07056438 - 0.17081208\ x_1x_7x_6 &= 0, \\ x_9 - 0.34504906 - 0.19612740\ x_{10}x_6x_8 &= 0, \\ x_{10} - 0.42651102 - 0.21466544\ x_4x_8x_1 &= 0. \end{cases} \tag{A.7}$$

The resulting objective function defined by Eq. (A.6), is 10-dimensional with global minimum $f(x^*) = 0$.

Test Problem 6 (TP6 - Neurophysiology Application) [163] This problem consists of the following system:

$$\begin{cases} x_1^2 + x_3^2 &= 1, \\ x_2^2 + x_4^2 &= 1, \\ x_5x_3^3 + x_6x_4^3 &= c_1, \\ x_5x_1^3 + x_6x_2^3 &= c_2, \\ x_5x_1x_3^2 + x_6x_4^2x_2 &= c_3, \\ x_5x_1^2x_3 + x_6x_2^2x_4 &= c_4, \end{cases} \tag{A.8}$$

where the constants, $c_i = 0$, $i = 1, 2, 3, 4$. The resulting objective function is 6-dimensional with global minimum $f(x^*) = 0$.

Test Problem 7 (TP7 - Chemical Equilibrium Application) [163] This problem consists

of the following system:

$$\begin{cases} x_1x_2 + x_1 - 3x_5 = 0, \\ 2x_1x_2 + x_1 + x_2x_3^2 + R_8x_2 - Rx_5 + 2R_{10}x_2^2 + R_7x_2x_3 + \\ R9x_2x_4 = 0, \\ 2x_2x_3^2 + 2R_5x_3^2 - 8x_5 + R_6x_3 + R_7x_2x_3 = 0, \\ R_9x_2x_4 + 2x_4^2 - 4Rx_5 = 0, \\ x_1(x_2 + 1) + R_{10}x_2^2 + x_2x_3^2 + R_8x_2 + R_5x_3^2 + x_4^2 - 1 \\ +R_6x_3 + R_7x_2x_3 + R_9x_2x_4 = 0, \end{cases} \tag{A.9}$$

where,

$$R = 10, \quad R_5 = 0.193, \quad R_6 = \frac{0.002597}{\sqrt{40}}, \quad R_7 = \frac{0.003448}{\sqrt{40}},$$
$$R_8 = \frac{0.00001799}{40}, \quad R_9 = \frac{0.0002155}{\sqrt{40}}, \quad R_{10} = \frac{0.00003846}{40}.$$

The corresponding objective function is 5-dimensional with global minimum $f(x^*) = 0$.

Test Problem 8 (TP8 - Kinematic Application) [163] This problem consists of the following system:

$$\begin{cases} x_i^2 + x_{i+1}^2 - 1 = 0, \\ a_{1i}x_1x_3 + a_{2i}x_1x_4 + a_{3i}x_2x_3 + a_{4i}x_2x_4 + a_{5i}x_2x_7 + \\ a_{6i}x_5x_8 + a_{7i}x_6x_7 + a_{8i}x_6x_8 + a_{9i}x_1 + a_{10i}x_2 + a_{11i}x_3 + \\ a_{12i}x_4 + a_{13i}x_5 + a_{14i}x_6 + a_{15i}x_7 + a_{16i}x_8 + a_{17i} = 0, \end{cases} \tag{A.10}$$

with a_{ki}, $1 \leqslant k \leqslant 17$, $1 \leqslant i \leqslant 4$, is the corresponding element of the k-th row and i-th column of the matrix:

$$A = \begin{bmatrix} -0.249150680 & 0.125016350 & -0.635550077 & 1.48947730 \\ 1.609135400 & -0.686607360 & -0.115719920 & 0.23062341 \\ 0.279423430 & -0.119228120 & -0.666404480 & 1.32810730 \\ 1.434801600 & -0.719940470 & 0.110362110 & -0.25864503 \\ 0.000000000 & -0.432419270 & 0.290702030 & 1.16517200 \\ 0.400263840 & 0.000000000 & 1.258776700 & -0.26908494 \\ -0.800527680 & 0.000000000 & -0.629388360 & 0.53816987 \\ 0.000000000 & -0.864838550 & 0.581404060 & 0.58258598 \\ 0.074052388 & -0.037157270 & 0.195946620 & -0.20816985 \\ -0.083050031 & 0.035436896 & -1.228034200 & 2.68683200 \\ -0.386159610 & 0.085383482 & 0.000000000 & -0.69910317 \\ -0.755266030 & 0.000000000 & -0.079034221 & 0.35744413 \\ 0.504201680 & -0.039251967 & 0.026387877 & 1.24991170 \\ -1.091628700 & 0.000000000 & -0.057131430 & 1.46773600 \\ 0.000000000 & -0.432419270 & -1.162808100 & 1.16517200 \\ 0.049207290 & 0.000000000 & 1.258776700 & 1.07633970 \\ 0.049207290 & 0.013873010 & 2.162575000 & -0.69686809 \end{bmatrix}.$$

The corresponding objective function is 8-dimensional with global minimum $f(x^*) = 0$.

Test Problem 9 (TP9 - Combustion Application) [163] This problem consists of the following system:

$$\begin{cases} x_2 + 2x_6 + x_9 + 2x_{10} &= 10^{-5}, \\ x_3 + x_8 &= 3 \times 10^{-5}, \\ x_1 + x_3 + 2x_5 + 2x_8 + x_9 + x_{10} &= 5 \times 10^{-5}, \\ x_4 + 2x_7 &= 10^{-5}, \\ 0.5140437 \times 10^{-7} x_5 &= x_1^2, \\ 0.1006932 \times 10^{-6} x_6 &= 2x_2^2, \\ 0.7816278 \times 10^{-15} x_7 &= x_4^2, \\ 0.1496236 \times 10^{-6} x_8 &= x_1 x_3, \\ 0.6194411 \times 10^{-7} x_9 &= x_1 x_2, \\ 0.2089296 \times 10^{-14} x_{10} &= x_1 x_2^2. \end{cases} \tag{A.11}$$

The corresponding objective function is 10-dimensional with global minimum $f(x^*) = 0$.

