HELLENIC REPUBLIC
National and Kapodistrian University of Athens
EST. 1837

Intelligent Online Optimization Algorithms for Portfolio Analysis and Management

BY
Spyridon D. Mourtas

Dissertation presented in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy in Finance
Department of Economics
National and Kapodistrian University of Athens

Athens, Greece
May 2023

Three-member Advisory Committee:

Supervisor: Vasilios N. Katsikis, Associate Professor, NKUA, Department of Economics

A. Member: Predrag S. Stanimirović, Professor, University of Nis, Department of Computer Science

B. Member: Charalampos Tsitouras, Professor, NKUA, General Department

Seven-member Examination Committee:

Vasilios N. Katsikis, Associate Professor, NKUA, Department of Economics

Predrag S. Stanimirović, Professor, University of Nis, Department of Computer Science

Charalampos Tsitouras, Professor, NKUA, General Department

Yiannis C. Bassiakos, Professor, NKUA, Department of Economics

Dimitra Kyriakopoulou, Assistant Professor, NKUA, Department of Economics

Ioannis Th. Famelis, Professor, University of West Attica, Department of Electrical and Electronics Engineering

Zacharoula Kalogiratou, Professor, University of Western Macedonia, Department of Mathematics

Acknowledgments

By completing my doctoral dissertation, I feel obliged to thank the people who are contributed to its completion. Firstly, I want to thank my supervisor professor, Vasilios N. Katsikis, Associate Professor of the Department of Economics at the National and Kapodistrian University of Athens for the impeccable cooperation. Throughout the course of my work, his essential guidance and suggestions in dealing with the difficulties I faced were catalytic. His valuable advices, the trust he showed me and especially his patience helped me greatly to complete my work.

I would like to thank my committee members, professor Predrag S. Stanimirović and professor Charalampos Tsitouras, whose expertise, guidance and encouragement have been beyond valuable for both the work of this doctoral dissertation and my academic development.

I gratefully acknowledge the funding received towards my PhD from the National and Kapodistrian University of Athens through the PhD scholarship "Stavros Tsakirakis".

I would like to thank the Erasmus+ Study Abroad program, 2021-22. In October 2021, I went to the University of Nis for an academic semester for the elaboration of research with professor Predrag S. Stanimirović. My time at University of Nis has been highly productive and working with professor Predrag S. Stanimirović was an extraordinary experience. Participating in the Erasmus+ Study Abroad program, 2021-22, broadened my research background and provided new innovations and findings to strengthen my doctoral dissertation.

And lastly, a big thanks belong to my parents, Dimitrios and Kalliopi, as well as my brother, Panagiotis, and my sister, Ourania, where their faith in my possibilities served as aid on my purposes and dreams.

List of Publications

[1] V.N. Katsikis, S.D. Mourtas, P.S. Stanimirović, S. Li, X. Cao, "Time-varying minimum-cost portfolio insurance problem via an adaptive fuzzy-power LVI-PDNN", *Applied Mathematics and Computation*, 441, 127700 (2023)

[2] V.N. Kovalnogov, R.V. Fedorov, D.A. Generalov, A.V. Chukalin, V.N. Katsikis, S.D. Mourtas, T.E. Simos, "Portfolio Insurance through Error-Correction Neural Networks", *Mathematics*, 10(18), 3335 (2022)

[3] S.D. Mourtas, V.N. Katsikis, "Exploiting the Black-Litterman framework through error-correction neural networks", *Neurocomputing*, 498, 43-58 (2022)

[4] S.D. Mourtas, C. Kasimis, "Exploiting Mean-Variance Portfolio Optimization Problems through Zeroing Neural Networks", *Mathematics*, 10(17), 3079 (2022)

[5] S.D. Mourtas, V.N. Katsikis, "V-Shaped BAS: Applications on Large Portfolios Selection Problem", *Computational Economics*, 60, 1353–1373 (2022)

[6] V.N. Katsikis, S.D. Mourtas, "Diversification of Time-Varying Tangency Portfolio under Nonlinear Constraints through Semi-Integer Beetle Antennae Search Algorithm", *AppliedMath*, 1(1), 63-73 (2021)

[7] V.N. Katsikis, S.D. Mourtas, P.S. Stanimirović, S. Li, X. Cao, "Time-varying mean-variance portfolio selection problem solving via LVI-PDNN", *Computers & Operations Research*, 105582 (2021)

[8] V.N. Katsikis, S.D. Mourtas, "Portfolio Insurance and Intelligent Algorithms". In: S. Patnaik, K. Tajeddini, V. Jain (eds) *Computational Management.* Modeling and Optimization in Science and Technologies, vol 18. Springer, Cham., 305-323 (2021)

[9] V.N. Katsikis, S.D. Mourtas, P.S. Stanimirović, S. Li, X. Cao, "Time-Varying Mean-Variance Portfolio Selection under Transaction Costs and Cardinality Constraint Problem via Beetle Antennae Search Algorithm (BAS)", *SN Operations Research Forum*, 2(2), 18 (2021)

[10] V.N. Katsikis, S.D. Mourtas, P.S. Stanimirović, S. Li, X. Cao, "Time-varying minimum-cost portfolio insurance under transaction costs problem via Beetle Antennae Search Algorithm (BAS)", *Applied Mathematics and Computation*, 385, 125453 (2020)

Intelligent Online Optimization Algorithms for Portfolio Analysis and Management

Extended Abstract

Optimization models play a significant role in financial decisions. Popular fields of them are portfolio insurance, portfolio selection, asset allocation, risk management, option pricing, model calibration, etc. and can be solved efficiently using modern optimization techniques. In the literature, due to the high complexity of the mentioned optimization problems, the time-varying (TV) attempts to solve financial problems restrict in solving the corresponding static problems sequentially. Our approach is to define and solve popular financial models in real-time through intelligent online optimization algorithms.

The financial models investigated in this dissertation are portfolio selection problems, and they can be classified into two categories. The one category includes TV linear programming (LP) problems and TV quadratic programming (QP) problems, while the other category includes TV nonlinear programming (NLP) problems and TV integer linear programming (ILP) problems. More precisely, the TV minimum-cost portfolio insurance (MCPI) problem is defined and studied as a TV LP problem, while the TV mean-variance portfolio selection (MVPS) problem and the TV Black-Litterman portfolio optimization (BLPO) problem are defined and studied as TV QP problems. In addition, by adding nonlinear constraints, the TV versions of the MCPI and MVPS problems, the multi-period version of the MCPI, and a TV tangency portfolio (TP) problem are defined and studied as a TV NLP problems, while the TV version of MVPS problem is also defined and studied as a TV ILP problem. Note that the nonlinear constraints refer to transaction costs and cardinality constraints.

Intelligent online optimization algorithms, which include modern neural network (NN) techniques and state-of-the-art metaheuristics, are employed to solve the aforementioned TV financial optimization problems using real-world datasets. More particularly, the TV LP/QP financial optimization problems are approached by two continuous-time NN solvers. These solvers are the zeroing NN (ZNN) and the linear-variational-inequality primal-dual NN (LVI-PDNN). Moreover, the TV NLP/ILP financial optimization problems are approached by modern metaheuristic optimization algorithms. These metaheuristic algorithms are variations of a popular nature inspired algorithm called Beetle Antennae Search (BAS), whose primary advantage is its low time consumption.

The main outcomes of numerical applications in real-world datasets and in various portfolio configurations show that our methods are excellent substitutes for other popular methods. Consequently, our approach will provide a new perspective on TV financial models and will demonstrate an online model solution.

Intelligent Online Optimization Algorithms for Portfolio Analysis and Management

Εκτεταμένη Περίληψη

Τα μοντέλα βελτιστοποίησης παίζουν σημαντικό ρόλο στις χρηματοοικονομικές αποφάσεις. Δημοφιλή πεδία τους είναι η ασφάλιση χαρτοφυλακίου, η επιλογή χαρτοφυλακίου, η κατανομή περιουσιακών στοιχείων, η διαχείριση κινδύνων, η τιμολόγηση συμβολαίων δικαιωμάτων προαίρεσης, η βαθμονόμηση μοντέλου κ.λπ. και μπορούν να επιλυθούν αποτελεσματικά χρησιμοποιώντας σύγχρονες τεχνικές βελτιστοποίησης. Στη βιβλιογραφία, λόγω της μεγάλης πολυπλοκότητας των αναφερθέντων προβλημάτων, οι χρονικά μεταβαλλόμενες (TV) προσπάθειες επίλυσης οικονομικών προβλημάτων περιορίζονται στην επίλυση των αντίστοιχων στατικών προβλημάτων τους ακολουθιακά. Η προσέγγισή μας είναι να καθορίσουμε και να λύσουμε δημοφιλή χρηματοοικονομικά μοντέλα σε πραγματικό χρόνο μέσω ευφυών online αλγορίθμων βελτιστοποίησης.

Τα χρηματοοικονομικά μοντέλα που εξετάζονται σε αυτή τη διατριβή είναι προβλήματα επιλογής χαρτοφυλακίου και μπορούν να ταξινομηθούν σε δύο κατηγορίες. Η μία κατηγορία περιλαμβάνει προβλήματα TV γραμμικού προγραμματισμού (LP) και προβλήματα TV τετραγωνικού προγραμματισμού (QP), ενώ η άλλη κατηγορία περιλαμβάνει προβλήματα TV μη γραμμικού προγραμματισμού (NLP) και προβλήματα TV ακέραιου γραμμικού προγραμματισμού (ILP). Πιο συγκεκριμένα, το TV πρόβλημα ασφάλισης χαρτοφυλακίου ελάχιστου κόστους (MCPI) ορίζεται και μελετάται ως πρόβλημα TV LP, ενώ το TV πρόβλημα επιλογής χαρτοφυλακίου μέσου-διακύμανσης (MVPS) και το TV πρόβλημα βελτιστοποίησης χαρτοφυλακίου των Black-Litterman (BLPO) ορίζονται και μελετώνται ως προβλήματα TV QP. Επιπλέον, με την προσθήκη μη γραμμικών περιορισμών, οι TV εκδοχές των προβλημάτων MCPI και MVPS, η εκδοχή πολλαπλών περιόδων του MCPI, και η TV εκδοχή του προβλήματος εφαπτομενικού χαρτοφυλακίου (TP) ορίζονται και μελετώνται ως προβλήματα TV NLP, ενώ η TV εκδοχή του προβλήματος MVPS ορίζεται και μελετάται επίσης ως πρόβλημα TV ILP. Σημειώστε ότι οι μη γραμμικοί περιορισμοί αναφέρονται στο κόστος συναλλαγών και στους cardinality περιορισμούς.

Ευφυείς online αλγόριθμοι βελτιστοποίησης, οι οποίοι περιλαμβάνουν τεχνικές σύγχρονων νευρωνικών δικτύων (NN) και μεταευρετικές μεθόδους τελευταίας τεχνολογίας, χρησιμοποιούνται για την επίλυση των προαναφερθέντων TV προβλημάτων χρηματοοικονομικής βελτιστοποίησης χρησιμοποιώντας πραγματικά σύνολα δεδομένων. Πιο συγκεκριμένα, τα προβλήματα χρηματοοικονομικής βελτιστοποίησης TV LP/QP προσεγγίζονται από δύο νευρωνικά δίκτυα συνεχούς χρόνου. Τα νευρωνικά δίκτυα που έχουμε επιλέξει είναι το zeroing NN (ZNN) και το linear-variational-inequality primal-dual NN (LVI-PDNN). Επιπλέον, τα προβλήματα χρηματοοικονομικής βελτιστοποίησης TV NLP/ILP προσεγγίζονται από δημοφιλείς μεταευρετικούς αλγόριθμους βελτιστοποίησης. Αυτοί οι μεταευρετικοί αλγόριθμοι είναι παραλλαγές ενός αλγορίθμου εμπνευσμένου από τη φύση που ονομάζεται Beetle Antennae Search (BAS), του οποίου το κύριο πλεονέκτημα είναι το χαμηλό υπολογιστικό κόστος.

Τα κύρια αποτελέσματα αριθμητικών εφαρμογών σε πραγματικά σύνολα δεδομένων και σε διαφορετικές διαμορφώσεις χαρτοφυλακίου καταδεικνύουν ότι οι μέθοδοι που προτείνουμε είναι ανταγωνιστικές σε σύγκριση με άλλες δημοφιλείς μεθόδους. Κατά συνέπεια, η προσέγγισή μας θα προσφέρει μια νέα προοπτική στα TV χρηματοοικονομικά μοντέλα και θα προτείνει μια online λύση μοντέλου.

Contents

List of Figures

List of Tables

1 Introduction

The objectives and challenges of this doctoral dissertation are presented in this chapter, as well as the state-of-the-art and innovation, along with the scientific and social impact.

1.1 Objectives and Challenges

Portfolio management is the science and art of choosing and administering a portfolio of assets that aligns with an investor's long-term financial goals and risk tolerance. Portfolio optimization is thus an important aspect of portfolio management. Risk management [1, 2], insurance costs [3], transaction costs [4], option replication [5], and other disciplines of portfolio optimization can be efficiently addressed utilizing traditional optimization techniques. Nature inspired algorithms [4], conic programming [6], branch and bound technique [7], non-differential optimization and cutting planes techniques [8], Riesz-space theory [3, 5] are some of these techniques.

In the literature, due to the high complexity of the mentioned problems, the time-varying (TV) attempts to solve financial problems restrict in solving the corresponding static problems "several" times. Our approach is to define and solve popular (such as the Markowitz Nobel Prize-winning model), financial models in real-time. This will provide a new perspective on TV financial models and will demonstrate an online model solution. It is worth mentioning that an online solution consists of consecutive solutions with the feature that the previous solution is used as an initial input instead of a random input at each solution's iterative process.

The major challenges in this direction that this doctoral dissertation intends to address are:
A) The proposed TV financial problems need to be TV analogues of the corresponding static problems and at the same time to satisfy the conditions of a multiperiod continuous or discrete-time model. Also, the models must be able to handle real-world datasets and to be applied in real-world situations. So, when an online solution is

produced, it will be as real as possible.
B) The methods which produce the online solution need to be contemporary. Consequently, they should be developed even further to handle the time-series data efficiently and at the same time to be highly competitive or even better alternatives than conventional ones.
C) The technical tools that we will use to implement our ideas are modern and their speed and effectiveness compared to conventional methods have already been proven for specific non-financial data sets. However, these methods need to be further developed to handle time-series financial data with the same efficiency and to provide new financial methods that are attractive to the interested user.

1.2 State-of-the-art and Innovation

This doctoral dissertation proposes and investigates new concepts of real-time financial optimization problems, while using intelligent online optimization algorithms to solve them. These intelligent online optimization algorithms include modern neural network (NN) techniques and state-of-the-art metaheuristics. The financial models investigated in this dissertation are portfolio selection problems, and they can be classified into two categories. The one category includes TV linear programming (LP) problems and TV quadratic programming (QP) problems, while the other category includes TV nonlinear programming (NLP) problems and TV integer linear programming (ILP) problems.

More particularly, the TV LP/QP financial optimization problems are approached by two continuous-time NN solvers. These solvers are the zeroing NN (ZNN) and the linear-variational-inequality primal-dual NN (LVI-PDNN). The TV NLP/ILP financial optimization problems are approached by popular metaheuristic optimization algorithms. These metaheuristic algorithms are variations of a nature inspired algorithm called Beetle Antennae Search (BAS), whose primary advantage is its low time consumption. In this way, this doctoral dissertation gives strong evidence that our approach is original, scientifically important and promising for real-life applications. The following diagram, in Fig. 1.1, presents the computational methods classification as it is used in this thesis.

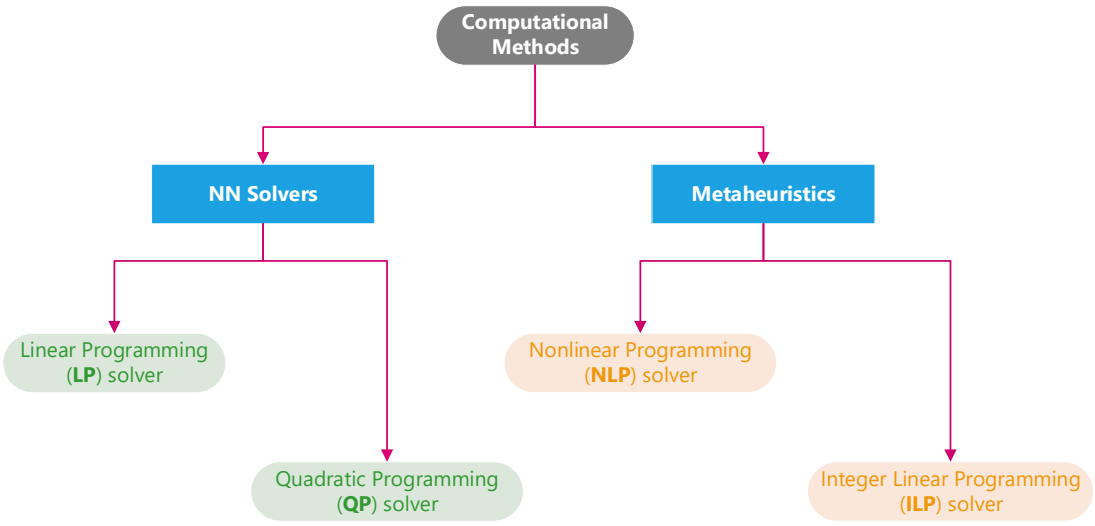


Figure 1.1: Computational methods classification.

Summing up the previous analysis, we can say:

A) Our TV versions of the proposed financial problems are novel approaches of the corresponding static financial problems. That is, we are dealing with multiperiod continuous and discrete-time models in the field of finance which are using real-world data in real-world situations.

B) The online solution is a technique that reduces the time consumption needed to generate a regular solution to a time-varying problem. The online solution has never been applied before in the field of finance.

1.3 Scientific and Social Impact

TV versions of popular static financial optimization problems that are not currently available in the literature are introduced and analyzed, and special types of NNs (ZNN and LVI-PDNN) and metaheuristic algorithms (BAS variations) are shown to be an excellent alternative to conventional optimization methods in providing an optimal solution to these TV financial optimization problems. To the best of our knowledge, this is a novel approach that comprises robust techniques from NNs and metaheuristics to provide an online, thus more realistic, solution to the TV financial problems. In this way, the static method limitations are eliminated. The following diagram, in Fig. 1.2, illustrates precisely the scientific directions that we plan to follow and connects the financial problems with the mathematical tools that we intend to use to achieve our goal.

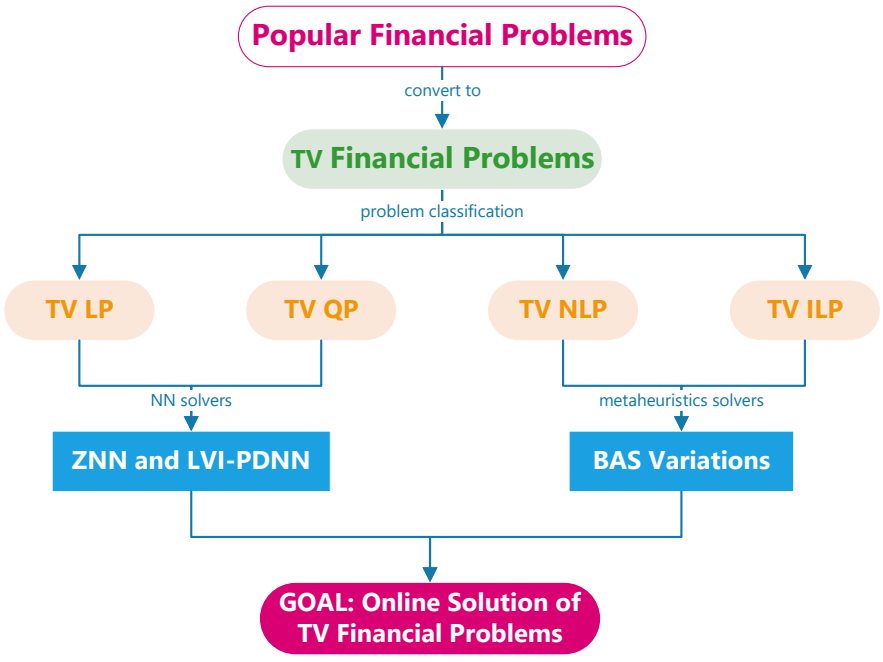


Figure 1.2: Research methodology flowchart.

Summarizing the benefits we may say that:

A) The online solution of a TV financial problem is a great technical analysis tool as well as an important financial analysis tool.

B) New methodologies and algorithmic procedures are introduced in the field of finance.

C) The online solution of the TV financial problems along with fundamental analysis will enable the investors to make better decisions. That is, it will enhance the process of evaluating finance-related transactions to determine their performance and suitability.

This thesis is organized in such a way that the above contributions are reflected, and at a chapter level, it is structured in the following way:

Chapter 2: This chapter provides an overview of nature-inspired optimization algorithms as well as brief descriptions of some of the most popular metaheuristic algorithms.

Chapter 3: Preliminaries on NN are covered in this chapter, as well as detailed descriptions of the NN solvers.

Chapter 4: In this chapter, the TV versions of popular static portfolio management financial problems are presented and classified as TV LP/QP/NLP/ILP problems, after which they are approached using NN solvers and metaheuristic algorithms.

Chapter 5: Numerical applications in real-world datasets and in different portfolio configurations, as well as their analysis are presented in this chapter. It is worth mentioning that the complete development and implementation of the computational methods can be obtained from GitHub, with links to their repositories included in this chapter.

Chapter 6: Concluding comments and observations along with study limitations and pointers for future work are given in this chapter.

2

An Introduction to Nature-Inspired Optimization Algorithms

Applications in engineering, business operations, and industrial design are all dominated by optimization. The goals of optimization can obviously range from minimizing energy consumption and costs to maximizing profit, output, performance, and efficiency. In real-world applications, resources, time, and money are constantly limited, therefore we must develop ways to make the best use of these valuable resources under a variety of limitations. The study of such planning and design problems using mathematical methods is known as mathematical optimization or programming. Because most real-world applications are highly nonlinear, they necessitate the use of advanced optimization methods. Computer simulations have now become an essential tool for solving such optimization problems using a variety of efficient algorithms.

Algorithms are always at work behind any computer simulation or computational process. How an algorithm operates, as well as its efficiency and performance, is determined by their core components and how they interact. This chapter introduces algorithms and delves into their inner workings. Then we go over how to formulate an optimization problem in general, as well as current methodologies like swarm intelligence (SI) and bio-inspired computation.

2.1 Introduction to Optimization

In this section, an introduction to optimization and some fundamental approaches for dealing with constraints in constrained optimization problems are presented.

2.1.1 Optimization

In terms of mathematics, most optimization problems can be written in the following generic form:

$$\min_{\mathbf{x}\in\mathbb{R}^d} \quad f_i(\mathbf{x}), \quad (i = 1, 2, \ldots, M) \tag{2.1}$$

$$\text{subject to} \quad h_j(\mathbf{x}) = 0, \quad (j = 1, 2, \ldots, L) \tag{2.2}$$

$$g_k(\mathbf{x}) \le 0, \quad (k = 1, 2, \ldots, K) \tag{2.3}$$

where $f_i(\mathbf{x}), h_j(\mathbf{x}), g_k(\mathbf{x})$ are functions of the design vector $\mathbf{x}$. Inhere, the entries x_i of $\mathbf{x}$ are called design or decision variables, and they might be real continuous, discrete, or a combination of these two.

The functions $f_i(\mathbf{x})$ with $i = 1, 2, \ldots, M$ are called the objective functions or cost functions, and there is just one objective when $M = 1$. The design space or search space $\mathbb{R}^d$ is the space spanned by the decision variables, whereas the solution space or response space is the space produced by the objective function values. The equalities for h_j and inequalities for g_k are called constraints. It is worth noting that the inequalities can also be written in another way, ≥ 0, and the objectives can be expressed as a maximization problem. Furthermore, in a rare but extreme case where there is no objective at all, there are only constraints. Such a problem is known as a feasibility problem since any feasible solution is an optimal solution.

If we try to categorize optimization problems based on the number of objectives, we may divide them into two groups: single objective $M = 1$ and multi-objective $M > 1$. Multi-objective optimization is also known as multi-criteria or even multi-attribute optimization in the literature. The majority of optimization tasks in real-world problems are multi-objective. Although that the algorithms we cover in this chapter can be used to solve multi-objective optimization problems with minor adjustments, we focus on single-objective optimization problems.

Likewise, we can divide optimization into categories based on the amount of constraints $J + K$. An unconstrained optimization problem is one in which there is no constraint at all, $J = K = 0$. An equality-constrained problem is defined as $K = 0$ and $J = 1$, whereas an inequality-constrained problem is defined as $J = 0$ and $K = 1$.

It is worth noting that equalities aren't explicitly mentioned in some formulations in the optimization literature, only inequalities are. This is the case since an equality can be expressed as two inequalities. For instance, $h(\mathbf{x}) = 0$ is equivalent to $h(\mathbf{x}) \le 0$ and $h(\mathbf{x}) \ge 0$. Equality constraints, however, have unique aspects that necessitate extra caution. One disadvantage is that the volume of satisfying an equality in the search space is basically zero, making it extremely difficult to obtain sampling points that perfectly satisfy the equality. In practice, some tolerance or allowance is applied.

For classification, we may also employ the actual function forms. That is, linear or nonlinear objective functions are possible. It becomes a linearly constrained problem if all of the constraints h_j and g_k are linear, and it becomes a linear programming problem when all of the constraints and objective functions are linear. Inhere, "programming" does not refer to computer programming; rather, it refers to planning and/or optimization. Furthermore, if any of the constraints or objective functions are nonlinear, we are dealing with a nonlinear optimization problem.

2.1.2 Dealing with Constraints in Constrained Optimization Problems

Both inequality and equality constraints can be addressed in a variety of methods [9, 10]. A variety of constraint-handling approaches can be categorized in several ways. In general, we may split them into two

simple categories: classic methods and recent methods. Classic/traditional methods are still widely utilized in many applications, and new or recent advancements have generally been based on a combination of evolutionary principles and conventional methods. As a consequence, the distinctions here between old and new are arbitrary and only for the sake of argument. For example, traditional methods include penalty methods, separation of objectives and constraints, specific representations, and transformation methods, whereas recent methods include stochastic ranking, feasibility methods, the multi-objective approach, new special operator methods, and hybrid or ensemble methods. The following is a brief description of some well-known traditional methods.

Penalty methods: Penalty methods try to convert a constrained optimization problem into an unconstrained one by incorporating its constraints in the revised objective. In [9], penalty functions operate in a series of sequences, modifying a set of penalty parameters each time, and beginning a new sequence with the previous one. During construction of any sequence the following penalty function is maximized:

$$G(x, R) = f(x) - \Omega(R, g(x), h(x)), \tag{2.4}$$

where Ω is the penalty term and $f(x)$ is the objective function. Moreover, R is a set of penalty parameters, $g(x)$ is the inequality constraint function and $h(x)$ is the equality constraint function. Note that, for equality or inequality constraints, different penalty terms are used. In general, the key benefit of this approach is that it enables any convex or nonconvex constraints to be complied with. However, this introduces more parameters into the problem, but if proper values are used, the converted unconstrained problem can often be solved by many algorithms relatively effectively.

Separation of objective functions and constraints: Another method that has gained popularity in recent years is the separation of objective functions and constraints. For instance, the following fitness function $\rho(x)$ is proposed in [11]:

$$\rho(x) = \begin{cases} f(x), & \text{if feasible} \\ 1 + \mu \left[\sum_{j=1}^{L} h_j(x) + \sum_{k=1}^{K} g_k(x) \right], & \text{otherwise.} \end{cases} \tag{2.5}$$

where $g(x)$ is the inequality constraint function, $h(x)$ is the equality constraint function, and $\mu \geq 0$ is a parameter. A feasible solution should always have a higher fitness value than an infeasible solution, according to this concept.

Transformation methods: Transformation methods attempt to map the feasible region into a normal mapped space while maintaining feasibility in some way. For instance, the homomorphous map is introduced in [12], which attempts to convert the viable region into a higher-dimensional cube. Though the approach proved competitive, practical implementations were challenging due to its non-trivial transformations and high computing cost. Other techniques, on the other hand, generate viable solutions while maintaining their feasibility using specific representations and operators. However, such special operators techniques are more commonly used to solve linear constraints [12]. These transformation approaches are commonly referred to as decoder and special operator methods [10].

2.2 Introduction to Algorithmic Processes

An algorithm is essentially a step-by-step process for doing calculations or giving instructions, with many of them being iterative. The specific steps and methods are determined by the algorithm utilized and the context in

question. However, because we are primarily concerned with optimization algorithms in this chapter, we lay a greater emphasis on iterative processes for creating algorithms.

From a mathematical standpoint, an iterative process tries to generate a new and superior solution x_{t+1} to a given problem from the current solution x_t at time or iteration t. That is,

$$x_{t+1} = g(x_t), \tag{2.6}$$

where g is a mathematical function of x_t or a set of mathematical equations. Note that several textbooks utilize the upper index form x^{t+1}. Inhere, x^{t+1} does not imply x to the power of $t+1$. When utilized correctly, such notations will become beneficial and will cause no confusion. In this chapter, we make use of such notations where necessary.

2.2.1 Classification of Optimization Algorithms

After an optimization problem has been correctly formulated, the major aim is to discover the best solutions using a solution approach that employs the appropriate mathematical methods. The algorithms employed for solving optimization may be traditional algorithms (e.g. gradient-based methods, simplex methods and quadratic programming), evolutionary algorithms, heuristic or metaheuristic algorithms and several hybrid techniques.

Traditional algorithms are good for the problems they can address, however most of them are local search algorithms. The following are their major disadvantages:

• With the exception of linear programming and convex optimization, most optimization problems have no guarantee of global optimality because existing algorithms are largely local search. As a result, the final solution will frequently be determined by the initial starting positions.

• Because they frequently require some information about the local objective landscape, such as derivatives, traditional algorithms tend to be problem-specific. Other methods, such as k-opt and branch and bound, might be substantially influenced by the sort of implementation problems.

• Traditional algorithms struggle to address highly nonlinear, multimodal problems and have difficulty dealing with problems involving discontinuity, particularly when gradients are required.

• Except for hill-climbing with random restart, almost all traditional algorithms are deterministic. Starting with the same initial positions, the ending solutions would be the same. In addition, because no random numbers are employed, the diversity of the solutions obtained may be limited.

To address the aforementioned drawbacks, modern algorithms are mostly heuristic and metaheuristic. Heuristic algorithms generate new solutions through trial and error, whereas metaheuristic algorithms are higher-level heuristics that employ memory, solution history, and other learning strategies. Currently, the majority of metaheuristic algorithms are inspired by nature, and the majority of these algorithms are built on SI inspired by nature [13, 14]. Metaheuristics, in contrast to traditional algorithms, are primarily developed for global search and include the following benefits and characteristics:

• They are more likely to uncover the genuine global optimality since they are global optimizers.

• They frequently treat problems as a black box, requiring no specific knowledge, allowing them to tackle a broader range of problems.

• Metaheuristic algorithms are mainly gradient-free and do not use derivative information, allowing them to deal

with extremely nonlinear and discontinuous problems.
• Because stochastic components, such as random numbers and random walks, are frequently employed, such algorithms are stochastic. Even when starting with the same initial positions, no identical solutions can be obtained, but the resulting solutions can be sufficiently close to allow the algorithm to escape any local modes. As a result, it is less prone to become trapped in local regions.

Despite these benefits, nature-inspired algorithms have certain drawbacks. Generally, computational costs are higher than traditional algorithms since more iterations are required, which could become computationally expensive if a simulator must evaluate a single objective for a long time. Furthermore, because the final solutions generated by such algorithms cannot be replicated exactly, many runs should be performed to guarantee consistency and statistical analysis.

Traditional algorithms are primarily deterministic, with no use of randomization in the generation of new solutions. This can help with exploitation, but it cannot help with exploration. Nature-inspired metaheuristic algorithms, on the other hand, incorporate some unpredictability and include stochastic components. A higher amount of randomness will boost exploration abilities while lowering exploitation abilities.

As previously stated, both traditional deterministic algorithms and stochastic metaheuristic algorithms have benefits and drawbacks. The benefits of stochastic algorithms vastly exceed their drawbacks in terms of global optimization. Randomness appears to be favorable to the overall performance of algorithms, according to both empirical observations and simulations. However, determining how much randomness is appropriate is challenging because it depends on the algorithmic structure, the type of problems, and the intended solution quality for each type of problem.

2.3 Optimization Algorithms Analysis

In the literature, SI and bio-inspired computation have gotten a lot of attention. Bio-inspired algorithms, particularly SI-based algorithms, have become increasingly popular in the optimization, computational intelligence, and computer science areas. Nowadays, these nature-inspired metaheuristic algorithms have become some of the most frequently used optimization and computational intelligence techniques. SI-based algorithms like bee algorithms, particle swarm optimization (PSO), cuckoo search (CS), and firefly algorithms (FA) have a number of advantages over traditional algorithms. In this chapter, we look at the evolutionary operators and functionality of the essential components of these nature-inspired algorithms.

2.3.1 Iterative Processes

An optimization algorithm is an iterative process that begins with a guess and it may converge toward a stable solution, preferably the optimal solution to an optimization problem of interest, after a specified and large enough number of iterations [15]. This is basically a self-organizing system, with states representing solutions and attractors representing converged solutions. A collection of rules or mathematical equations can guide the evolution of such an iterative, self-organizing system. As a consequence, a complex system like this can interact and self-organize into convergent states, exhibiting certain emergent self-organization features. In this way, finding efficient techniques to replicate the development of a self-organizing system, particularly developing biological systems [16], is comparable to designing an effective optimization algorithm.

Assume the iterative process in (2.6) that tries to create a new and better solution $\mathbf{x}_{t+1}$ to a given problem from the existing solution $\mathbf{x}_t$ at each iteration or time t. For instance, finding the critical points or roots of $f'(\mathbf{x}) = 0$ in a d-dimensional space is similar to using the Newton-Raphson technique to determine the optimal value of $f(\mathbf{x})$ [17]. That is, for a deterministic method, the iterative equation in (2.6) is valid. Two methods for improving the convergence of (2.6) are the use of randomization and the introduction of parameters. Randomization is often employed in current metaheuristic algorithms, and in many situations, randomization takes the form of a collection of m random variables $\epsilon = (\epsilon_1, \ldots, \epsilon_m)$ in an algorithm [18]. In addition, an algorithm frequently has a collection of k parameters. The four parameters in PSO, for example, are two learning parameters, one inertia weight, and the population size. Normally, we might have a vector of parameters $p = (p_1, \ldots, p_k)$. We may design, in terms of mathematics, an algorithm with k parameters and m random variables as follows:

$$\mathbf{x}^{t+1} = h(\mathbf{x}^t, p(t), \epsilon(t)), \tag{2.7}$$

where h is a nonlinear mapping from a given solution $\mathbf{x}^t$, which is a d-dimensional vector, to a new solution vector $\mathbf{x}^{t+1}$.

Note that (2.7) corresponds to a trajectory-based, single-agent system. We may expand (2.7) for population-based algorithms with a swarm of n solutions to the following:

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}^{t+1} = h\left((x_1^t, \ldots, x_n^t), (p_1, \ldots, p_k), (\epsilon_1, \ldots, \epsilon_m)\right) \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}^t, \tag{2.8}$$

where $p_1, \ldots, p_k$ are k algorithm-dependent parameters and $\epsilon_1, \ldots, \epsilon_m$ are m random variables.

This interpretation of (2.7) is mostly dynamical or functional. It treats the functional (2.7) as a dynamical system that will eventually settle into equilibrium or attractor states. The eigenvalues of h and its parameters can be used to characterize the system's behavior in the context of linear and/or weakly nonlinear dynamical system theories. This, however, does not give enough insight into the diversity and complexity of the features. Instead, self-organization may give superior understanding.

Note that the number of iterations t required to discover an optimal solution for a certain accuracy is a big factor in the entire computing effort and algorithm performance. A better algorithm should utilize fewer iterations and less computation.

2.3.2 A Self-Organization System

A complex system can be self-organizing when given the appropriate circumstances. This occurs when the system's size is large enough and the number of degrees of freedom or potential states S is great enough. Furthermore, the system should be allowed to develop over an extended period of time away from noise and equilibrium states. Most importantly, a selection process must be in place to allow for self-organization. The following are the primary circumstances for self-organization in a complex system:

- The system size is large with an adequate number of freedom's degrees or states.
- The system has sufficient variability, such as noise, perturbations, edge of chaos, or it is far from equilibrium.

• The system is permitted to develop over a long time.
• In the system, there is a selection mechanism or a constant rule.

To put it another way, a system having states S will develop toward the self-organized state S_* if a mechanism $\alpha(t)$ with a set of parameters α is used. That is,

$$S \xrightarrow{\alpha(t)} S_*. \tag{2.9}$$

In terms of self-organization, an algorithm A, which includes the iterative process in (2.7), is a self-organization system that starts with a large number of potential states $\mathbf{x}^t$ and, driven by the iterative process of (2.7), seeks to converge to the best solution/state $\mathbf{x}_*$. That is,

$$f(\mathbf{x}^t) \xrightarrow{A} f_{\min}(\mathbf{x}_*). \tag{2.10}$$

We may compare the requirements for self-organization and algorithm features as presented in Tab. 2.1.

There are, however, some key distinctions between a self-organizing system and an algorithm. On the one hand, the paths to self-organization may not be obvious, and time is not a significant factor. On the other hand, the method by which an algorithm converges is critical, as is the speed with which it converges, in order to achieve genuinely global optimality at the lowest computing cost.

Table 2.1: Similarity between self-organization and an optimization algorithm.

Self-Organization	Features	Algorithm	Characteristics
Noise, perturbations	Diversity	Randomization	Escape local optima
Selection mechanism	Structure	Selection	Convergence
Reorganization	State changes	Evolution	Solutions

2.3.3 Exploration and Exploitation

The methods in which nature-inspired optimization algorithms explore the search space can also be investigated. In principle, all algorithms must have two important elements: exploitation and exploration, often known as intensification and diversification [19].

Exploitation employs any knowledge gleaned from the problem of interest to aid in the development of new solutions that are superior to existing ones. Nevertheless, this process is usually local, as is the information (such as the gradient). As a result, exploitation is for local search. Furthermore, exploitation has the benefit of achieving very fast convergence rates, but it has the problem of becoming stuck in a local optimum since the ultimate solution point is very dependent on the beginning position.

Exploration, on the other hand, allows for a more efficient search of the search space, as well as the generation of answers that are diverse and distinct from current solutions. As a result, exploration is for global search. Exploration has the benefit of being less likely to get trapped in a local mode, and the global optimality can be more easily reached. However, because many new solutions might be distant from the global optimality, it has the drawbacks of delayed convergence and the wasting of some computing resources.

As a consequence, proper balance is necessary in order for an algorithm to perform well. The system may converge more rapidly if there is too much exploitation and not enough exploration, but the likelihood of finding

the real global optimality may be minimal. Too little exploitation and too much exploration, on the other hand, might lead the search route to roam around with very sluggish convergence. The right balance should include the correct amount of exploration and exploitation, which can lead to the best algorithm performance. As a result, maintaining a sense of equilibrium is critical. Note that finding a process to attain such balance remains an unresolved topic, and no algorithm in the present literature can claim to have accomplished it.

2.3.4 Evolutionary Operators

It is also beneficial to examine how an algorithm works by looking at its operations directly. Consider the case of genetic algorithms (GA), which are invented by J. Holland and his associates in the 1960s and 1970s. GA are based on the abstraction of Darwinian development of biological systems, and employ genetic operators like crossover and recombination, selection, and mutation [20]. GA have been shown to offer several benefits over traditional algorithms. Gradient-free, parallelism, and highly explorative are three unique advantages. GA do not require gradient or derivative information, therefore they can handle complicated, discontinuous problems. Because crossover and mutation are stochastic, GA may more effectively traverse the search space, increasing the likelihood of finding the global optimum solution. Furthermore, because GA are population-based and use many chromosomes (meaning a set of properties or usually named genotype), they may be implemented in parallel [13].

The three most important evolutionary operators in GA are the followings:

• **Crossover:** Recombination of two parent chromosomes (solutions) to generate descendants (new solutions) by swapping part of one chromosome with a matching portion of the other.

• **Mutation:** To produce new genetic traits, a portion of a chromosome (a bit or multiple bits) is changed. Mutation might happen at a single site or several sites at once, and is as easy as switching between 0 and 1 in binary encoding.

• **Selection:** The survival of the fittest, which indicates that the population with the best quality chromosomes and/or traits will survive. This frequently takes the form of elitism, the most basic of which is to pass on the better genes to future generations of the population.

Crossover, in mathematical terms, is a mixing process in which there is substantial local search in a subspace [21], whereas mutation is a method for global exploration. In general, crossover is a local search operator that can be made global if the subspace is large enough, while mutation is a global search operator that can be made local if the mutation rate is low enough and the step sizes are small enough. As a result, the distinction between local and global might be hazy and subjective. New solutions will emerge as a result of both crossover and mutation. It is worth noting that crossover offers good mixing and has a limited diversity in the subspace. Although far-away solutions may move the population away from converged/evolved features, mutation can give more variety.

Selection, on the other hand, is a unique operator that serves a dual purpose: it selects the best solutions in a subspace while also acting as a driving force for self-organization or convergence. There is no driving force to pick what is optimal for the system without selection, therefore selection allows a system to grow toward a goal. Only the fittest solutions and desirable phases may be permitted to pass on with a suitable selection process, while unfit solutions in the population would eventually die off. Note that only the finest are chosen in selection, which can be as basic as a high degree of elitism, while other selection processes, such as fitness-proportional crossover, can, of course, be utilized.

These three evolutionary operators can be classified according to their roles and primary functions:

• Crossover is mostly used for blending inside a subspace. It will aid in the system's convergence.

• Mutation is a key process for global search, and it may be thought of as a form of randomization.

• Selection acts as a catalyst for the system's evolution toward the desired states. It is simply exploitation on a large scale.

It is worth noting that mutation may take several forms, one of which is using stochastic processes or randomization. The traditional Hooke-Jeeves pattern search (PS), for example, is a gradient-free method that has influenced a slew of other algorithms. The sequential increment of one dimension followed by the increment along the other dimensions is the essential step in PS. The steps will be tested and, if required, shrunk [22].

The PS concept may be produced using the following formula:

$$x_i = x_i + \Delta x_i = x_i + (x_{newmove} - x_i), \quad i = 1, 2, \ldots, d. \tag{2.11}$$

This can be written as a vector equation

$$\mathbf{x} = \mathbf{x} + \Delta \mathbf{x} = \mathbf{x} + (\mathbf{x}_{newmove} - \mathbf{x}). \tag{2.12}$$

In a d-dimensional space, Δx operates as a mutation operator in $2d$ directions. Note that differential evolution (DE) employs this type of mutation in higher dimensions.

2.4 Popular Nature-Inspired Algorithms

Nature-inspired optimization algorithms are a collection of algorithms inspired by natural phenomena, such as SI, biological systems, physical and chemical systems, and so on. There are several nature-inspired algorithms; around a hundred distinct algorithms and variations are believed to exist [23]. Clearly, even a small portion of these algorithms cannot be included inhere. Furthermore, rather than providing extensive explanations and background information for each algorithm, we will focus on the similarities and differences across algorithms, as well as the methods for creating new solutions, selecting the best solutions, and other key features.

2.4.1 Genetic Algorithms (GA)

The roots of contemporary evolutionary computing are basically formed by GA , which was pioneered in [20]. As previously mentioned, GA contains three major genetic operators: crossover, mutation, and selection (see Fig. 2.1). Although the original genetic algorithm did not include explicit mathematical equations, it did provide comprehensive methods and stages for creating descendants from parent solutions/strings.

Crossover aids in exploiting and enhancing convergence. From practical and theoretical research, it appears that crossover has a greater probability p_c in the range of 0.6 to 0.95, whereas mutation probability p_m is generally extremely low, around 0.001 to 0.05. These numbers indicate a high level of mixing and exploitation with a lesser level of exploration. In application, this means that GA may frequently converge effectively, and global optimality can be attained quickly in many instances.