Test Problem 10 (TP10 - Economics Modeling Application) [163] This problem consists of the following system:

$$\begin{cases} \left(x_k + \sum_{i=1}^{n-k-1} x_i x_{i+k} \right) x_n - c_k &= 0, \\ \sum_{l=1}^{n-1} x_l + 1 &= 0, \end{cases} \tag{A.12}$$

where $1 \leqslant k \leqslant n-1$, and $c_i = 0$, $i = 1, 2, \ldots, n$. The problem was considered in its 20-dimensional instance. Thus, the corresponding objective function was also 20-dimensional, with global minimum $f(x^*) = 0$.

Author's Publications

Journals

- D. Souravlias, K.E. Parsopoulos and G.C. Meletiou. "*Designing Bijective S-boxes Using Algorithm Portfolios with Limited Time Budgets*", Journal of Applied Soft Computing, 2017, Elsevier, accepted for publication.
- D. Souravlias, K.E. Parsopoulos and I.S. Kotsireas. "*Circulant Weighing Matrices: A Demanding Challenge for Parallel Optimization Metaheuristics*", Optimization Letters, 10 (6), pp. 1303-1314, 2016, Springer.
- D. Souravlias and K.E. Parsopoulos. "*Particle Swarm Optimization with Neighborhood-Based Budget Allocation*", International Journal of Machine Learning and Cybernetics (IJMLC), 7 (3), pp. 451-477, 2016, Springer.

Edited Volumes

- I.S. Kotsireas, P.M. Pardalos, K.E. Parsopoulos and D. Souravlias. "*On the Solution of Circulant Weighing Matrices Problems Using Algorithm Portfolios on Multi-Core Processors*", Lecture Notes in Computer Science (LNCS), Vol. 9685, pp. 184-200, 2016, Springer.
- T. Korkou, D. Souravlias, K.E. Parsopoulos and K. Skouri. "*Metaheuristic Optimization for Logistics in Natural Disasters*", Springer Proceedings in Mathematics & Statistics, Vol. 185 (Dynamics of Disasters - Key Concepts, Models, Algorithms, and Insights), pp. 113-134, 2016, Springer.
- D. Souravlias, K.E. Parsopoulos and E. Alba. "*Parallel Algorithm Portfolio with Market Trading-based Time Allocation*", Operations Research Proceedings 2014, by M. Lubbecke et al. (eds.), pp. 567-574, 2014, Springer.

Conferences/Workshops

- D. Souravlias, G. Luque, E. Alba and K.E. Parsopoulos. *"Smart Traffic Lights: A First Parallel Computing Approach"*, in Proc. of the 8th International Conference on Intelligent Networking and Collaborative Systems (INCoS 2016), September 07-09, 2016, Ostrava, Czech Republic.

- D. Souravlias and K.E. Parsopoulos. *"Particle Swarm Optimization with Budget Allocation through Neighborhood Ranking"*, in Proc. of the 22nd Genetic and Evolutionary Computation Conference, (GECCO 2013), July 06-10, 2013, Amsterdam, Netherlands.

- G. Koloniari, D. Souravlias and E. Pitoura. *"On Graph Deltas for Historical Queries"*, in Proc. of the 1st Workshop on Online Social Systems (WOSS 2012), in conjunction with the VLDB 2012 Conference, August 31, 2012, Istanbul, Turkey.

- D. Souravlias, G. Koloniari and E. Pitoura. *"InterSocialDB: An Infrastructure for Managing Social Data"*, in Proc. of the 1st International Workshop on Online Social Networks (IWOSN 2012), June 15, 2012, Patras, Greece.

- G. Koloniari, N. Ntarmos, E. Pitoura and D. Souravlias. *"One Is Enough: Distributed Filtering for Duplicate Elimination"*, in Proc. of the 20th ACM Conference on Information and Knowledge Management (CIKM 2011), October 24-28, 2011, Glasgow, Scotland, UK.

- D. Souravlias, M. Drosou, K. Stefanidis and E. Pitoura. *"On Novelty in Publish/Subscribe Delivery"*, in Proc. of the 4th International Workshop on Ranking in Databases (DBRank 2010), in conjunction with the ICDE 2010 Conference, March 1, 2010, Long Beach, California, USA.

- D. Souravlias. *"Data Stream Summarization to Avoid Overlap"*, in Proc. of the 3rd Panhellenic Scientific Student Conference on Informatics, Related Technologies and Applications (EUREKA 2009), September 10-12, 2009, Corfu, Greece.

Under review

- D. Souravlias, I.S. Kotsireas, P.M. Pardalos and K.E. Parsopoulos. *"Parallel Algorithm Portfolios with Performance Forecasting for the Detection of Circulant Weighing Matrices"*.

Short Biography

Dimitrios Souravlias was born in Ioannina in 1987. He received his BSc and MSc Degrees from the Department of Computer Science and Engineering of the University of Ioannina, Greece in 2009 and 2011, respectively. In 2013 he was admitted to the PhD program of the same institution. His research focus is on parallel computational optimization algorithms, especially algorithm portfolios and their applications on combinatorics, operations research, and smart cities.