The survival of the fittest provides a good method for selecting the best solution, where elitism may ensure that the best solution remains in the population, enhancing the algorithm's convergence. Nevertheless, under

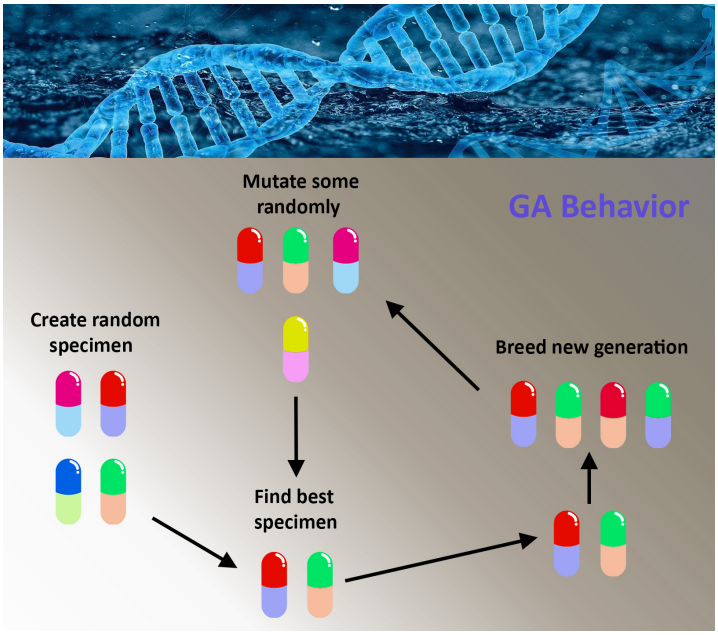


Figure 2.1: Genetic algorithm behavior.

specific conditions, the global optimality will be reached, and mutation may be viewed as a double-edged sword, as it can both enhance the likelihood of reaching the global optimality while also decreasing convergence.

2.4.2 Differential Evolution (DE)

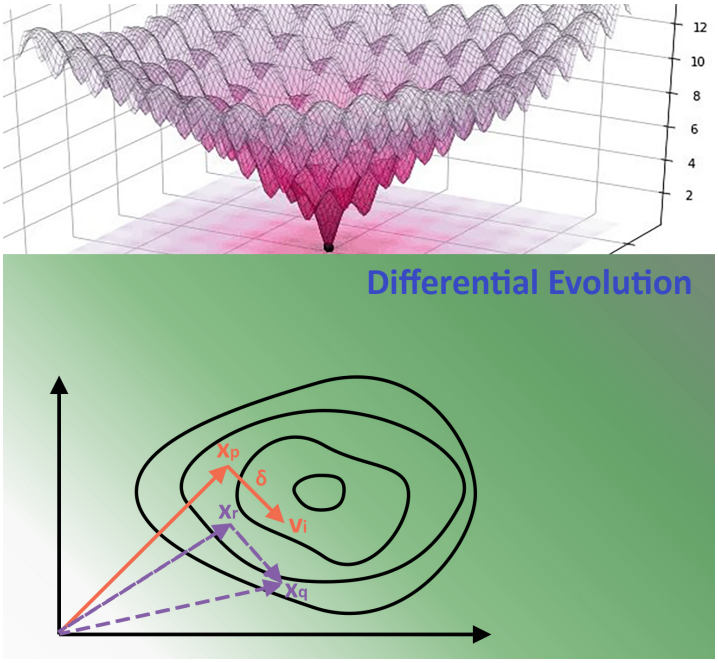


Figure 2.2: Differential evolution behavior.

DE was introduced by R. Storn and K. Price in 1996 and 1997 [24, 25]. It is worth noting that contemporary DE has a striking resemblance to the traditional PS's mutation operator. In reality, the mutation in DE may be

seen as a generic PS in any arbitrary direction $(\mathbf{x}_p - \mathbf{x}_q)$ by

$$\mathbf{x}_i = \mathbf{x}_r + (\mathbf{x}_p - \mathbf{x}_q)F. \tag{2.13}$$

where F is the differential weight in the range of $[0,2]$, and r, p, q, i denote four different integers generated by random permutation (see Fig. 2.2).

DE also includes a crossover operator, which is controlled by a crossover probability $C_r \in [0,1]$, and the crossover may be done in two ways: binomial or exponential. The selection process is substantially the same as in GA. It is to choose the most appropriate and, in the case of a minimization problem, the smallest objective value. As a result, we have the following:

$$\mathbf{x}_i^{t+1} = \begin{cases} \mathbf{v}_i^{t+1}, & \text{if } f(\mathbf{v}_i^{t+1}) \le f(\mathbf{x}_i^t) \\ \mathbf{x}_i^t, & \text{otherwise.} \end{cases} \tag{2.14}$$

where $f(\cdot)$ is the objective function and $\mathbf{v}_i^{t+1}$ denotes a new solution.

The majority of research has concentrated on the selection of F, C_r, the population size n, and the mutation scheme adjustment. Furthermore, when the condition in (2.14) is satisfied, it is apparent that selection is utilized. Crossover, mutation, and selection are used in almost all DE variations, with the primary changes being in the mutation and crossover steps. It is worth noting that there are over ten distinct versions of DE [26].

2.4.3 Particle Swarm Optimization (PSO)

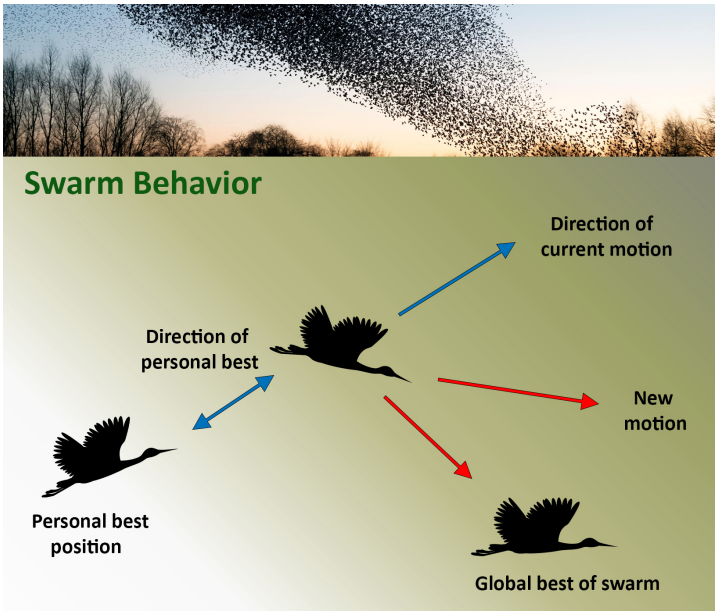


Figure 2.3: Particle swarm optimization behavior.

PSO was introduced by Kennedy and Eberhart in 1995 [18] and is based on swarm behavior (see Fig. 2.3). In principle, a particle's position $\mathbf{x}_i$ and velocity $\mathbf{v}_i$ could be updated in the following way:

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \alpha\epsilon_1[\mathbf{g}^* - \mathbf{x}_i^t] + \beta\epsilon_2[\mathbf{x}_i^* - \mathbf{x}_i^t], \tag{2.15}$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1}, \tag{2.16}$$

where ϵ_1 and ϵ_2 indicate two random vectors with values ranging from 0 to 1 for each entry. The learning parameters or acceleration constants are α and β, which may be written as $\alpha \approx \beta \approx 2$ in most cases.

In (2.15) and (2.16), we observe that the new position is created by a pattern-search mutation, while selection is done implicitly by utilizing the current global best solution $\mathbf{g}^*$ as well as the individual best $\mathbf{x}_i^*$ discovered so far. Nevertheless, the importance of individual best is unclear, even if, as shown in the accelerated PSO (APSO) [23], the present global best appears to be highly essential for selection. As a result, PSO is mostly made up of mutation and selection. Because PSO has no crossover, it can have great agility in particles with a significant level of exploration. The usage of $\mathbf{g}^*$, on the other hand, appears to be highly selective, which might be a two-edged sword. Its benefit is that it speeds up convergence by pulling toward the current best $\mathbf{g}^*$, but it could also cause premature convergence, even if this isn't the real optimal solution to the problem of interest.

2.4.4 Firefly Algorithms (FA)

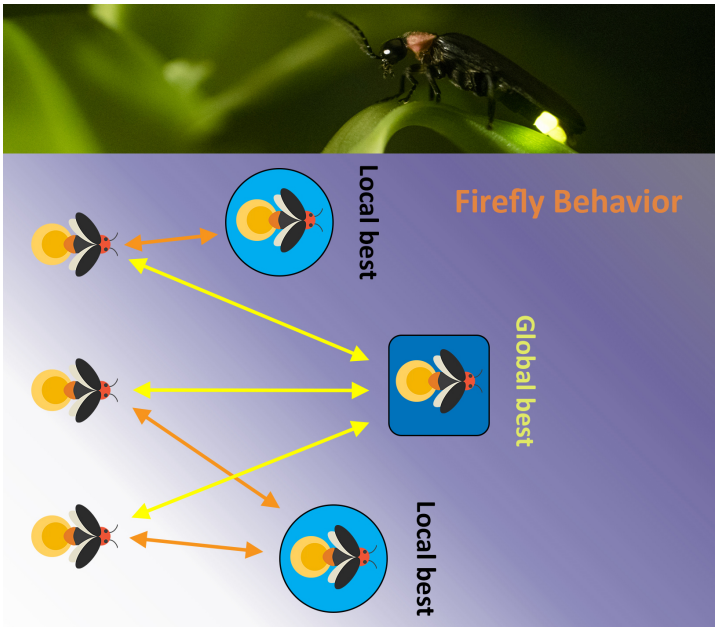


Figure 2.4: Firefly algorithm behavior.

FA is based on the behaviour of tropical fireflies and flashing patterns, and it was introduced by Xin-She Yang in 2008 [27]. FA is uncomplicated, adaptable, and straightforward to use.

The motion of a firefly i is attracted to another, more attractive (brighter) firefly j (see Fig. 2.4) can be formulated as follows:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \beta_0 e^{-\gamma r_{ij}^2}(\mathbf{x}_j^t - \mathbf{x}_i^t) + \alpha \epsilon_i^t, \tag{2.17}$$

where the second term is related to attraction, with β_0 representing attractiveness at zero distance $r = 0$. The third term is related to randomization, with α representing the randomization parameter, and ϵ_i^t being a vector of random numbers taken from a Gaussian distribution at time t. In [28], it has been used randomization in terms of ϵ_i^t, which may be readily expanded to other distributions like Lévy flights.

From (2.17), it can be observed that mutation is employed for both local and global search. When ϵ_i^t is calculated using a Gaussian distribution and Lévy flights, it results in larger scale mutation. If α is set to a low number, however, mutation will be confined to a subspace. Notably, since $\mathbf{g}^*$ is not utilized in FA, there is no specific selection in (2.17). Nevertheless, both ranking and selection are employed throughout the update in the two loops in FA.

FA has a unique characteristic in that it employs attraction, which is the first of its type in any SI-based algorithm. Because local attraction is greater than long-distance attraction, the population of FA may split into many subgroups, each of which can swarm around a local mode. Note that there is always a global optimum solution that is the real optimality of the problem between all local modes. Thus, FA is capable of dealing with multimodal problems in a natural and effective manner.

From (2.17), when $\gamma = 0$ and $\alpha = 0$, it can be observed that FA changes into a form of differential evolution. Furthermore, it changes into simulated annealing (SA) when $\beta_0 = 0$. Note that SA, one of the simplest stochastic algorithms, was introduced by Kirkpatrick et al. in 1983 [29] and is based on the features of the metal annealing procedure. In addition, FA changes into the APSO when $\mathbf{x}_j^t$ is substituted by $\mathbf{g}^*$. As a result, DE, APSO, and SA are all special instances of the FA, and the FA may have the benefits from all three algorithms. It is no wonder that FA is flexible and efficient, and that it outperforms competing algorithms like GA and PSO.

2.4.5 Cuckoo Search (CS)

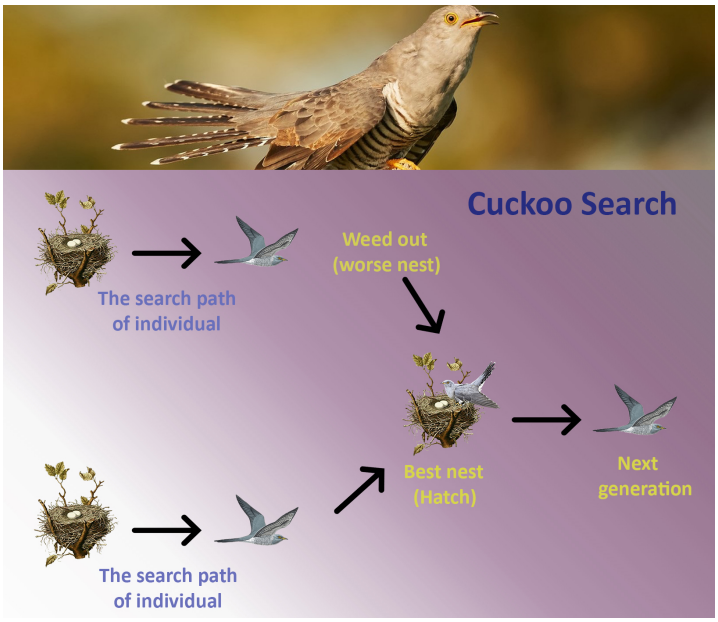


Figure 2.5: Cuckoo search behavior.

CS was introduced by Xin-She Yang and Suash Deb in 2009 [30], and is one of the latest nature-inspired metaheuristic algorithms. CS is based on some cuckoo species' brood parasitism (see Fig. 2.5). Furthermore, rather than ordinary isotropic random walks, this method benefits from the Lévy flights [31]. According to [13, 16], CS has the potential to be considerably more efficient than PSO and GA. A switching parameter p_α controls a balanced mix of a local random walk as well as a global explorative random walk in CS. The local

random walk may be formulated as follows:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \alpha s \odot H(p_\alpha - \epsilon) \odot (\mathbf{x}_j^t - \mathbf{x}_k^t), \tag{2.18}$$

where $\mathbf{x}_j^t$ and $\mathbf{x}_k^t$ imply two different solutions chosen randomly by random permutation, ϵ signifies a random number taken from a uniform distribution, $H(\cdot)$ signifies a Heaviside function, and s implies the step size. Note that $\odot$ denotes the Hadamard (or entry-wise) product.

The global random walk, on the other hand, is performed using Lévy flights [31]:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \alpha L(s, \lambda), \tag{2.19}$$

where

$$L(s, \lambda) = \frac{\lambda \Gamma(\lambda \sin(\pi\lambda/2))}{\pi s^{1+\lambda}}, \quad (s \gg s_0 > 0) \tag{2.20}$$

where $\Gamma(\cdot)$ is the gamma function. Note that the step size scaling factor is $\alpha > 0$, and it should be linked to the scales of the problem of interest.

In comparison to other algorithms like GA and SA, CS offers two major advantages: balanced mixing and efficient random walks. CS may be highly effective in global search since Lévy flights are generally considerably more efficient than any other random-walk-based randomization methods. Recent research suggests that CS can ensure global convergence [32]. Furthermore, egg similarity may lead to improved new solutions, that is basically fitness-proportional generation with high mixing capability. In other terms, Lévy flights offer variable mutation, and the fitness-proportional creation of new solutions based on similarity offers a subtle kind of crossover. Furthermore, selection is done using p_α, with excellent solutions being passed on to future generations and bad solutions being replaced by new ones. Moreover, simulations demonstrate that CS has the capacity to auto-zoom, meaning that new solutions can automatically zoom towards the region where the prospective global optimality is situated.

Additionally, in the context of Markov chains, (2.19) is basically the generalized SA. In (2.18), CS can change into a form of differential evolution when $p_\alpha = 1$ and $\alpha s \in [0, 1]$. In (2.18), CS can change into a form of APSO if $\mathbf{x}_j^t$ is replaced by the present best solution $\mathbf{g}^*$. This indicates that SA, DE, and APSO are all special instances of CS, which is one of the reasons behind the efficiency of CS.

In general, CS has significant mutation at both the local and global scales, and excellent mixing is accomplished by the use of solution similarity, that also serves as an analogous crossover. Elitism is used to choose solutions, which means that a high percentage of them will be handed on to the next generation. The premature convergence problem seen in PSO can be avoided without explicitly using $\mathbf{g}^*$.

2.4.6 Bat Algorithms (BA)

The metaheuristic bat algorithm (BA) is based on the echolocation behavior of microbats and was introduced by Xin-She Yang in 2010 [33]. It is worth noting that BA is the first algorithm of its type to employ frequency tuning (see Fig. 2.6). At iteration t, each bat is related to a location $\mathbf{x}_i^t$ and a velocity $\mathbf{v}_i^t$ in a d-dimensional search or solution space. There is a current optimal solution $\mathbf{x}_*$ between all the bats. As a result, the following three

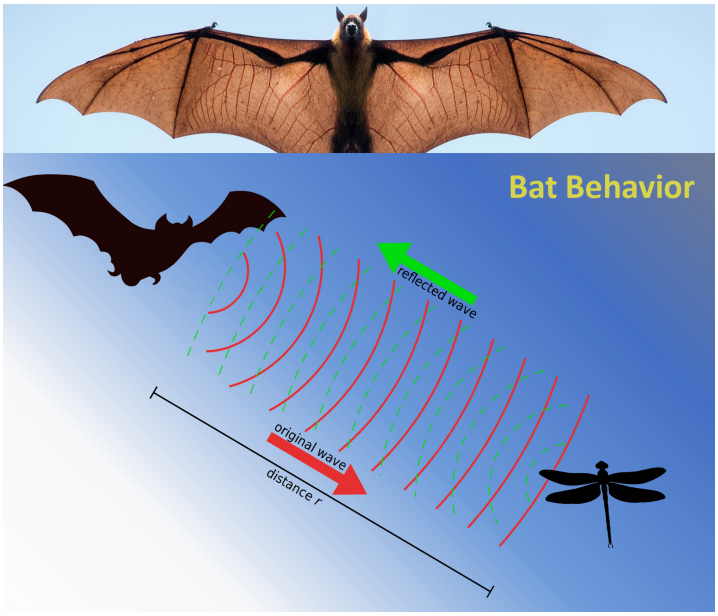


Figure 2.6: Bat algorithm behavior.

principles may be expressed as updating equations for $\mathbf{x}_i^t$ and velocities $\mathbf{v}_i^t$:

$$f_i = f_{\min} + (f_{\max} - f_{\min})\beta, \tag{2.21}$$

$$\mathbf{v}_i^t = \mathbf{v}_i^{t-1} + (\mathbf{x}_i^{t-1} - \mathbf{x}_*) f_i, \tag{2.22}$$

$$\mathbf{x}_i^t = \mathbf{x}_i^{t-1} - \mathbf{v}_i^t, \tag{2.23}$$

where $\beta \in [0, 1]$ is a random vector taken from a uniform distribution.

The following equations control the loudness A_i^{t+1} and pulse emission rates r_i^{t+1}:

$$A_i^{t+1} = \alpha A_i^t, \tag{2.24}$$

and

$$r_i^{t+1} = r_i^0 [1 - e^{-\gamma t}], \tag{2.25}$$

where $0 < \alpha < 1$ and $\gamma > 0$ are constants.

Frequency tuning serves as mutation in BA, although selection pressure is kept relatively constant by using the current optimal solution $\mathbf{x}_*$ discovered thus far. Although there is no explicit crossover, mutation differs owing to loudness and pulse emission changes. Furthermore, when the search approaches the global optimality, the changes in loudness and pulse emission rates give an auto-zooming capability, allowing for more extensive exploitation.

2.4.7 Artificial Bee Colony (ABC)

Nature-inspired bee algorithms have emerged as a promising and strong tool in recent years [34, 35]. The basis of bee algorithms is a bee's capacity to communicate or broadcast to certain nearby bees, allowing them to "know" and "follow" a bee to the optimal source, places, or routes to accomplish the optimization goal. The

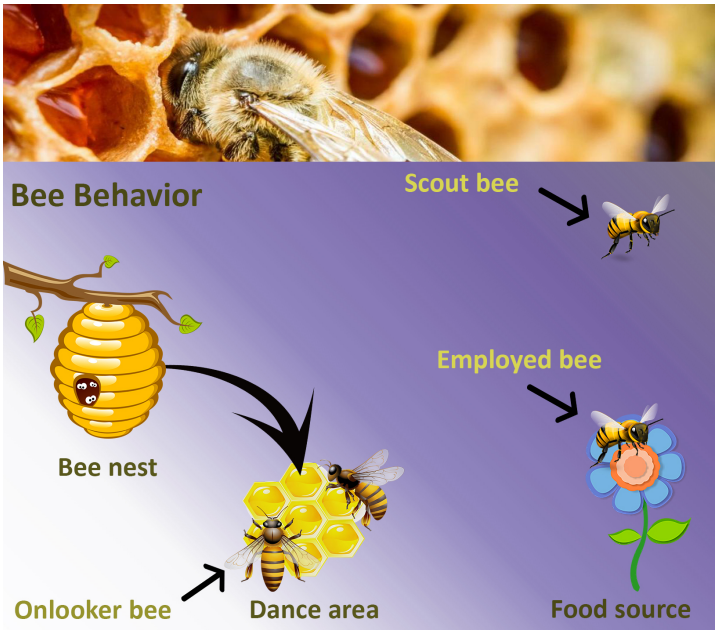


Figure 2.7: Bee algorithm behavior.

specifics of the implementation will be determined by the algorithms themselves, which may alter significantly and vary with different versions. In [14], there has been provided a comprehensive description of several versions of bee algorithms, such as the artificial bee colony (ABC) optimization, the honeybee algorithm, the virtual bee algorithm, the honeybee-mating algorithm, etc.

Take for example the ABC optimization algorithm, which was first developed by D. Karaboga in 2006 [36]. In the ABC algorithm, the bees in a colony are classified into three groups: onlooker bees (observer bees), employed bees (forager bees), and scouts (see Fig. 2.7). There is only one employed bee for each food source and, hence, the quantity of employed bees is proportional to the amount of food sources. The employed bee of a rejected food source is compelled to work as a scout, looking for new food sources at random. Employed bees in a hive exchange information with onlooker bees so that they might pick a food source to forage. Scout bees and employed bees both do randomization, and both rely heavily on mutation. Furthermore, while there is no explicit crossover, selection is linked to honey or objective.

To express the quantity of nectar at position $\mathbf{x}$, a given objective function $f(\mathbf{x})$ could be translate as $F(\mathbf{x})$. As a result, the probability P_i of an onlooker bee picking the chosen food source at $\mathbf{x}_i$ may be calculated as follows:

$$P_i = \frac{F(\mathbf{x}_i)}{\sum_{j=1}^{S} F(\mathbf{x}_j)}, \tag{2.26}$$

where S implies the number of food sources.

The intake efficiency of a specific food source is defined by F/T, where F is the quantity of nectar consumed and T is the length of time spent at the food source. If a food source is tried/foraged a certain number of times and does not improve, it is abandoned, and the bee in that site will move on to new spots at random.

ABC relies solely on fitness-related selection and mutation, and it is capable of doing an excellent global search. Furthermore, it can successfully traverse the search space, but because it lacks crossover, convergence could be sluggish, and therefore subspace exploitation is restricted. In reality, many metaheuristic algorithms do not use

crossover.

In regard to exploration and exploitation, bee algorithms have a great exploring capability but a relatively poor exploitation capacity. This might explain why they can handle some difficult optimizations relatively well, yet the computing effort, like the amount of function evaluations, can be rather large.

2.4.8 Slime Mould Algorithm (SMA)

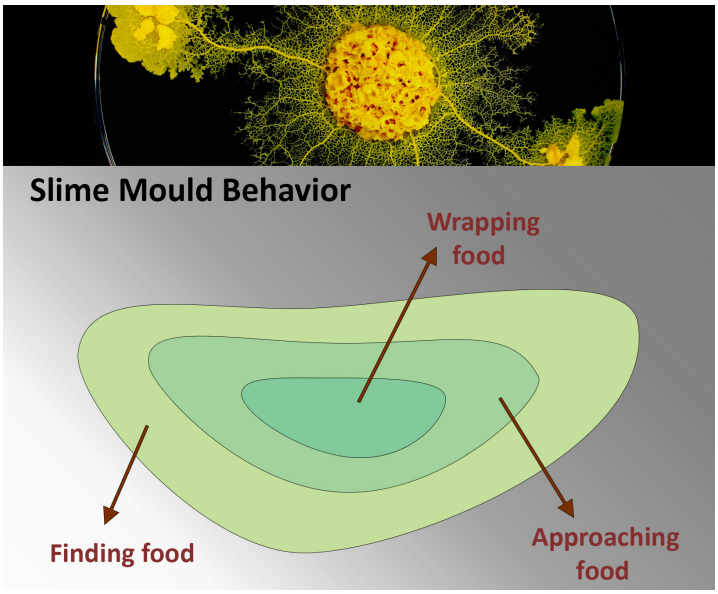


Figure 2.8: Slime mould algorithm behavior.

The slime mould Physarum polycephalum is a big, single-celled amoeboid organism with a body that consists of tubes. Assume that the form of Physarum is represented by a graph, with plasmodial tubes referring to the graph's edges and tube junctions referring to the graph's nodes. It has the ability to build a dynamic tubular network that connects the food sources identified during foraging. The physiological process behind tube production and selection adds to the Physarum's path-finding ability: tubes thicken in a particular direction when the flow through it continues in this direction for a set period of time. It functions as a nonlinear spatially extended intelligent active medium encased in an elastic membrane. In attractant and repellant designs, the cell optimizes its development patterns. Physarum grows as an omnidirectional wave on a nutrient substrate, such as a standard excitation wave in a 2-dimensional excitable medium (see Fig. 2.8).

To describe the approaching method of slime mould in food, the following rule is created as a mathematical equation to initiate contraction mode:

$$\mathbf{x}^{t+1} = \begin{cases} \mathbf{x}_b^t + v_b(w\mathbf{x}_\alpha^t - \mathbf{x}_\beta^t), & r < p \\ v_c\mathbf{x}^t, & r \geq p, \end{cases} \tag{2.27}$$

where $\mathbf{x}_b^t$ denotes the current individual position with the strongest odor intensity, $\mathbf{x}_\alpha^t$ and $\mathbf{x}_\beta^t$ denote two randomly chosen individuals from the current population, w signifies the weight of slime mould and $p = \tanh|f(\mathbf{x}^t) - f(\mathbf{x}_*)|$. In addition, v_b and v_c are parameters, where v_b oscillates in a random manner in $[-a, a]$ and v_c oscillates in a random manner in $[-1, 1]$. Both v_b and v_c converge to zero eventually.

Slime mould algorithm (SMA) relies solely on fitness-related selection and its design is dynamic, with a stable balance of local and global search drifts. Although SMA lacks crossover and mutation, it has outperformed several competing algorithms in solving real-world optimization problems in research and industry. It can solve a variety of graph theory problems, such as the shortest path problem [37] and network design [38].

2.4.9 Beetle Antennae Search (BAS) Algorithm

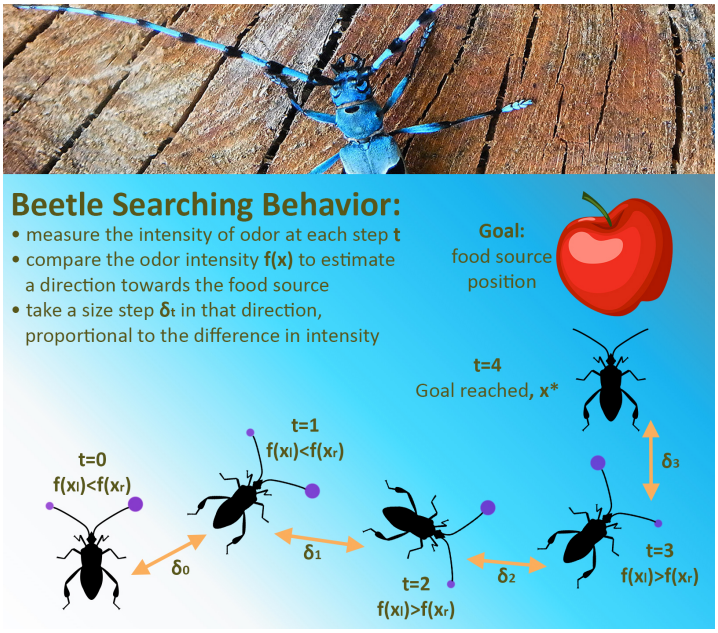


Figure 2.9: Beetle antennae search behavior.

A beetle explores its nearby area randomly using both antennae in order to receive the odor when it is preying or finding mates. When the antenna in one side detects a higher concentration of odors, the beetle would turn to the direction towards the same side, otherwise, it would turn to the other side (see Fig. 2.9). A meta-heuristic optimization algorithm, known as Beetle Antennae Search (BAS), which mimics the searching behavior of a beetle with two antennae was presented in [39].

At t-th iteration, consider the location of beetle as a vector, $\mathbf{x}_t$, $t = 1, 2, \ldots$, and signify the odour concentration at position $\mathbf{x}$ to be $f(\mathbf{x})$ defined as a fitness function. The maximum of $f(\mathbf{x})$ refers to the source odour point. The searching behavior model is described by the random path of the beetle searching as described in the following:

$$b = \frac{\text{rnd}(g,1)}{\|\text{rnd}(g,1)\|}, \quad B = \frac{b}{2^{-52} + \|b\|}, \tag{2.28}$$

where rnd$(\cdot)$ implies a random function, and g denotes the dimensions of position. For imitating the movements of the beetle's antennae, the searching behaviors of right-hand ($\mathbf{x}_R$) and left-hand ($\mathbf{x}_L$) side are formulated as follows:

$$\mathbf{x}_R = \mathbf{x}_t + Bd_t, \tag{2.29}$$

$$\mathbf{x}_L = \mathbf{x}_t - Bd_t, \tag{2.30}$$

where d is the sensing diameter of antennae equivalent to the exploit ability. Furthermore, the behavior of detecting can be formulated as follows:

$$\mathbf{x}_t = \mathbf{x}_{t-1} + B\delta_t \text{sign}(f(\mathbf{x}_R) - f(\mathbf{x}_L)), \tag{2.31}$$

where δ is the search step size that accounts for convergence speed after a decreasing t function, and sign$(\cdot)$ represents a sign function. Lastly, the update rules of d and δ are the followings:

$$d_t = z_1 d_{t-1} + z_2, \tag{2.32}$$

$$\delta_t = z_1 \delta_{t-1}. \tag{2.33}$$

where z_1 and z_2 are tuning parameters that are usually belong to $[0, 1] \subseteq \mathbb{R}$. A diagram of the BAS algorithm is presented in Fig. 2.10.

BAS is one of the simplest metaheuristic algorithms, relying only on fitness-related selection and achieving a decent balance of local and global exploitations. BAS has surpassed numerous rival algorithms in tackling real-world optimization issues in research and industry, despite its absence of crossover and mutation, while its simplistic design permits novel optimisation algorithms to be developed [40–44].

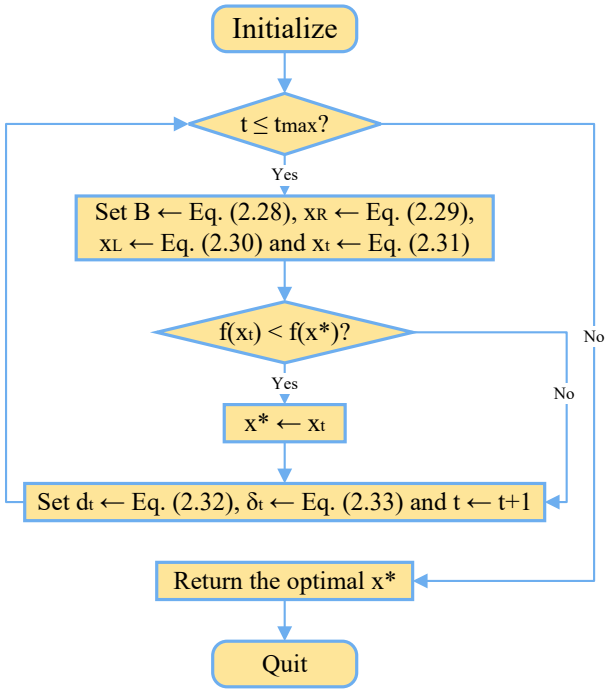


Figure 2.10: Beetle antennae search algorithm.

2.5 Parameter Tuning and Parameter Control

Algorithm-dependent parameters exist in all nature-inspired algorithms. The values of these parameters have a significant impact on an algorithm's behavior and performance. The question of how to appropriately adjust and regulate these algorithms remains unsolved [45].

A parameter-tuning tool, i.e., a tuner, is required to tune $A(\Phi, p)$ for a given problem Φ and algorithm A with a number of parameters p to obtain the optimum performance. Even if we have strong tools for tuning an algorithm, the optimum parameter setting and hence performance are entirely dependent on the performance metrics utilized in the tuning. The parameters should, in theory, be able to withstand random seeds, modest parameter changes, and even problem instances [45]. However, these may not be feasible in practice. Parameter tuning may be classified into iterative and non-iterative tuners, single-stage and multistage tuners, according to [45]. The definitions of these terminologies are self-evident. Existing tuning approaches include sample techniques, screening techniques, model-based techniques, and metaheuristic techniques. Because the success and efficiency of these approaches varies, no well-established techniques for universal parameter adjustment exist.

Another problem related to parameter tuning is parameter control. Parameter values are frequently constant during iterations after parameter tuning, but parameters must fluctuate during iterations for parameter control. The goal of parameter control is to change the parameters of an algorithm such that it can obtain the best convergence rate and consequently the best performance. As a consequence, another difficult optimization problem to address is the optimization of the parameter control [33].

3

An Introduction to Neural Networks

The main purpose of this chapter is to provide a brief overview of artificial neural networks (ANNs), also known as neural networks (NNs), and the NN techniques used in this dissertation, including NN solvers for linear and quadratic programming.

ANNs are computing systems modeled after the biological NNs that make up animal brains. A typical NN is made up of several simple, connected processors known as neurons, each of which produces a sequence of real-valued activations. Sensors detecting the environment activate input neurons, whereas weighted connections from previously active neurons stimulate additional neurons. By triggering actions, some neurons may be able to alter the surroundings. Finding weights that cause the NN to exhibit desired behavior is the goal of learning or credit assignment. Depending on the problem as well as how the neurons are connected, such behavior may necessitate extended causal chains of computational steps, each of which modifies the aggregate activation of the network (typically in a non-linear way). Deep Learning is concerned with appropriately allocating credit over a variety of stages.

The origins of NNs can be traced back to artificial intelligence (AI). The basic scientific goal of AI is to decipher the principles that allow natural or artificial systems to behave intelligently [46]. The goal of early AI research was to make computers think and derive facts [47]. The focus of modern AI research has switched from designing and developing intelligent agents to designing and building intelligent agents. A living object, a robot, a sensor, or an automobile can all be considered agents. Computational agents whose decisions can be justified are of particular interest in AI. Mathematics, logic, philosophy, probability theory, linguistics, neuroscience, and decision theory are the foundations of AI. Computer vision, robotics, natural language processing, and machine learning (ML) are just a few of the fields where AI can be used.

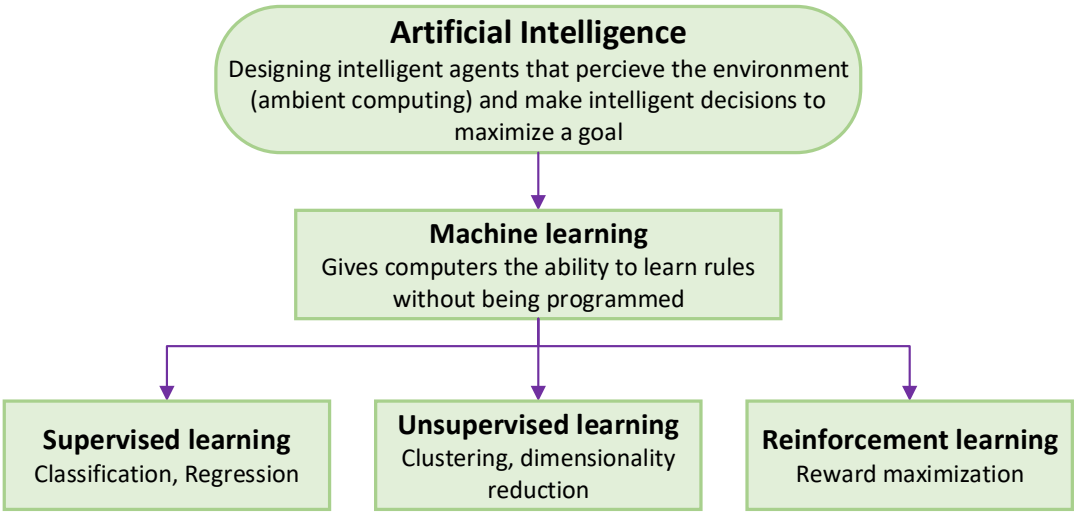


Figure 3.1: Machine learning areas.

AI has numerous subfields, including ML. Without being explicitly programmed, ML algorithms construct models that can find interesting patterns in data and make predictions. ML learning can be split into three basic categories: supervised, unsupervised, and reinforcement. For classification and regression, supervised methods are utilized, while for clustering, unsupervised methods are utilized. Reinforcement learning is concerned with how agents operate in a given environment in order to maximize a reward concept. It is also worth mentioning that there is a classification called semi-supervised learning, which sits in between unsupervised and supervised learning. The basic subareas of ML are shown in Fig. 3.1, which also summarizes the tasks. Other classifications of ML methods can, however, be found in the literature.

On a computer, ML seeks to mimic human learning. Human learning is primarily based on experience, whereas ML is based on data. A ML project's starting point is data, which is utilized to train and test a model. Labeled data is required for supervised methods. Data that has been accurately tagged is referred to as labeled data. The labeled data is utilized to train and evaluate the learner's accuracy. Because unsupervised learners do not use labeled data, evaluating a trained cluster algorithm is more difficult.

Common supervised and unsupervised methods are presented in the graph of Fig. 3.2. Because some learning systems can be used for both supervised and unsupervised tasks, the classification is not exhaustive. For instance, ANNs can be used in a supervised as well as in an unsupervised setting. Ensemble learning, reinforcement learning, and active learning are some of the most commonly utilized learning methods. Ensemble learning brings together many supervised methods to create a more powerful learner. Reinforcement learning algorithms are reward-based algorithms that learn how to achieve a challenging goal, while active learning is a special form of semi-supervised learning.

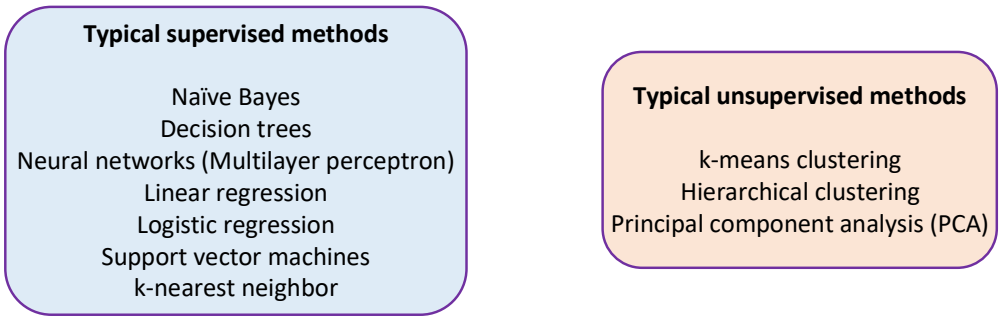


Figure 3.2: Common supervised and unsupervised methods.

3.1.1 Supervised learning

The term "supervised" alludes to the requirement for labeled training data. The learner is trained with a set of pre-classified, labeled data in order to categorize fresh data into distinct categories. The categories are referred to as "labels". During training, the learner generates an internal representation of the training data that constitutes a generalization. Induction is the process of forming internal representations. Unsupervised learning is used when there is no labeled training data. The learner must find the categories during training when utilizing unsupervised approaches. Semi-supervised learning methods can be utilized when there is a small amount of labeled data and a big number of unlabeled data. ANNs, support vector machines, k-nearest neighbor, Bayesian models, and decision tree induction are examples of supervised learning techniques. Hierarchical clustering and k-means clustering are two common unsupervised learning techniques. Because there is no labeled data in unsupervised learning, the taught learner's accuracy cannot be assessed. Instead, measures like density estimation and well separation are used. The vast majority of ML initiatives employ supervised methods.

An algorithm in ML is a repeatable technique for obtaining a trained model from a set of data. Each algorithm has a distinct personality and can be used to solve a variety of issues. The amount of bias and variation produced by supervised learning algorithms is one of the fundamental differences between them. Note that bias and variation are called prediction errors. The disparity between the expected and actual, measured prediction is known as bias. The sensitivity of an algorithm to a certain set of training data is referred to as variance. Both errors, bias, and variance are minimized throughout training until they converge. The irreducible error, a third prediction error, is caused by data noise or randomness. It is impossible to diminish it through training. In other circumstances, however, improved data cleansing steps may reduce the irreducible error.

For classification and regression, supervised learning is applied. When the goal features are discrete, classification is employed; when the target features are continuous, regression is utilized. Because classification techniques may be used to support decision making, they are used to solve the bulk of analytic problems.

Supervised learning needs, next to training data: input features, target features, and measure of improvement. The training data is used to extract the input features, which are commonly represented as an input vector. The output or response variable(s) is the target feature(s). Every learning cycle, the measure of improvement is utilized to see if we are improving. The ML scheme learns to map input features to target features during training. In other words, the learner's internal representation is tweaked until the target accuracy is achieved. The mapping is frequently an estimate, and 100% precision is rarely obtained. A learner's accuracy is a performance metric.

Precision and recall, as well as the receiver operating characteristic, are examples of additional evaluators.

A complete training cycle commonly goes through the following five steps:

1. Data collection
2. Data pre-processing
3. Training a ML algorithm, i.e. fitting model parameters
4. Testing/evaluating trained learning scheme
5. Measuring error

Training a ML algorithm is an iterative process that often takes many iterations to achieve good results. In addition, numerous ML algorithms are frequently trained and assessed. The best model is then utilized to create predictions based on previously unseen data. If the results aren't sufficient, more complex learning methods can be developed and tested. If two learners perform equally well, the Occam's razor principle dictates that the simpler one be chosen.

Classification and Regression Analysis

On the one hand, we try to assign a label to a test instance in classification. A classifier is a learning algorithm that assigns an instance to a category. A discrete target label y is predicted through classification. We have a binary classification problem if there are just two labels, and a multiclass classification problem if there are more labels. A classifier learns a function f that maps an input x to an output y from labeled training data, as follows:

$$y = f(x) + \epsilon, \tag{3.1}$$

where x and y denote the input and output, respectively, and ϵ signifies the irreducible error. Note that the irreducible error stems from noise and randomness in the training data and it cannot be reduced during training. Common classifiers include Bayesian models, decision trees, support vector machines and ANNs.

Regression analysis, on the other hand, is a set of statistical procedures for estimating the relationship between variables. It's used to see if there's a link between two variables. One of the most essential data analysis methodologies is regression analysis. The most prevalent regression analysis algorithms in ML are linear and logistic regression, however there are many others.

Forecasting, time series analysis, and correlation analysis all use regression analysis to make data-driven judgments. It's crucial to remember, however, that correlation does not imply causality. A regression line fitting a group of data points well does not imply a cause-and-effect relationship. In essence, regression analysis seeks to fit a line or curve to a group of data points so that the distance between the line and the data points is as small as possible. A regression curve fitted to a collection of data points is shown in Fig. 3.3a.

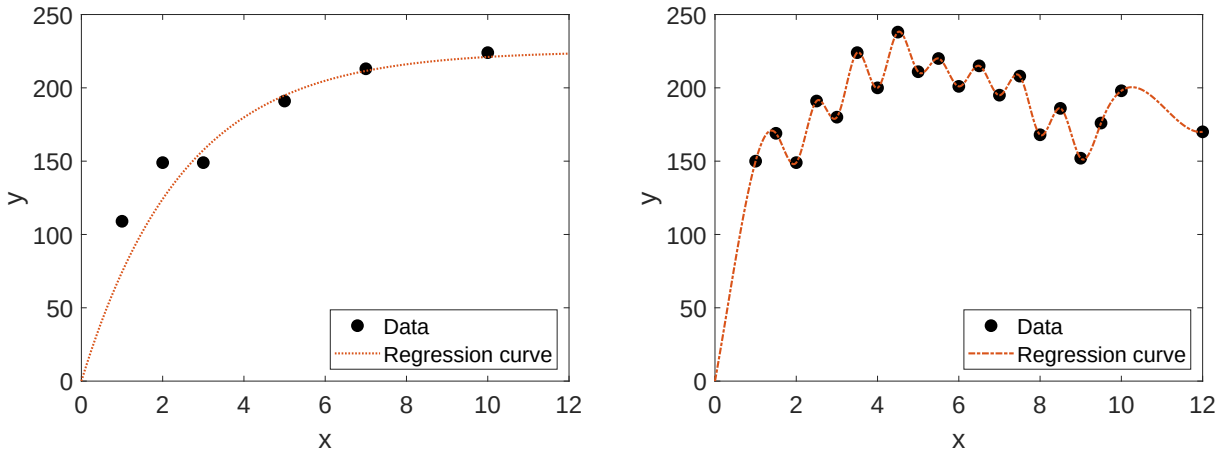
Some remarks about Fig. 3.3a are the following four:

- Not every point is on the curve, and some are very far away from the regression line.
- The regression line is an approximation.
- It shows if the dependent and independent variables have a substantial relationship.
- Different scales can be used to compare the effects of the measured variables.

Regression analysis is a statistical method for determining how input and output variables are related. In ML, we are more concerned with creating accurate predictions through minimizing a model's error. Regression is

used to figure out:

- The dependant variable(s), how effectively a group of predictor variables predicts an outcome.
- Which variables are significant in determining the outcome.



(a) Regression analysis.

(b) An overfitted regression curve.

Figure 3.3: Regression curves.

To produce predictions, there are a variety of regression approaches available. New regression algorithms can also be developed. A curve has been constructed in Fig. 3.3a since a straight line would not provide a reasonable fit for the data points. Which regression method is best for you is mostly determined by three factors: the number of independent variables, the type of the dependent variables, and the shape of the regression line.

Regression analysis is used to describe a phenomenon and allows us to discover which variables have an impact, which factors matter the most, which factors can be ignored, how the factors are interrelated, and, most crucially, how certain we are about these factors using mathematics.

Polynomial regression, lasso (least absolute shrinkage and selection operator) regression, and stepwise or ridge regression, to name a few, are all types of regression in addition to linear and logistic regression. If necessary, regression models can be mixed. A linear model is usually utilized first. If a good fit cannot be established, a nonlinear model can be used because it can fit a wider range of curves.

Overfitting is an issue in regression analysis, as it is in all ML techniques. Linear regression will overfit when the input data is highly correlated. Other regression models, such as polynomial regression, are equally prone to overfitting, necessitating the application of data regularization techniques to avoid illogical outcomes. An overfitted regression curve is seen in Fig. 3.3b. A learner that is overfit captures noise and does not generalize well to new, unknown input. To reduce the difficulty, some regression models, such as lasso or ridge regression, have been created.

Artificial Neural Networks

Fig. 3.4 shows one of the most basic configurations of ANNs. It is made up of three layers: an input layer, a hidden layer, and an output layer. It may be used for binary classification or even multiclass classification because it contains two output neurons. Each input neuron in the ANN depicted in Fig. 3.4 is connected to every neuron in the hidden layer, and every neuron in the output layer is connected to every neuron in the hidden layer. The

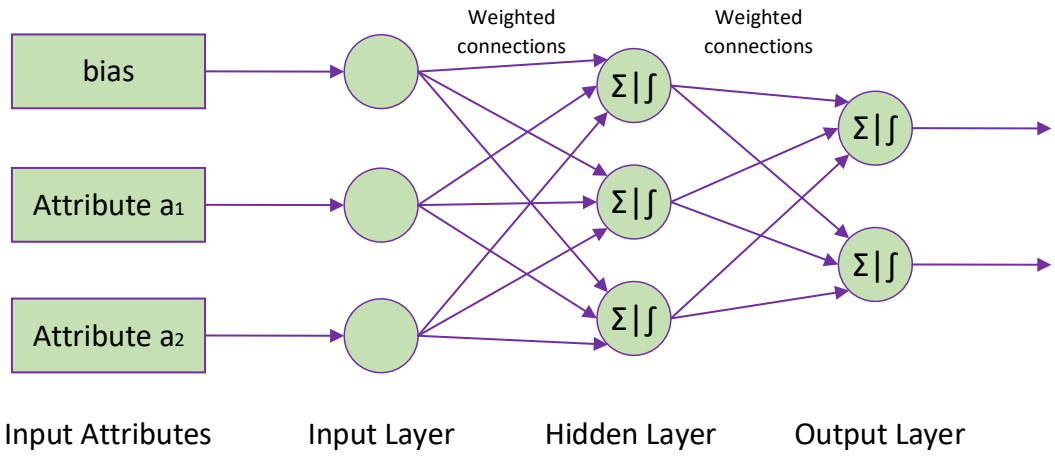


Figure 3.4: Artificial neural network.

layers are considered to be totally connected in this way. It's called a feedforward NN because the signal goes straight from the input layer to the output layer. Neurons can also have recurrent connections with themselves, as shown in Fig. 3.5, or can have connections with upstream neurons to form a recurrent NN. ANNs offer several advantages, including a high tolerance for noisy input and the ability to classify patterns that they have not been trained [48]. They can be employed when there is a lack of understanding of the traits and their relationships.

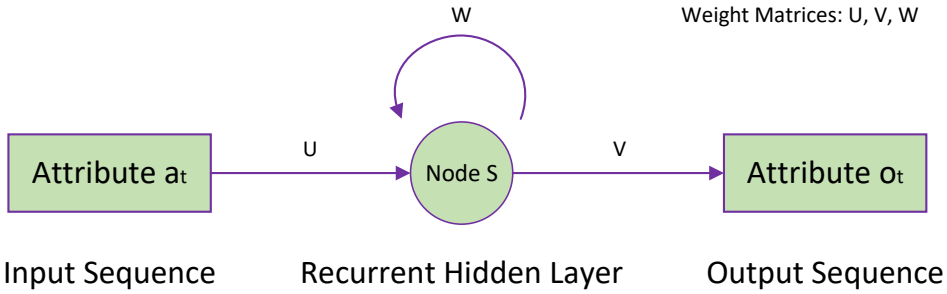


Figure 3.5: Recurrent neural network.

Perceptron, Multilayer Perceptron and Back-propagation

As shown in Fig. 3.6, a perceptron is made up of input signals $x_0, x_1, \ldots, x_n$ paired with weights w. Actual observations or, in the event of multiple layers, intermediate values from upstream perceptrons might be used as inputs. The sum of the weighted input signals is sent to an activation function (AF). It fires (emits) a signal if the AF is sufficiently stimulated.

Different types of AFs exist in general. The action potential of a genuine neuron is represented by an AF. In its most basic form, it is a step function that either fires or does not fire when appropriately stimulated. It is binary since it produces either 1 or 0 and only separates classes linearly. The perceptron computes $\hat{y} = \sum_{i=0}^{n} w_i x_i + b$. The signal is sent from the input node to the output node without being transformed. The bias is added to the

total of the weighted inputs in the output node. The bias b is a constant that functions in the same way as the intercept. It moves the AF to the left or right along the x-axis.

The weights are learned during training employing the following weight update formula:

$$w_j^{k+1} = w_j^k + \lambda(y_i - \hat{y}_i^k)x_{ij}, \tag{3.2}$$

where w^k denotes the weight of i-th input after the k-th iteration, λ signifies the learning rate, $\hat{y}_i^k$ implies the predicted output, x is the value of j-th attribute of training instance x_i and y_i is the target output.

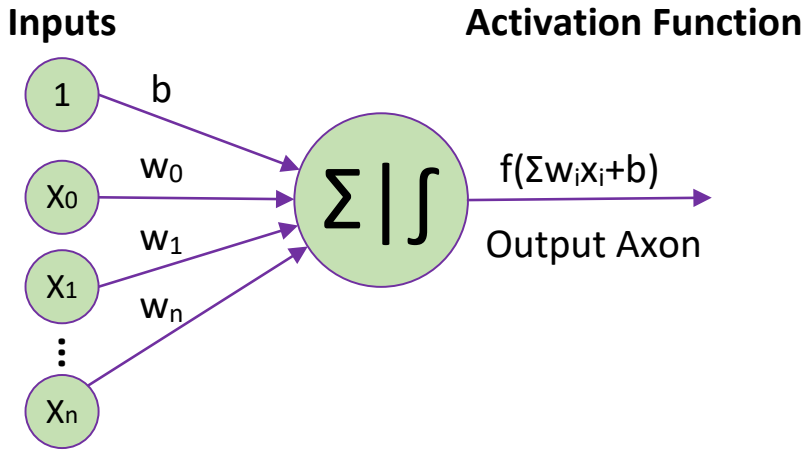


Figure 3.6: Perceptron.

Typically, random numbers are used to set the initial weights. The prediction error is computed as $(y - \hat{y})$. As illustrated in (3.2), the new weight is a combination of the old weight and the prediction error. The learning method iterates as many times as necessary until the output is converging, that is, the perceptron output properly classifies the training cases. The learning rate λ determines the amount of weight adjustment per iteration and is a number between 0 and 1. If the learning steps are too tiny, that is, λ is close to 0, the method may take a long time to converge; on the other hand, if they are too large, the process may never fully converge. As a result, some implementations alter the learning rate during training as well.

As seen in Fig. 3.4, multilayer perceptrons (MLPs) are a type of feedforward network that has at least three layers: an input layer, one or more hidden middle layers, and an output layer. The layers, with the exception of the input layer, are made up of perceptrons with nonlinear AFs. Feedforward networks carry signals in a single route from the input layer through the middle layer(s) to the output layer, with no cycles in between. Forward-propagation is the term for this. Data points that are not linearly separable can be separated using a MLP. A sigmoid AF is commonly used in MLPs. The AFs of all neurons in a neural network do not have to be the same. The sigmoid or logistic AF in (3.3), which is also used in logistic regression, and the hyperbolic tangent in (3.4) are two common AFs in a MLP.

$$y(x_i) = (1 + e^{-x_i})^{-1}, \tag{3.3}$$

$$y(x_i) = \tanh(x_i), \tag{3.4}$$

where e denotes the Euler's number, y_i is the output of the i-th node, while x_i is the input to the node and is the weighted sum of the input signals plus the bias. The sigmoid function is called after its S-shaped form and ranges from 0 to 1, but never reaches 0 or 1 as $x \to \infty$.

The output of a perceptron with AF f is as follows:

$$y(x_i) = f(w_1 x_1 + w_2 x_2 + \cdots + w_j x_j + b), \tag{3.5}$$

where $y(x_i)$ is the output of the perceptron, w_j denotes the weight of the j-th input of the i-th neuron and b implies the bias.

Back-propagation (BP) is a weight-adjustment algorithm that uses a learning process. The error or loss at the output of the MLP is calculated and propagated back through the network after each training iteration. The weights are modified so that the next iteration's error is as small as possible. Typically, the initial weights are chosen at random. We do not know what the hidden layer's correct weights are. Because it is not feasible to examine and correct the values of the neurons in the middle of the network directly, they are referred to as the hidden layer [47]. The gradient of the loss or error function at the output must first be calculated. The gradient is generated using the derivative of the loss function, and the gradients of each layer are calculated using the chain rule. The gradient is minimized in each iteration until the function's minimum is found. Gradient descent is the name for this method. The gradient descent algorithm adjusts the weights through BP. BP of errors is a term used to describe this process. Gradient descent is an iterative optimization approach for determining a function's minimal value. The loss function must be differentiable in order to determine the gradient. The mean squared error (MSE) function is the most popular loss function in ML. The MSE is easily distinguishable and is frequently utilized in regression analysis. The log loss function, which is used for classification, is another prominent loss function. Other loss functions and variants are available, as is customary.

A complete training iteration goes through the following steps:

1. Forward-propagation
   (*a*) Propagate input forward through the network to calculate the output
   (*b*) Calculate the error at the output
2. Back-propagation
   (*a*) Backpropagate the error through the network
   (*b*) Adjust the weights

Batch training is used to train the MLP. The network is presented with the entire training set in each training iteration. To change the weights, the average sum-of-squared errors are calculated and transmitted back. The entire cycle, in which all of the training cases have been transmitted through the network, is referred to as an epoch. Sequential training, in which the error is computed and the weights are updated for each training input, is a considerably easier technique to implement training. Sequential training is normally less efficient, but because it is easier to program, it is frequently used.

Unlike decision trees, which are easy to understand, MLPs have been criticized for being difficult to explain, particularly their inner workings. We know the weights and AFs of a trained network, but that does not tell us much. Deep learners have the same issue too.

Bayesian models

The Bayes theorem is the foundation of Bayesian models [49]. In general, the Bayes classifier reduces the likelihood of misclassification. It is a model that uses the posterior distribution to make judgments. A prior distribution and a likelihood are used in Bayesian models, and Bayes' theorem connects them. The Bayes rule divides the computation of a posterior probability into likelihood and prior probability computations. It computes the posterior probability $P(c|x)$ from $P(c)$, $P(x)$ and $P(x|c)$, as follows:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}, \tag{3.6}$$

where $P(c|x)$ denotes the posterior probability of class c (target) given predictor x, $P(x|c)$ implies the likelihood which is the probability of predictor given class c, $P(c)$ is prior probability of class c and $P(x)$ is the prior probability of predictor x.

In Bayesian models, probability is expressed as a degree of belief in an event that can vary as new information becomes available. The Bayes rule describes how to infer hypotheses from data when inference uncertainty is stated as a probability. Inference can be exhibited in the form of learning and prediction.

Decision Trees

Another sort of extensively used classifier is decision trees, often known as decision tree induction. The learning of decision trees from class-labeled training tuples is known as decision tree induction [48]. Except for the leaf nodes, it generates a tree-like structure in which each node indicates a judgment for an attribute. In computer science, trees are one of the most frequent and powerful data structures.

The root node is at the top and retains the first attribute, while the leaf nodes are at the bottom and hold the class label. The branches represent the outcome of an attribute decision. Starting at the root node and working down to the leaf nodes, new data is routed through a decision tree. Each feature is assessed in a node as we advance to the leaf nodes, where we receive an instance's classification result, or class label. It is a classification tree if the target variable is categorical, and a regression tree if the target variable is continuous. A decision tree can be as simple as a set of if-then rules (logical disjunctions) in its most basic form. Decision trees resemble flowcharts and are essentially restricted graphs because they are made up of nodes and edges between them.

It is significantly more difficult to build a decision tree algorithm than it is to use one. The majority of algorithms, on the other hand, are built on the same premise. Based on what we already know, we should choose the property that provides the most information at each step. Decision trees are evaluated differently than other classifiers. In a decision tree, the sequence of the splits matters, thus we cannot use a loss function. We need to make the best splits early in the tree when developing the decision tree. We do so by assessing the amount of information gained at each phase. The impurity of the data set D, which is a training set of tuples, can be determined using the Gini index or the information entropy. The Gini index is defined as follows:

$$D = 1 - \sum_{i=1}^{c} p_i^2, \tag{3.7}$$

where p_i implies the probability that a tuple in D belongs to class C_i and c denotes the number of classes.

The Gini index, also known as the Gini coefficient, Gini impurity, or Gini ratio, is a statistical dispersion measure established in economics to describe the wealth distribution of a nation's population. The Gini index, which ranges from 0 to 1, is used to evaluate splits when developing a decision tree. For each attribute, the Gini index examines a binary split [48]. The Gini index is 0 if the separation is perfect. If the index is 1, the attributes in each of the two splits are divided evenly. As a result, we prefer splits with a low Gini index. The Gini impurity is used by the CART algorithm to construct a tree.

Support Vector Machine

A support vector machine (SVM) is a common ML approach for classification and regression problems [50]. It is a binary linear classifier that is discriminative but not probabilistic. Support vector machines are generally favoured over other ML methods, such as NNs, since they are simpler, involve less work, and can often achieve high accuracy. There are also non-linear classification versions. Support vector machines work by finding a hyperplane in a n-dimensional space that clearly divides data points into two classes, where n is the number of features. A hyperplane is a decision boundary used to classify data points in ML. We can identify two parallel hyperplanes that partition the two classes of data if the data is linearly separable.

Unlike linear regression or NNs, which include all data points, SVM only consider the training set's support vectors for determining the position of the dividing hyperplane. The support vectors are just touching the margin's edge. Finding the best hyperplane is a problem of optimization. The perpendicular distance between the support vectors and the hyperplane is used to compute the margin. The hyperplane is defined as follows:

$$w^{\mathrm{T}} x - b = 0, \tag{3.8}$$

where w denotes the weight vector, x implies the group of $x_i \in \mathbb{R}^p, i = 1, 2, \ldots, n$, points and b signifies the bias. For a given weight vector w and a bias b we can then write:

$$\begin{cases} w^{\mathrm{T}} x_i - b \geq 1, & \text{if} \quad d_i = +1 \\ w^{\mathrm{T}} x_i - b < -1, & \text{if} \quad d_i = -1, \end{cases} \tag{3.9}$$

where $d_i, i = 1, 2, \ldots, n$, denotes the margin of separation between the hyperplane and the data point x_i. The optimum hyperplane is the one with the maximum margin. The data in a real-world context is often noisy, and a hyperplane cannot properly separate it. As a result, the maximum margin criterion is modified by introducing slack variables, which are additional coefficients. Some data points may wind up on the wrong side of the hyperplane as a result of this. This is named the soft margin classifier. Adding more coefficients, however, increases complexity and necessitates more computational capacity.

k-Nearest Neighbor

The k-nearest neighbor (KNN) method is a popular classification and regression technique [51]. It is extensively used since the output is simple to read and it takes little time to compute. KNN is a slacker when it comes to learning. Learning does not occur as quickly as training instances arise, as eager learners such as support vector machines or decision trees do, but only after additional instances appear. Learners that are eager to learn

generate a generalization from test cases. KNN and other lazy learners use a distance metric to compare each new instance to existing ones, deferring learning until new instances are seen. To put it another way, learning occurs at the time of prediction. This is also known as instance-based learning, because it necessitates the deployment of test instances alongside the model. Because it employs the instances themselves rather than a rule set or decision tree inferred from the training examples, it is a whole new manner of encoding the "knowledge" from the training data. The closest existing instance's class is allocated to a new instance. The term for this is nearest-neighbor classification. The Euclidean distance is commonly employed. Because the training instances are maintained in memory, instance-based learning is also known as memory-based learning. KNN is one of the most basic ML algorithms. If it's utilized for classification, the result is a discrete value called class membership. A majority vote of its neighbors classifies the new occurrence. When used for regression, the output is the object's value, which means it predicts continuous values. KNN employs feature similarity to determine the distance between the training and the new instance.

KNN is a lazy learning algorithm that is non-parametric. The term non-parametric refers to the fact that the method makes no assumptions about the underlying data distribution. As a result, KNN is a good choice if no prior information of the underlying data distribution is available. Because real-world data does not conform to theoretical assumptions, it is a "natural" choice for many non-linear data sets. KNN is very simple to grasp and interpret while still providing high accuracy, making it an ideal starting point for a ML project. On the downside, because KNN maintains all or nearly all training data, it necessitates a lot of memory and processing capacity, particularly when k is large.

Linear Regression

Linear regression is a common method for predictive analysis [52]. It's a linear strategy to modeling the relationship between one or more explanatory or independent variables x and the dependent variable y. The following is the simplest version of the regression equations with one dependent and one independent variable:

$$y = ax + b, \tag{3.10}$$

where a denotes the slope and b the intercept of the regression curve. Note that a and b are the regression coefficients or weights. The input variable or variables x and the single output variable y are assumed to have a linear relationship in linear regression. The coefficients are learned from data using ML. To put it another way, we strive to find the best fit line for the data points we have. In ML, a variable is referred to as a feature. In ML jargon, the dependent variable is referred to as the target, predictor, or label.

An observation in simple linear regression, also known as bivariate regression, is made up of two variables: one independent variable x and one dependent variable y. The relationship between the two variables is represented by a straight line. The least square technique is used to represent the best fit for the regression line. The best fit line is the one with the smallest prediction error or residual for each data point. The vertical distance between the observer data point and the regression line is the residual. The positive and negative errors would cancel each other out if the prediction error was not squared. The following is the relationship between x and y:

$$y = ax + b + \epsilon, \tag{3.11}$$

where ϵ denotes the error term or regression residual and represents the fact that regression is an approximation and not perfectly precise. To adjust for a prediction error between the observed and predicted value, the error term is required. It's also known as a residual term, disturbance term, or remainder term, and it's formed when the model doesn't adequately reflect the actual relationship between the independent and dependent variables. It is an observable estimate that is the sum of the deviations with the regression line.

Multiple Linear Regression and Polynomial Regression

To predict the factor of interest, simple linear regression uses only one predictor variable. Because more than one independent variable influences the prediction in many real-world problems, regression analysis frequently includes more than one predictor. Multiple linear regression, also known as multivariable linear regression, is used when there are two or more independent variables [52]. By fitting a linear equation to observed data points, multiple linear regression aims to explain the linear relationship between the explanatory (independent) variables and the response (dependent) variable. The model can be defined as follows given n observations:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n + \epsilon, \tag{3.12}$$

where $\beta_i, i = 0, 1, \ldots, n$ denotes the regression coefficient. Multiple linear regression assumes the followings:
- The dependent variable and the independent variables have a linear relationship.
- There is not a lot of correlation between the independent variables (i.e. absence of multicollinearity).
- The residuals of the regression are normally distributed.

If the variables x and y have a nonlinear relationship, polynomial regression can be applied [52]. A n-th degree polynomial is used to model the connection between the dependent and independent variables y and x. The model can be defined as follows for a single predictor x:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n + \epsilon. \tag{3.13}$$

Although polynomial regression can fit a nonlinear model, it is considered a particular instance of multiple linear regression because the regression function is linear. Despite the fact that x^2 is quadratic, the unknown coefficients $\beta_0, \beta_1, \ldots, \beta_n$ are linear, and the estimator treats x^n as a separate variable. We can apply the same analysis approaches as in multiple linear regression, such as least squares.

Logistic Regression

The binary dependent variable is estimated using logistic regression (or logit regression) based on one or a set of discrete explanatory values [53]. It is a frequently used model in ML and is utilized in a variety of applications. A linear relationship between the dependent and independent variables is not required for logistic regression. When the dependent variable is binary, it is used for classification (dichotomous). The probability of an event, such as success or failure, is calculated using logistic regression. Multinomial logistic regression is used when the dependent variable contains more than two categories.

The logistic function lies at the core of logistic regression. A sigmoid curve is the logistic function, often known as the logit function. Any real number x can be converted to a value between zero and one using the logit

function. The explanatory variable x is linearly mixed with coefficient values β_0 and β_1 in logistic regression. During training, these coefficients are learned. The logit function can therefore be written like this:

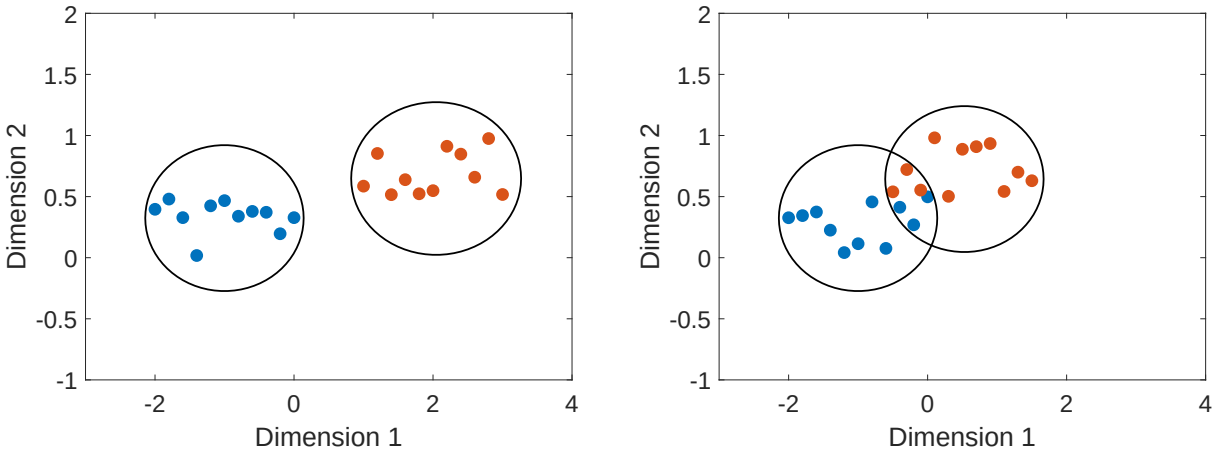
$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}, \tag{3.14}$$

where β_0 implies the intercept and β_1 the regression coefficient. p is defined as the probability of the dependent variable belonging to a class.

The coefficients are usually learned through maximum likelihood estimation in logistic regression. A frequent learning process used to optimize the coefficients for a variety of ML algorithms is maximum likelihood estimation. The likelihood function is used in maximum likelihood estimation to maximize the likelihood that the model's coefficients represent the data that is actually seen and, as a result, minimize the prediction error.

3.1.2 Unsupervised learning

When there is no labeled data, unsupervised learning methods are applied. To put it another way, there is no such thing as ground truth. Clustering is the most extensively used unsupervised method. Clustering divides data points into groups based on a similarity metric, rather than predicting a class like supervised learning does. The purpose of clustering is to analyze data by identifying the data set's underlying structure. Cluster evaluation is more difficult than supervised learner evaluation because there is no ground truth in clustering. Furthermore, it is not always clear how to evaluate a cluster algorithm's performance, and the evaluation is frequently subjective. The clusters must be applicable to the problem at hand.



(a) Exclusive clusters.

(b) Non-exclusive clusters.

Figure 3.7: Exclusive and non-exclusive clusters.

Every data point in a cluster is closer or more similar to any data object in the cluster than it is to any object in another cluster. The clusters should ideally be separated from one another. This is what it means to be well-separateness. If the distance between data items in a cluster is short, the cluster is cohesive, and the distance between data points in various clusters is considerable, the cluster is well-separated. Clusters don't have to be spherical, as seen in Fig. 3.7, but they can be of any shape. Clusters can also have several dimensions.

Centroid-based clustering

The data points in centroid-based clustering are grouped around a centroid, which is the mean of all the objects in the cluster [54]. In most cases, a centroid does not equate to a real data point. A medoid, on the other hand, must be a data point. In a cluster, a medoid is the most representative data point. When there is no relevant centroid, such as when the data is categorical rather than continuous, it is employed. The cluster is defined by the prototype in prototype-based clustering. Each data point is more similar to the prototype than any other cluster's prototype. The centroid is frequently the prototype. The prototype can be considered the most central point, and the clusters are spherical in nature.

A cluster is a densely packed area of instances bordered by a sparsely packed area. The clusters can be interwoven and the high density area might have any shape. When clusters are irregular and noise and outliers are present, density-based models are used.

k-means clustering

One of the most extensively used clustering techniques is k-means clustering. It is a prototype-based, partitional clustering algorithm. Partitional clustering means that each instance is only assigned to one cluster at a time and that no instance is left unassigned to clusters [55]. Clustering by k-means divides instances into k distinct clusters, where k is a user-defined number. A k that has been chosen incorrectly can produce undesirable outcomes.

k-means clustering is an iterative approach. The procedure begins by picking k initial centroids at random from the data set. The user must choose k, and it is not always known at the start of a clustering project. Based on its distance from the centroid, each instance of the data set is subsequently assigned to a cluster. The Euclidean distance is usually employed. The centroids are recalculated after each instance has been assigned to a cluster by determining the mean of each cluster's instances. This procedure is repeated until the centroids no longer vary. The fundamental k-means algorithm is properly specified in the diagram shown in Fig. 3.8.

In general, the k-means algorithm can be very time consuming since a large number of distance measures must be performed in each iteration to compute the similarity to a centroid.

Hierarchical clustering

Hierarchical clustering, often known as hierarchical cluster analysis (HCA), is a group of cluster algorithms that work together to create hierarchies of clusters [54]. Not only does hierarchical clustering separate the data, but it also illustrates the relationships between the clusters. Agglomerative and divisive clustering are the two most used techniques. Agglomerative clustering works from the bottom up, with each instance starting as its own cluster. Each data point is combined with one or more other points based on their similarity to build larger clusters. We end up with a single cluster including all instances in the last stage. Divisive clustering, on the other hand, is a top-down technique that works in the other direction. Clustering begins with a single cluster, which is then divided into subclusters. In subsequent cycles, the subclusters are further subdivided until each data point is in its own cluster. Once a stop criterion is fulfilled, divisive clustering does not need to continue through all iterations and can stop. Fig. 3.9 depicts the two ways. Agglomerative approaches are easier to programmatically

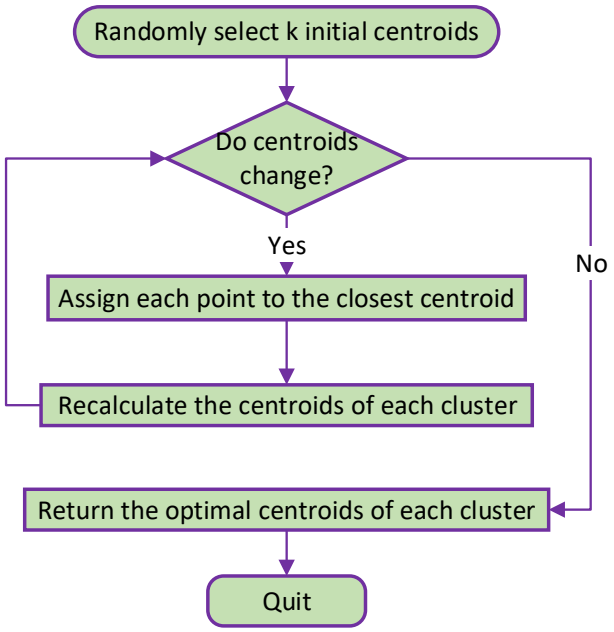


Figure 3.8: k-means algorithm.

apply, whereas divisive approaches appear to be more in line with how the human brain functions. Agglomerative clustering is more commonly utilized in practice.

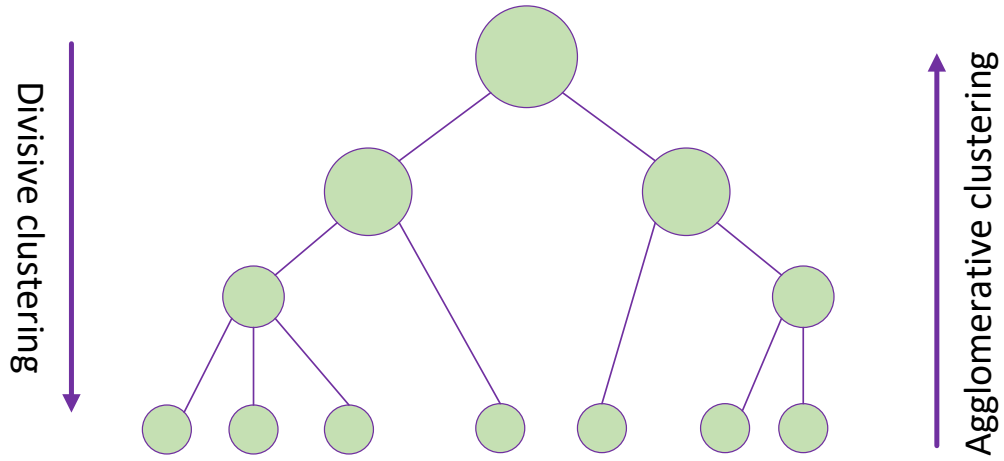


Figure 3.9: Agglomerative and divisive clustering.

Visualizing clusters

A dendrogram, or tree-like diagram, is a common graphic representation of a hierarchical cluster [54]. The clusters, their relationships, and the order in which they were fused or divided are all shown on a dendrogram. Individual clusters can be created by cutting a dendrogram at a specific level. The dendrogram, on the other hand, cannot be used to calculate the number of clusters. It frequently suggests the correct number.

Evaluation of clusters

Because there is usually no ground truth, evaluating a cluster is difficult. In an exploratory data analysis project, cluster analysis is frequently utilized. As a result, it is unclear why cluster evaluation is significant. There are various reasons why evaluating a cluster is important:

- Using different cluster analysis methodologies, different clusters may be discovered.
- It is possible that the correct number of clusters will only become apparent during cluster evaluation.
- Clusters that have been discovered are irrelevant, or clusters have been discovered despite the absence of cluster data.

If clusters have been discovered, indicating that non-random structures exist, we must establish how accurate the clustering was. If there is no ground truth, such as labeled data, intrinsic methods must be utilized. Intrinsic approaches assess clustering by looking at how well clusters are separated and how compact they are [48]. Furthermore, various types of cluster analysis methodologies necessitate various evaluation procedures. Cohesiveness (compactness, tightness) and separateness are two measurements that can be used to assess the quality of many cluster types (isolation). Cohesiveness is a measure that determines how close instances are to the centroid. Separateness measures how distinct or well separated a cluster is from other clusters. We want clusters that are both cohesive and well-separated in general. The Silhouette index is a method that combines both.

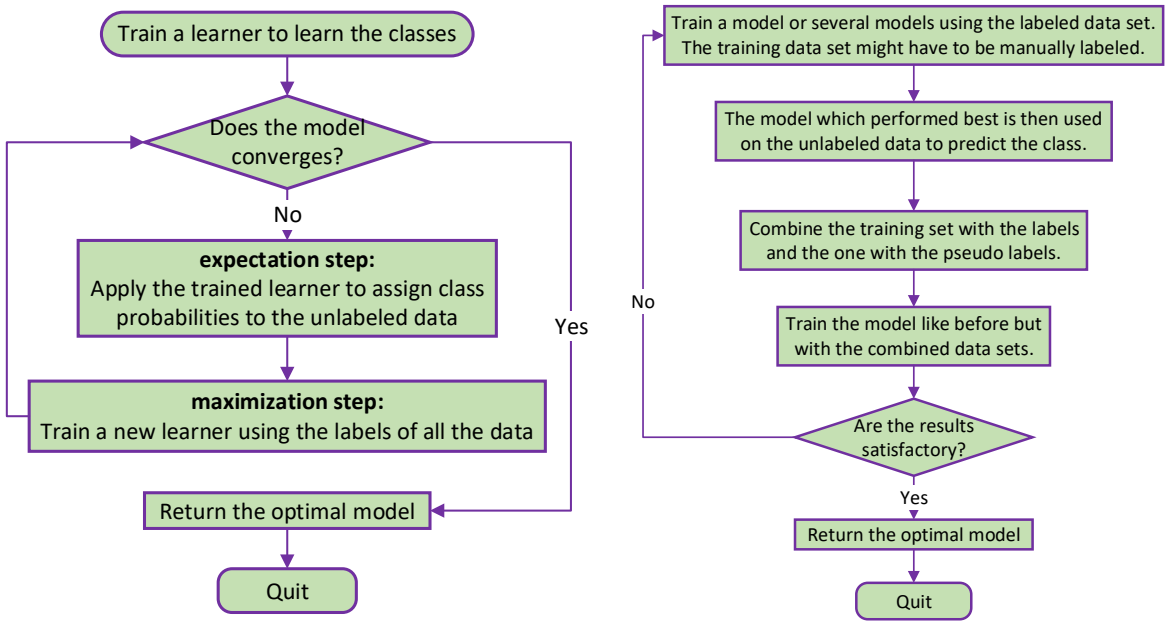
The silhouette coefficient is a measure for how similar instances are within a cluster (cohesion), compared to other clusters (separation). The silhouette coefficient is a number that goes from -1 to 1. The higher the index, the closer an instance is to items in its own cluster and the further it is from objects in other clusters. If the majority of the instances have a high index, the clustering is well segregated and cohesive. A distance measure can be used to determine the silhouette index. Only partitional clustering may benefit from the silhouette index, not hierarchical clustering.

3.1.3 Semi-supervised learning

For training, supervised methods employ labeled data, whereas unsupervised approaches use unlabeled data and must assess feature value on their own. A hybrid strategy is used in semi-supervised approaches. This method provides a number of advantages. A data scientist is frequently used to label data, which is a time-consuming and costly process. Human bias can also be introduced by manual labeling. Using unlabeled data during training can help improve accuracy, and semi-supervised algorithms generally outperform unsupervised methods. Semi-supervised approaches enable access to vast amounts of unlabeled data in situations when assigning supervision information is impractical. Instead of training the model on a small piece of manually labeled data or using an unsupervised approach on the unlabeled data, semi-supervised methods seek to enhance classification accuracy by combining both labeled and unlabeled data. However, it is not always viable to use semisupervised algorithms, because we frequently do not know the distribution of unlabeled data.

Transductive and inductive learning are the two primary semi-supervised techniques. We aim to infer the labels of the unlabeled data using labeled data in transductive learning. Inductive learning, on the other hand, aims to produce a prediction function that is defined on the full space X [56]. We strive to extract rules from labeled data that may subsequently be applied to the unlabeled data set in inductive learning.

The expectation maximization (EM) algorithm, which is described in the diagram of Fig. 3.10a, is a transduc-



(a) Expectation maximization algorithm. **(b)** Pseudo labeling algorithm.

Figure 3.10: Transductive methods.

tive approach. EM is a technique for resolving problems involving maximum likelihood estimation. In general, expectation maximization is a strategy for maximizing a likelihood function when some of the variables in the model, known as latent variables, cannot be directly observed. The assumption that the data is made up of many multivariate normal distributions is a strong one in expectation maximization. It seeks to develop an optimal model iteratively by alternating between enhancing the model and improving the item assignment to the model.

Pseudo labeling is a simple and effective semisupervised learning technique. It has been utilized in deep learning and is traditionally thought of as a transductive method. In fact, most NNs and training approaches can benefit from pseudo labeling. We learn the features from the labeled data in semi-supervised learning. We'd like to make use of the unlabeled data's information to gain a better grasp of the data's structure. To learn from unlabeled data, pseudo labeling might be utilized. The core idea behind pseudo labeling is to train a model on labeled data, then use that model to label unlabeled data. A pseudo labeling algorithm is described in the diagram of Fig. 3.10b.

3.2 Machine Learning Classification, Statistics and Evaluation

The ML classification into discriminative and generative models, as well as ML statistics and learner evaluation, are presented in this section.

3.2.1 Generative and discriminative models

Generative and discriminative models are two main categories of predictive algorithms. This distinction assumes that ML algorithms are seen from a probabilistic perspective. A generative model learns the joint prob-

ability distribution $p(x, y)$, but a discriminative model learns the conditional probability distribution $p(y|x)$, or the probability of y given x. That is, a discriminative model makes predictions based on conditional probability and is either used for classification or regression, while a generative model revolves around the distribution of a dataset to return a probability for a given problem.

As a consequence, the discriminative models are used particularly for supervised ML. They create new instances using probability estimates and maximum likelihood. However, they are not capable of generating new data points, while the ultimate goal of discriminative models is to separate one class from another. On the other hand, the generative models are capable of generating new data points. Because of this, they are used in unsupervised ML to perform tasks such as probability and likelihood estimation, modelling data points, and distinguishing between classes using these probabilities. In addition, because discriminative models handle a problem immediately, they perform better than generative models, which require an intermediate step. Generative models, on the other hand, converge far more quickly.

3.2.2 Machine Learning and Statistics

ML makes extensive use of statistics. As a result, statistical and ML concepts are frequently used interchangeably in the literature. The phrases independent and dependent variable, for example, are statistical terminology, while feature and label are ML counterparts, respectively. Some popular ML concepts and their comparable statistics phrases are opposed in Tab. 3.1.

Table 3.1: Machine learning and statistical terminology.

Statistics	Machine Learning
Fitting	Learning
Classification, Regression	Supervised Learning
Clustering, Density estimation	Unsupervised Learning
Independent variable, covariate	Feature
Dependent variable, Response	Label
Observation	Instance

The terms statistical learning (SL) and ML are frequently confused. There is a distinction between the two approaches. Both strategies are based on data. ML, however, uses data to increase a system's performance. The foundation of SL is a conjecture. The data in SL are assumed to share some qualities with a degree of regularity. As a result, SL can utilize probabilistic approaches to describe statistical regularity, and probabilistic distributions can be used to characterize statistical regularity. ML is not dependent on speculation. SL requires a lot of arithmetic and normally works with smaller data sets with fewer features, whereas ML can work with billions of occurrences and features. SL necessitates a thorough comprehension of the data, but ML can iteratively uncover patterns with far less human effort. As a result, ML is frequently chosen over SL. SL, however, helps us acquire confidence in our model by requiring us to acquaint ourselves with the data.

ML is frequently criticized for being little more than glorified statistics. While ML does employ a lot of statistics, it also uses other concepts like data mining and neurocomputing, making it a subfield of computer science. Statistics, is usually considered as a subfield of mathematics. Both ML and SL are data-driven learning algorithms.

ML, however, is an algorithm that learns from data without the need for a developer to program it. In the form of equations, SL is a mathematical formalization of relationships between variables.

3.2.3 Evaluation of learner

Learning is the process of enhancing one's performance based on previous experience. We must evaluate the learner's performance during training in order to ensure that the learner progresses. We also need to estimate how well a trained learner will do on new data that has not been seen before. A trained learner must capture the signal in the training set properly while also generalizing adequately to fresh data. The training error can be used to quantify the learner's accuracy, but it cannot be used to forecast how well the learner would perform on new data. Furthermore, because it does not appropriately account for model complexity, the training error is not a reasonable approximation of test error [57].

We frequently use labeled data that was not used for training to assess how well a supervised ML scheme has learned. We usually divide the data into three sets: one for training, another for evaluating, and yet another for testing the learner. The training data set is used to train the learning algorithm, while the validation data set is used to optimize the learning algorithm's parameters, and the test data set is used to calculate the true error rate on the final, trained method. To achieve a reliable estimate of the error rate, each data set must be independent. Typically, data sets are separated into three groups at random. Each data collection must be representative, with no feature being overrepresented or underrepresented in any one set. The ratio of sets is determined by the problem and the amount of data available. Commonly, the largest amount is used for training and we may use proportions. The disadvantage is that we will be limited in the amount of data we can use for training. Techniques like k-fold cross-validation can be employed if there is a lack of data.

There are numerous possible metrics for evaluating a predictor's success, and not all of them are appropriate for all learners. Some are better suited to classification, while others may be better suited to regression analysis.

Accuracy

Accuracy is one statistic for evaluating classification models. The accuracy of a model is simply the fraction of correctly classified predictions. Accuracy is defined as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}. \tag{3.15}$$

Nevertheless, accuracy does not reveal the whole story. The number of misclassifications must be taken into account. When we have a class-imbalanced data set, accuracy performs very poorly. In real-world challenges, however, unbalanced data sets are the norm.

Precision, Recall and F-score

Precision and recall take both correctly and incorrectly classified cases into account. True positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) are the terminology we use. The classifier's prediction is referred to as positive or negative. False positives and false negatives are wrongly classified situations, whereas genuine positives and true negatives are accurately classified.

Precision, also known as positive predictive value, is the proportion of correct positive classifications in a classification task. Precision can be defined as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{3.16}$$

The proportion of actual positives correctly identified is known as recall, also known as true positive rate or sensitivity. Recall is defined as follows:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{3.17}$$

To completely assess a classification task's outcome, we must look at both precision and recall. In practice, increasing precision often leads to a decrease in recall, and vice versa. As a result, comparing models with high precision and low recall, or vice versa, is challenging.

The F-score, or F-measure, is a metric for categorization task accuracy that combines both precision and recall. The F-score is defined as the harmonic mean of precision and recall:

$$\text{F-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{3.18}$$

An F-score ranges from 1 to 0, with 1 indicating flawless precision and recall and 0 indicating the worst case. Because the F1-score considers both precision and recall, comparing models with high precision and low recall, or vice versa, is relevant. In practice, the F-score is frequently employed in natural language processing tasks or to assess the classification performance of search results.

Confusion Matrix

The usage of a confusion matrix, a type of contingency table, is a frequent means of showing the results of a classification task. The projected values are represented by the rows of the confusion matrix, while the actual values are represented by the columns. Depending on the number of labels, a confusion matrix can have two or more columns. The name comes from the fact that a confusion matrix makes it simple to see if the classifier is confusing two classes. Because it comprises all findings, a confusion matrix provides for a more complete study of a classification result than simply using proportions of classifications and misclassifications.

Receiver Operating Characteristic

The receiver operating characteristic (ROC) is a graph of the true positive rate (also known as sensitivity or recall) in function of the false positive rate (also known as specificity). It is used to measure a binary classifier's performance at various threshold settings. The false positive rate is (1−specificity), where specificity is defined as follows:

$$\text{specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}. \tag{3.19}$$

The false positive rate (FPR) is then defined as follows:

$$\text{FPR} = 1 - \text{specificity} = \frac{\text{FP}}{\text{TN} + \text{FP}}. \tag{3.20}$$

The receiver operating characteristic curve of a perfect classifier will move up straight along the y-axis until it reaches $y = 1$, then along the x-axis (100% sensitivity, 100% specificity). The values are entirely random if the receiver operating characteristic curve is diagonal. A sensitivity/specificity pair corresponding to a specific choice threshold is represented by each point on the receiver operating characteristic curve.

The receiver operating characteristic curve allows us to assess a classifier's performance over its whole operating range. The receiver operating characteristic curve is frequently used in ML to compare statistical models. The performance of two or more classifiers at one point can be compared using a single threshold, or the entire curve can be compared. A receiver operating characteristic curve of less than 0.5 may suggest that the classifier identifies relationships that are the polar opposite of what we expect, indicating that the experiment was set up incorrectly.

3.3 Deep Learning

Deep learning, a technique based on ANNs, has emerged as a potent tool for ML in recent years, promising to transform the future of artificial intelligence [58]. To achieve cutting-edge outcomes, deep learners rely on enormous labeled training data sets. Deep learning has been used for image classification, machine translation, and speech recognition, and it can be found in many modern smart phones for voice commands and face recognition. They are also utilized in self-driving cars, handwriting recognition, drone reconnaissance, and coloring black-and-white photos and films, to mention a few. Despite their widespread success, they are not fundamentally different from regular ANNs in their operation. They frequently use rectifiers as AFs, rather than sigmoid curves, as opposed to earlier networks like the MLP, and they are trained using BP and gradient descent, i.e. stochastic gradient descent. However, because we have to deal with high dimensionality and a huge number of features, the computations might be quite resource intensive due to their massive size. Training data is frequently scarce, and obtaining labeled training data sets in the proper formats might be prohibitively expensive.

3.3.1 Deep learning preliminaries

When a learner is labeled a deep learner or a shallow learner, there is no agreed-upon definition in the literature. Deep learners are essentially multilayered ANNs. Deep learners are typically NNs with more than one hidden layer. Deep learners can be used to learn in a supervised, semi-supervised, or uncontrolled environment. Because of its influence on computer vision, natural language processing, and bioinformatics, deep learners have grown highly popular in recent years. Deep learners have the ability to automatically extract features, which is an important feature. By traveling through the layers of a deep NN, low-level features are abstracted into higher-level features.

Deep learners are classified as representation learners. Prior to producing the final prediction, representation learners generate an internal representation of the characteristics. Before translating features to the final prediction, deep learners turn them into several representations.

Deep learners, as previously stated, are similar to shallow NNs. Deep learners, however, frequently use a different form of AF. The rectified linear unit, or ReLU, is the most popular AF used in deep NNs. It's a simplistic

function that returns 0 for negative input and a value for positive x. It can be written as follows:

$$f(x) = \max(0, x). \tag{3.21}$$

A ReLU is a ramp function that, despite its simplicity, is remarkably good at handling non-linear and interaction effects. The link between the independent variables and the predictions is not linear in non-linear effects. Interaction effects occur when a variable a has a varying effect on a prediction based on variable b. ReLUs can be combined in a variety of ways, which is why they perform so well for non-linearities. Deep learners are large functions in essence. The ReLU has several variations. The softplus function, for example, is a smooth approximation of the ReLU and can be represented as:

$$f(x) = \log(1 + e^x). \tag{3.22}$$

The logistic function, which is also an approximation of the ReLU's derivative, is the softplus function's derivative.

There are various advantages to using ReLUs. They are one-sided and do not produce any negative outcomes, which is more biologically reasonable. They are also quick to compute because they just do addition, multiplication, and comparison operations. Rectifiers have replaced sigmoid functions as the AF of choice for deep learners because they allow for faster and more efficient training even on huge networks with big data sets. They also don't require any semi-supervised pre-processing. Even though they can benefit from semi-supervised setups with extra-unlabeled data, deep rectifier networks can achieve their highest performance on completely supervised tasks with huge labeled datasets without any unsupervised pre-training [59].

When utilizing deep learners, there are a few things to keep in mind. A large amount of labeled training data is required to train a deep learner. Finding the best deep learning architecture can be time-consuming. Overfitting and convergence can cause issues, which is exacerbated by the number of layers in the network. During training, neurons in deep learners might "die", which means they always output the same value for any input. Units that have died are unlikely to resurrect and will not participate in the trained network. When rectifiers are employed as an AF, this is most common. Another difficulty is the problem of vanishing gradients. When utilizing BP during training, the derivative of the error function can become tiny, leading the weights to stop changing. In the worst-case scenario, the network ceases to learn at all. For deep learning, these and other challenges necessitate the use of supplemental learning strategies.

Feature learning is a set of techniques that allows a machine to detect features automatically rather than requiring a feature engineer to manually extract the characteristics of interest. Feature learning is part of an attempt termed automated ML to automate the entire end-to-end ML process. Automated ML attempts to automate all ML operations that are now done manually, such as model selection and hyperparameter estimation. Automatic feature extraction, however, is not always possible. Feature learning has been applied with shallow learners, such as k-means clustering and principal component analysis, and is not limited to deep learning.

3.3.2 Convolutional neural networks

Convolutional NNs (CNNs), also known as ConvNets, have grown in popularity as a tool for image processing. They were built specifically for image analysis and are inspired by the human visual cortex and how it assimilates

visual information. Yan LeCun suggested one of the first CNNs in 1998 [60]. The first convolutional network, however, was proposed in the 1980s, and many different CNN topologies have since been presented.

CNN's architecture reflects some of the image processing qualities because they use images as input. CNN is made up of a series of interconnected feedforward layers that implement convolutional filters, followed by reduction, rectification, or pooling layers. A CNN, unlike the MLP, is not entirely connected. In addition, CNNs have three-dimensional neurons that take a three-dimensional fixed-size input image. A single vector is returned as the result. They are otherwise fairly similar to traditional NNs.

In essence, a CNN is made up of five layers: an input layer, an output layer, and three types of intermediate layers: convolutional layers, pooling layers, and non-linearity layers, the latter of which is usually a ReLU. A typical process goes through the following steps:

1. The input layer accepts an image, which is a three-dimensional pixel array with a fixed size: [width × hight × color channel]
2. The convolutional layer captures spatial characteristics from small areas of the image and relates them to one unit in the following layer.
3. The pooling layer downsamples the image by combining nearby features into single units.
4. The AF is applied by the ReLU layer, but the image size remains unaffected.
5. The output layer, which takes the form of a fully linked MLP, calculates the class score and outputs a vector containing it.

The success of CNN's application to computer vision problems is largely due to them. They are, however, employed for natural language processing, video analysis, and medical analysis as well. They have also been taught how to play board games like checkers and Go. Despite CNN's appealing properties and the relative efficiency of their local architecture, they have remained prohibitively expensive to apply to high-resolution images on a broad scale [61].

3.3.3 Recurrent neural networks

MLPs and CNN have the drawback of requiring a fixed-sized vector as input and producing a fixed-sized output vector. Traditional NNs also have independent inputs. Recurrent NNs (RNNs) can produce sequences of vectors as output from sequences of values $x_1, \ldots, x_n$. They are useful for applications when the input size isn't known ahead of time because they can process temporal sequences. RNN's capacity to analyze sequences of inputs makes it ideal for jobs like time series analysis, speech recognition, and sentiment analysis with varying input sizes. Most recurrent networks can handle variable-length sequences [62].

John Hopfiled completed the initial research on RNN, which was published in 1982 [63]. Elman networks, gated recurrent units, bidirectional RNNs, and deep RNNs are only a few of the RNN types that have been developed since then. RNNs, unlike feedforward networks, have directed cycles. RNNs and CNNs, however, are rarely employed in isolation, but rather as part of a larger architecture, frequently in conjunction with an MLP.

Loops are added to the network architecture by RNN. In an RNN, a neuron may send a signal to another neuron in the same layer or have a connection with itself in addition to a connection with a neuron in the next layer. The network's output sequence could potentially be used as feedback with the next input sequence. Some alternative RNN architectures are shown in Fig. 3.11.

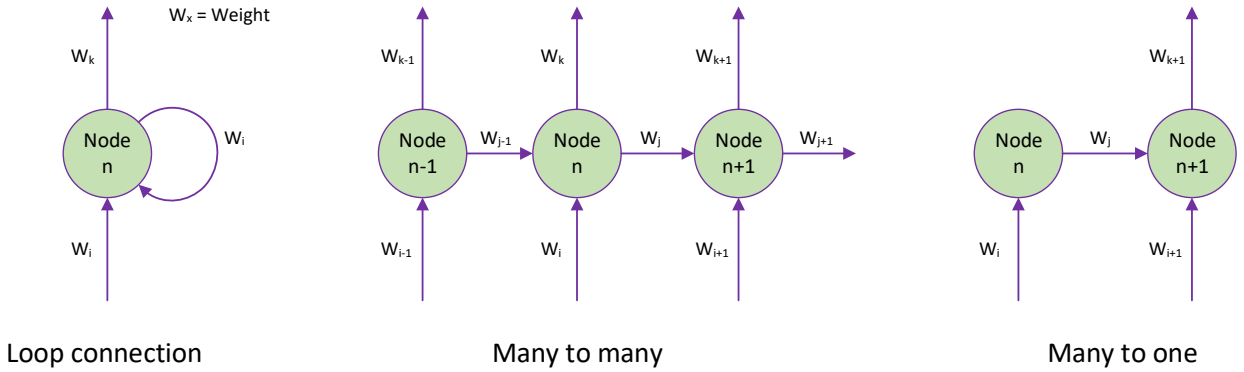


Figure 3.11: Architectures of RNNs.

The network gains memory or state as a result of the recurrent connections. The state is usually saved as a state vector that is controlled by the network. To create a new state vector, the state vector is joined with the input using a learnt function. In this way, a RNN considers not just the present input but also what it has learned in the past. Time step $t-1$ influences the decision at time step t. This makes sense because we make decisions based on our previous experiences. The sequential information is stored in a hidden state, allowing the RNN to uncover long-term dependencies, or correlations over time. The hidden state h at time t can be computed as follows:

$$h_t = f(Wx_t + Uh_{t-1}), \tag{3.23}$$

where h_t and x_t denote the hidden state and input vector, respectively, at time step t. Furthermore, W and U denote the weight and transition matrix, respectively, while $f(\cdot)$ implies the AF.

The memory of the RNN is represented by the transition matrix U, which is the hidden-state-to-hidden-state matrix. The hidden state, not simply the most recent, contains remnants of all prior hidden states. In the same manner that other types of networks do, the weight matrix W defines how much attention to give to the input. A logistic or hyperbolic tangent function is commonly used as AF.

A deep learner is not always an RNN. A recurrent network is also arranged into layers, as seen in Fig. 3.11, and an RNN is commonly referred to as a deep RNN if it includes more than one hidden layer.

A RNN, like a feedforward network, is trained through BP and gradient descent. The BP process, however, does not work in the same way as a feedforward network because of the loops. The algorithm in a feedforward network moves backwards through the layers from the final error and adjusts the weights up or down, whichever decreases the error. BP through time (BPTT) is a BP extension that is used to train a recurrent network. BPTT is a gradient-based recurrent network training approach. In BPTT, time is defined by an ordered set of calculations moving from one time step to the next. In essence, the RNN's structure is unraveled.

Not all RNNs, however, can be unraveled. BPTT is not possible with some architectures. There are also several types of BPTT, such as truncated BPTT, which is a simplified version of full BPTT. When there are many time steps and computation is sluggish, truncated BPTT is utilized. RNNs are also hampered by the vanishing gradient problem, especially when there are several layers. That is, if the gradient cannot be determined, we have no idea how to alter the weights in a way that reduces the error. In the worst-case scenario, the network ceases to learn at all.

3.3.4 Restricted Boltzmann machines

Restricted Boltzmann machines (RBMs) are shallow, two-layer networks that can be used for classification, regression, dimensionality reduction, collaborative filtering, and feature learning, and they are the building blocks of deep belief networks. RBMs outperformed the competition in the Netflix prize [64], achieving state-of-the-art performance.

RBMs are probabilistic, which means they assign probabilities rather than discrete values. A Boltzmann machine is a form of RBM. RBMs do not allow intralayer connections between hidden units, but Boltzmann machines are recurrent networks. This is why they are referred to as "restricted".

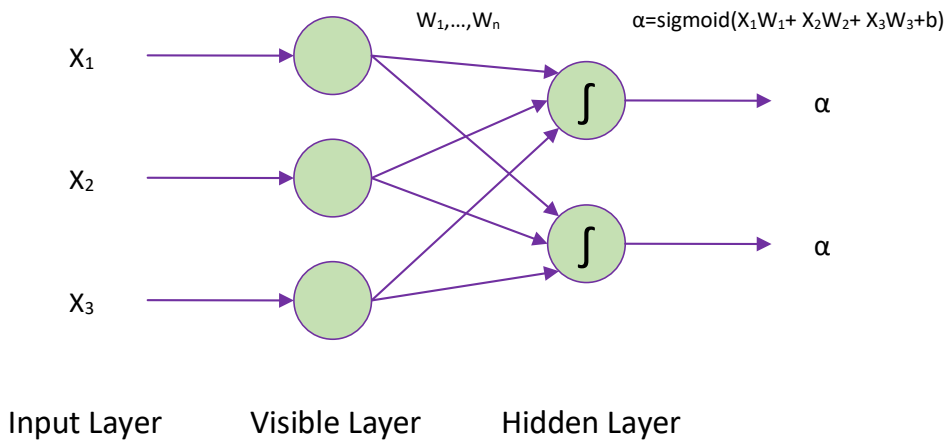


Figure 3.12: Restricted Boltzmann machine.

As demonstrated in Fig.3.12, RBMs have only two layers: an input layer and a concealed layer. They don't have a layer for output. In contrast to traditional feedforward networks, the predictions are created in a unique way. They can be classified as energy-based models. Energy-based models attempt to minimize an energy function in which a high energy value equates to poor accuracy. There are two types of RBMs: visible and hidden. The training set is given to the visible units. During training, the energy function is minimized by modifying the weights, biases, and states of the visible and hidden units until it reaches a minimum, a process known as simulated annealing. Every node that can be seen is linked to every node that can be concealed. The inputs are weighted and then added to a bias. A sigmoid AF is then applied to the combined inputs. To update the weights, RBMs are trained using Gibbs sampling and contrastive divergence [65]. If there are many layers, the output α of the hidden layer is utilized as the input for the next hidden layer.

In an unsupervised manner, RBMs learn how to recreate the input. In a backwards pass, the activations α are used as input. They are multiplied by the same weights that were used to multiply the input x. The summed products are applied to the visible layer's bias. The visible layer's biases are distinct from the hidden layer's biases. The output is a reconstruction or approximation of the input. The RBM determines the probability of the output given a weighted $x : p(\alpha|x; W)$ during the forward pass. The RBM assesses the probability of the inputs x given the weighted activations $\alpha : p(x|\alpha; W)$ during the backwards pass. $p(x, \alpha)$ is the combined probability distribution of x and α as a result of both estimations. Several forward and backward passes are made during

the training. The weights are modified until the original input's probability distribution $p(x)$ and the estimated input's probability distribution $r(x)$ converge.

The activations of the hidden layer become the input of the next hidden layer once the RBM training is done. The input from the previous hidden layer is approximated by the second hidden layer, which is trained until it can approximate it. This step-by-step technique to training hidden layers in an RBM is one of the most popular deep learning strategies because it allows for the efficient training of a large number of hidden layers. It's unsupervised because it doesn't need labeled data to train. By constructing a feature hierarchy, this successive creation of a multilayered network allows for the learning of more complicated representations. A deep belief network is built on top of a pre-trained multilayered RBM.

3.3.5 Deep belief networks

A deep belief network (DBN) is made up of a series of RBMs. It is a form of deep learner with numerous hidden layers that connect to previous and subsequent levels but not within layers. It can learn to recreate its input in the same way that an RBM can. Unsupervised training is possible thanks to the layered design. A sort of unsupervised pre-trained network is the pre-trained DBN. Each layer performs the function of a feature detector. After the first training, it can be further trained in a supervised manner to generate a classifier. The DBN could then be finished with a classification layer, such as a softmax or logit layer. A greedy DBN can be trained, with each layer being taught separately. For the next hidden layer, each hidden layer becomes the visible layer. As data, a DBN receives binary input. Continuous decimal values are also accepted through an addition called continuous DBNs.

3.3.6 Deep autoencoders

An autoencoder is a network that learns an unsupervised representation (encoding) of a data collection. An autoencoder compresses data into a code, then decompresses it into something that is near to the original input. In this way, it is taught to ignore data noise. The autoencoder learns to compress (encode) and reconstruct (decompress) the input. They're commonly utilized to reduce dimensionality. As a generative model, autoencoders can be employed. The autoencoder can use the learnt codes to create an image of something it has never seen before.

An autoencoder is a feedforward network having an input, output, and one hidden layer in its most basic form. The number of neurons in the input and output layers is the same. An input is always encoded and decoded by an autoencoder. Autoencoders attempt to learn a function h such that $h_{W,b}(x) \approx x$, with W representing weights and b representing bias. Unsupervised autoencoders are taught by minimizing a loss function such as the squared error. An autoencoder can learn the identity function and reconstruct the input similarly if the hidden layers are greater than the input layer.

Two symmetrical DBNs make up deep autoencoders. They have the ability to compress data. They can also be used to model a topic. The method of determining the topic or themes in a batch of documents is known as topic modeling. One encoding part and one symmetrical decoding part make up deep autoencoders. Each component is made up of one or more DBNs. An autoencoder accepts binary data as input, but it can also accept data with real values.

Autoencoders can be regarded of as a subset of feedforward networks, and they can be trained using the same methods, most commonly minibatch gradient descent with BP gradients [62]. Learning takes place by minimizing a loss function L that can be described as follows:

$$L(x, g(f(x))), \tag{3.24}$$

by penalizing $g(f(x))$ for being dissimilar to x. When employing an autoencoder, we are usually more interested in the qualities of the compressed representation than in the output.

3.4 Weights Direct Determination of Feedforward Neural Networks

ANN has been extensively studied and applied in a variety of scientific and engineering domains, particularly using error BP training techniques. The feedforward NN based on the error BP training algorithm or its variants is one of the most important and popular feedforward NN models, with numerous theoretical analyses and real-world applications. BP algorithms are gradient-based iterative approaches that alter the artificial NN weights in a gradient-based descent direction to bring the input/output behavior into a desired mapping. BP-type NNs, in particular, appear to have the following weaknesses:

- the probability of being trapped in some local minima;
- difficulty selecting suitable learning rates (or, say, speed of training);
- inability to design the optimal or smallest NN structure in a deterministic manner (or, say, high computational complexity).

Many improved BP-type algorithms have been developed and explored as a result of the foregoing inherent weaknesses. There are two types of improvements in general. On the one hand, the normal gradient-descent method could be used to improve BP-type algorithms. Numerical optimization approaches, however, might be used to train a NN model. It is worth noting that in order to increase the performance of BP-type NNs, many researchers focus on the learning algorithm itself [66]. Almost majority of the improved BP-type algorithms, on the other hand, have yet to overcome the aforementioned fundamental weaknesses [67].

The aforementioned weaknesses of multi-input BP-type NNs can be overcome and NN performance improved by focusing on the use of various AFs and determining the optimal NN structure. Note that three major issues must be resolved during the design of a NN model for any application:

- the AF;
- the number of hidden-layer neurons (or, say, the structure);
- the computation of connecting weights between two separate layers.

According to the previous analysis, obtaining the optimal connecting weights and the optimal number of hidden-layer neurons for the multi-input NN are useful and important, especially in the general multi-input NNs, because they can considerably reduce the computational complexity and promote hardware realization. That is, they improve the efficiency of the NNs [68]. To overcome the problems produced by the BP algorithms and to specify

the optimal NN structure for better practical applications, several weights-and-structure-determination (WASD) algorithms are introduced in [68] as better alternatives. The WASD algorithm, which employs the weights-direct-determination (WDD) subalgorithm, specifies the optimal connecting weights between hidden layer and output layer directly (i.e., just in one step), and, at the same time, finds the optimal structure of the NN during the training process.

3.4.1 Time-series Forecasting

In [69], employing a novel multi-function activated WASD for time-series (MAWTS) algorithm, a 3-layer feed-forward multi-input MAWTS-based NN (MAWTSNN) model is proposed for handling time-series modeling and forecasting problems. The MAWTS algorithm employs four AFs to determine the MAWTSNN's optimal weights and structure, while employing cross-validation to avoid overfitting. More precisely, the MA-WASD algorithm finds and keeps only the powers of the AFs that reduce the model's error during the validation process. This reduces the computational complexity even more than conventional WASD algorithms, which may need a significant number of hidden-layer neurons, while cross-validation during the training phase improves the accuracy of anticipated results even further. Note that, each one of the optimal powers may have a different AF from the others. As a consequence, during the training phase, the error produced decreases even more than if only one AF was applied. The similarities between the MA-WASD and the other WASD algorithms for handling time-series modeling and forecasting problems in [68, 70] are that they embody the WDD process and they are liable for training the NN model, while the differences are the multiple power AFs, and the cross-validation during the training process.

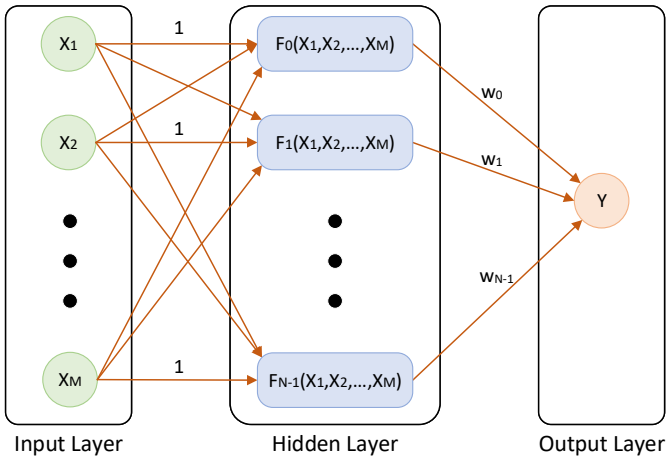


Figure 3.13: Structure of the multi-input MAWTSNN Model.

This subsection introduces a 3-layer feed-forward NN model with M input and N hidden layer neurons, as shown in Fig. 3.13. Particularly, the input layer receives and distributes input X to the relevant neuron in the next layer with equal weight 1. The input layer's values $X_1, X_2, \ldots, X_M$ are received from the middle layer, which has at most N in number power activated neurons and the corresponding AFs $F_{d+1}(X_1, X_2, \ldots, X_M)$, $d \in [0, n-1] \subseteq \mathbb{Z}$ are power activated functions. Last, the third layer is the output layer and has one linear activated neuron. The weight vector W is made up of the weights W_i in the neuron-to-neuron interaction between the second and

third layers' neurons. It is worth mentioning that the weights W_i are found through the WDD algorithm. The corresponding NN model is called MAWTSNN and its main features are its multiple AFs employment and its minimum hidden layer usage. These features can be achieved by a novel MAWTS algorithm, specially designed to train the MAWTSNN model.

WDD Process

Machine learning is a heavy computational process, and the measurement of the training error is not a simple task. In order to accelerate this process and clarify the network structure, the WASD algorithm for NN training is introduced [68].

Comprehensive outlines of main theoretical underpinnings and analyses are provided in this subsection for the construction of the MAWTSNN. The Taylor polynomial (TP) theorem [71] is given below.

Theorem 3.4.1 *Assume that the target function $\Phi(\cdot)$ has continuous derivatives of $(K+1)$-order, $K \geq 0, K \in \mathbb{Z}$, then for $x \in [a, b]$,*

$$\Phi(x) = P_K(x) + R_K(x), \tag{3.25}$$

where $P_K(x)$ is a polynomial employed to approximate $\Phi(x)$ while $R_K(x)$ is the error term.

For a fixed value $r \in [a, b]$, $\Phi(x)$ may be approximated as below:

$$\Phi(x) \approx P_K(x) = \sum_{i=0}^{K} \frac{\Phi^{(i)}(r)}{i!}(x-r)^i, \tag{3.26}$$

where $\Phi^{(i)}(r)$ signifies the value of the i-order derivative of $\Phi(x)$ at the point r and $i!$ signifies the factorial of i. It is worth noting that $P_K(x)$ is the K-order TP of function $\Phi(x)$.

Theorem 3.4.2 *The TP approximation theorem may be used to approximate multivariable functions [71]. For a target function $\Phi(x_1, x_2, \ldots, x_g)$ with $(K+1)$-order continuous partial derivatives in an origin's neighborhood $(0, \ldots, 0)$ and g variables, the K-order TP $P_K(x_1, x_2, \ldots, x_g)$ about the origin is:*

$$P_K(x_1, x_2, \ldots, x_g) = \sum_{i=0}^{K} \sum_{i_1+\cdots+i_g=i} \frac{x_1 \cdots x_g}{i_1 \cdots i_g} \left(\frac{\partial^{i_1+\cdots+i_g} \Phi(0, \cdots, 0)}{\partial x_1^{i_1} \cdots \partial x_g^{i_g}} \right), \tag{3.27}$$

where $i_1, i_2, \ldots, i_g$ are nonnegative integers.

The relationship among the NN's input $X_1, X_2, \ldots, X_M$ and the output target Y may be circumscribed with the next nonlinear function:

$$Y = \Phi(X_1, X_2, \ldots, X_M). \tag{3.28}$$

Because we are dealing with time-series, an approach similar to the NNs in [68, 70], which is inline with the K-order TP, is employed. Considering that $y_t, y_{t-1}, y_{t-2}, \ldots, y_{t-M}$ is a time-series, where $y_{t-1}, y_{t-2}, \ldots, y_{t-M}$ corresponds to the NNs variables $X_1, X_2, \ldots, X_M$, respectively, and y_t corresponds to the NN's output target Y,

the NN model performs a nonlinear functional mapping from the past observations to the future value y_t. That is, setting

$$y_t = \Phi(y_{t-1}, y_{t-2}, \ldots, y_{t-M}), \tag{3.29}$$

with $t \geq 1$ variables and continuous partial derivatives of $(K+1)$-order in an origin's neighborhood $(0, \ldots, 0)$, the K-order TP $P_K(y_{t-1}, y_{t-2}, \ldots, y_{t-M})$ may map (3.29) as below:

$$P_K(y_{t-1}, y_{t-2}, \ldots, y_{t-M}) = \sum_{v=0}^{N-1} q_v w_v, \tag{3.30}$$

where $q_v = F_v(y_{t-1}, y_{t-2}, \ldots, y_{t-M}) \in \mathbb{R}^{1\times M}$ with $F_v(\cdot)$, $v = 0, 1, \ldots, N-1$, implying a power AF of all inputs, and $w_v \in \mathbb{R}^M$ signifies the weight for q_v.

Moreover, for a given number of samples $s \in \mathbb{N}$, we set $q_{s,v} = F_v(y_{t-1-s}, y_{t-2-s}, \ldots, y_{t-M-s}) \in \mathbb{R}^{s\times M}$ and, as a consequence, the input-activation matrix becomes

$$Q = \begin{bmatrix} q_{1,0} & q_{1,1} & \cdots & q_{1,N-1} \\ q_{2,0} & q_{2,1} & \cdots & q_{2,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ q_{s,0} & q_{s,1} & \cdots & q_{s,N-1} \end{bmatrix} \in \mathbb{R}^{s\times MN}, \tag{3.31}$$

the weight vector $W = [w_0, w_1, \ldots, w_{N-1}] \in \mathbb{R}^{MN\times s}$ and the desired-output vector $Y = [y_t, y_{t-1}, \ldots, y_{t-s}] \in \mathbb{R}^s$. Considering that and opposed to the iterative weight training in traditional NN, the MAWTSNN's weights are found through the WDD algorithm [71]. In line with the NN of Fig. 3.13, WDD calculates the optimum weights from hidden-layer to output-layer neurons using the following process:

$$W = (Q^{\mathrm{T}}Q)^{-1}Q^{\mathrm{T}}Y = Q^{\dagger}Y, \tag{3.32}$$

where the operators $(\cdot)^{-1}$ and $(\cdot)^{\dagger}$, respectively, signify inversion and pseudo-inversion. To put it another way, the weights are obtained using a direct pseudo-inverse procedure.

Several AFs, such as Chebyshev polynomials, Euler polynomials, sine, square wave, are employed on NNs in [72–74]. In the case of the second layer of the MAWTSNN, we employ the following four AFs:

$$\begin{cases} \textbf{(AF1)}\ \text{Power:} \quad F_v(X) = X^{\odot v} \in (-\infty, \infty) \\ \textbf{(AF2)}\ \text{Power sigmoid:} \quad F_v(X) = e^{\odot X^{\odot v}} \odot / (e^{\odot X^{\odot v}} + 1) \in \left[\frac{1}{2}, 1\right) \\ \textbf{(AF3)}\ \text{Power inverse exponential:} \quad F_v(X) = e^{\odot -X^{\odot v}} \in (0, 1) \\ \textbf{(AF4)}\ \text{Power softplus:} \quad F_v(X) = \ln(1 + e^{\odot X^{\odot v}}) \in (0, \infty) \end{cases} \tag{3.33}$$

where X denotes the function input, $v \in \mathbb{Z}^+$ implies the power number, $\odot$ and superscript $()^{\odot}$ signify the Hadamard (or element-wise) product and the Hadamard exponential, respectively. Note that $\mathbb{Z}^+$ denotes the nonnegative integers. It is worth noting that these four power AFs have different output range. Moreover, since the WDD process is based on the K-order Taylor-polynomial, the WASD-based NNs use only power type AFs. Keeping this

in mind, standard NN AFs can be easily converted to power AFs for use in the WDD process while maintaining their mathematical properties by simply adding a power in their dependent variable. The aforementioned conversion is used in the proposed power sigmoid, power inverse exponential, and power softplus AFs, which are based on the standard NN AFs sigmoid, Gaussian, and softplus, respectively. This allows a WASD-based NN to exploit the properties of the standard AFs. Since each AF has its own empirical performance and mathematical properties, implementing multiple AFs in a WASD algorithm instead of only one will improve the NN's performance and accuracy. As a result, the hidden-layer neurons of the K-order TP NN employ four different power AFs to produce different activation in each hidden-layer neuron.

The MAWTS Algorithm

This section introduces the MAWTS algorithm, which is in charge of training the NN model. The MAWTS algorithm is described in detail throughout this section, and the entire process is depicted in Fig. 3.14. Assuming that $X \in \mathbb{R}^G$ is a time-series and by $X_{\min}$ and $X_{\max}$, respectively, we denote the minimum and maximum values X, then we normalize X to a range of $[-0.5, -0.25]$ as follows:

$$X_{\text{nor}} = \frac{X - X_{\min}}{4(X_{\max} - X_{\min})} - \frac{1}{2}. \tag{3.34}$$

It is worth noting that the MAWTSNN can deal with over-fitting in this way. Furthermore, we set the parameter $p \in (0, 1] \subseteq \mathbb{R}$, which denotes the exact percentage among the fitting and the validation set, X_{tr} and Y_{tr}, respectively. Note that this is the well-known cross-validation method, which is employed to evaluate the consistency of a machine learning model employing the validation data, as part of training. Since the validation set is disjointed from the training set, validation helps to secure that the model generalizes beyond the set of training. Setting r the round number of the product pG, we have $Y_{\text{tr}} = X_{\text{nor}}(r+1:G) \in \mathbb{R}^K$, where $K = G - r$.

Then, for the delays number $M = 1, 2, \ldots, r/3$, the MAWTS algorithm repeats the following 7 steps.

Step 1: By rearranging X_{nor}, it creates the fitting set as follows:

$$X_{\text{tr}} = \begin{bmatrix} X_{\text{nor}}(r-M+1) & X_{\text{nor}}(r-M+2) & \ldots & X_{\text{nor}}(r) \\ X_{\text{nor}}(r-M+2) & X_{\text{nor}}(r-M+3) & \ldots & X_{\text{nor}}(r+1) \\ \vdots & \vdots & \ddots & \vdots \\ X_{\text{nor}}(G-M) & X_{\text{nor}}(G-M+1) & \ldots & X_{\text{nor}}(G-1) \end{bmatrix} \in \mathbb{R}^{K \times M}. \tag{3.35}$$

and for each $v = 0, 1, \ldots, v_{\max}$ repeats the following *(2)-(5)* steps.

Step 2: MAWTS creates $Q_i, i = 1, \ldots, 4$ matrices, one for each of the four AF of (3.33), on the fitting set for X_{tr} in line with Alg. 1.

Step 3: The weights $W_i, i = 1, \ldots, 4$ are obtained for the first $k = pK$ samples (fitting set) of Q_i, i.e., $Q_i(1:k,:)$, in line with the WDD process of (3.32). That is, we set $W_i = Q_i(1:k,:)^{\dagger} Y_{\text{tr}}(1:k), i = 1, \ldots, 4$.

Step 4: Based on W_i of the previous step and the last $K - k$ samples (validation set) of Q_i, i.e., $Q_i(k+1:K,:)$, their predictions mean absolute percentage error (MAPE) over the target value Y_{tr} is measured. That is, we set $\hat{Y}_{\text{tr}}(k+1:K) = Q_i(k+1:K,:)W_i, i = 1, \ldots, 4$, where $\hat{Y}_{\text{tr}}(k+1:K)$ denotes the predictions on the validation.

Algorithm 1 Computing the matrix Q.

Input: The data X, the delays number M, the vector V which includes the optimal powers of every hidden-layer neuron checked so far, the $A = [AF1\ AF2\ AF3\ AF4]$ which includes the optimal AFs of V.

$i \leftarrow 1$
while $i \leq length(V)$ **do**
  $\hat{X} \leftarrow X^{\odot V(i)}$
  if $A(i) == AF1$ **then**
    $Q(:, M(i-1)+1 : Mi) \leftarrow \hat{X}$
  else if $A(i) == AF2$ **then**
    $Q(:, M(i-1)+1 : Mi) \leftarrow e^{\hat{X}} \odot (e^{\hat{X}} + 1)^{-1}$
  else if $A(i) == AF3$ **then**
    $Q(:, M(i-1)+1 : Mi) \leftarrow e^{-\hat{X}}$
  else if $A(i) == AF4$ **then**
    $Q(:, M(i-1)+1 : Mi) \leftarrow \ln(1+\hat{X})$
  end if
  $i \leftarrow i+1$
end while

Output: The matrix Q.

Note that the MAPE is a well-known statistic tool that measures the accuracy of a forecasting approach and is commonly used in machine learning as a loss function for regression problems. In addition, MAPE values that are closer to zero are preferable, and is calculated as follows:

$$\text{MAPE} = \frac{100\%}{K} \sum_{k=1}^{K} \left| \frac{Y_k - \hat{Y}_k}{Y_k} \right|, \tag{3.36}$$

where Y and $\hat{Y}$ signify the target and the forecasted price, respectively.

Step 5: Based on the MAPE, the MAWTS algorithm choose the best performing AF of each v, iteratively. If the MAPE for the particular v is lower than the previous best MAPE, then a new hidden-layer neuron is created under that v, and if the MAPE for the particular v is higher than the previous best MAPE, then this v is bypassed and not included in the hidden-layer neurons. As a result, MAWTS algorithm is able to keep at minimum the NN hidden-layer neurons while reducing the overall MAPE of the NN model.

Step 6: The MAWTS algorithm compares the best MAPE of the current M to the minimum MAPE of the optimal delays number so far. If so, the best MAPE of the current M becomes the minimum MAPE, and the current M becomes the optimal delays number.

Step 7: The MAWTS algorithm computes and outputs the optimal M and the optimal W on the whole samples of Q along with the optimal powers in vector V and their corresponding AF number in vector A. Note that the optimal number of the MAWTSNN's hidden-layer neurons N is equal to the length of vector V.

To summarize, normalizing the time-series data and then splitting them into the fit and the validation set, MAWTSNN model employs the MAWTS algorithm for training. Given the maximum v, $v_{\max}$, of the AF, the MAWTS algorithm finds the optimal number of delays M, the optimal AFs and the optimal structure of the NN

model measuring the MAPE of the validation set. It is worth mentioning that the optimal structure of the NN model corresponds to the optimal number of input variables M and the optimal number of hidden-layer neurons N along with their optimal powers in vector V and optimal AFs in vector A. When the optimal structure of the NN is obtained, the next Z in number prices of the time-series can easily be calculated iteratively. Notice that the input prices must be normalized and the NN output must be denormalized based on the $X_{\min}$ and $X_{\max}$, which were determined before the NN's training. Also, the complete process for modeling and forecasting with the MAWTSNN model is described in Fig. 3.15.

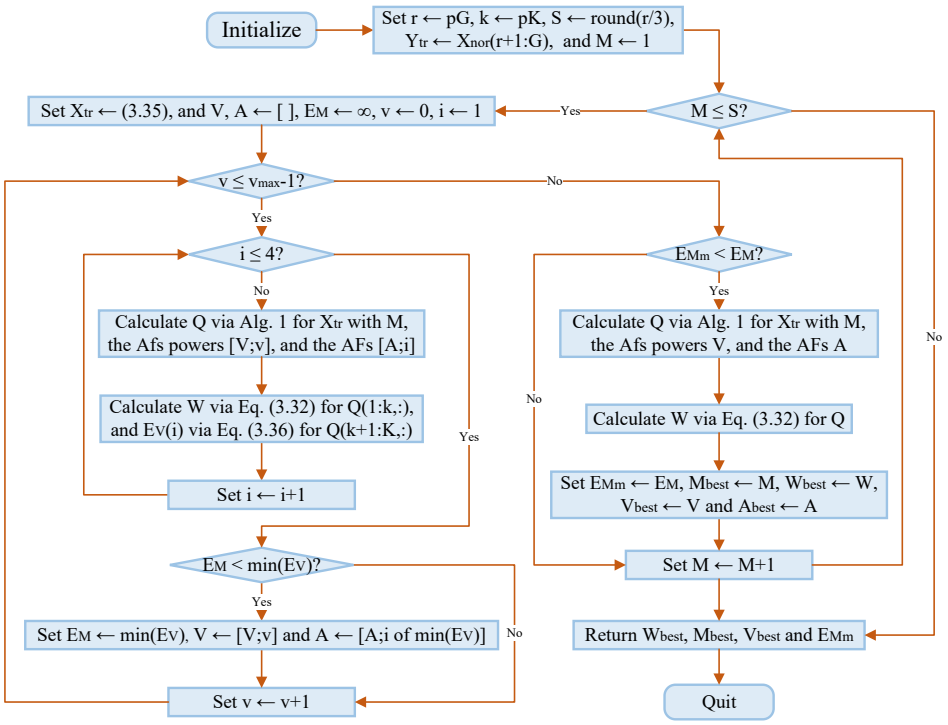


Figure 3.14: The MAWTS algorithm.

3.5 Zeroing Neural Networks

Nowadays, it is considered normal in academia and industry to handle intractability problems and solve complex computation equations by using NNs. These problems can be formulated as a set of equations by finding the zeros of them. Zeroing NNs (ZNN) are a class of NNs which are particularly dedicated to find zeros of equations. In the past years, ZNN have played an essential role in the online solution of time-varying (TV) problem and many useful research results have been reported in the literatures.

Various complicated problems have been approached with NNs in many fields of scientific research [75, 76]. For instance, an adaptive fuzzy controller based on NN is presented for a class of nonlinear discrete-time systems with discrete-time dead zone in [75]. An adaptive decentralized scheme based on NN is constructed for multiple-input and multiple-output (MIMO) nonlinear systems with the aid of back-stepping techniques in [76], where a scheme like that guarantees the uniform ultimate boundedness of all signals in the closed-loop system with

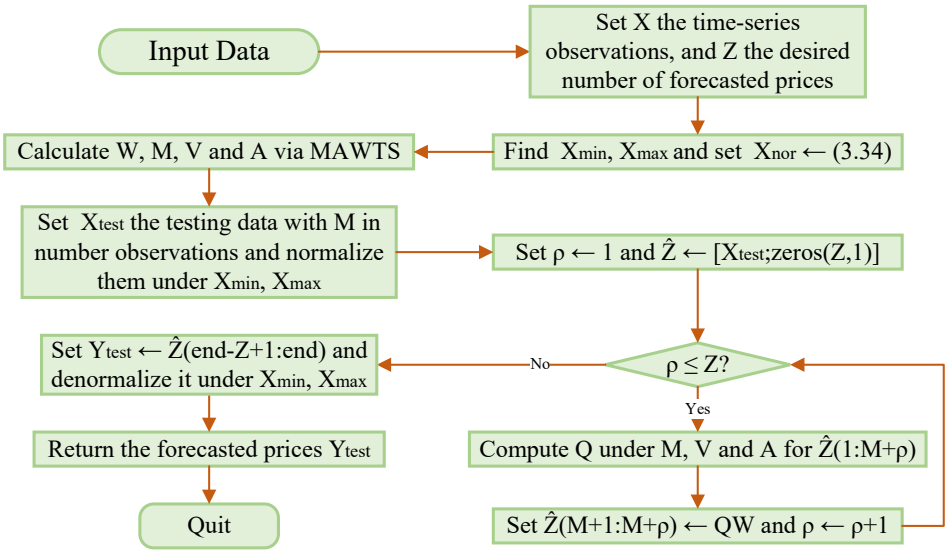


Figure 3.15: Complete process for modeling and forecasting.

respect to mean square. RNN models, as a branch of artificial intelligence, have received considerable investigation in many scientific and engineering fields, which is often exploited for computational problems [77, 78] and nonlinear optimizations are solved by many methods [79, 80].

In general, RNNs can be divided into two classes: (1) the continuous-time RNNs and (2) the discrete-time RNNs. By exploiting a numerical differential formula, a continuous-time RNN model can be discretized into a discrete-time one. Nevertheless, a numerical differentiation rule does not necessarily generate a convergent and stable discrete-time RNN model even though the original continuous-time RNN model is convergent. In addition, it can be considered as a numerical algorithm [81] if the discrete-time RNN model is coded as a serial-processing program and performed on the digital computer. ZNN is able to perfectly track TV solution by exploiting the time derivative of TV parameters as a novel type of RNN specifically designed for solving TV problems. Hence, many researchers make progresses along this direction by proposing various kinds of ZNN models for solving problems with different features.

3.5.1 Continuous-time ZNN

Consider a performance index whose minimal point is identical to the solution to the task problem, a standard approach is to design a RNN evolving along the negative gradient descent to achieve a minimum of the performance index. Many gradient-related methods had been reported on the solutions of algebraic equations and optimizations before the proposal of ZNN approach, i.e., zero-finding problems [82, 83]. Nevertheless, these methods may fail to work well when exploited to the online solution of dynamic problems with TV coefficients, which is intrinsically due to the lacking of the compensation to the velocity components of the TV coefficients. Consequently, any method designed intrinsically for computing the static problem can no longer guarantee the decrease of the performance index of a TV problem in view of the variability of coefficients, hence, the task is possibly leading to a failure with large residual error. For example, in [84, 85] it is observed, investigated and analyzed that the residual error of gradient-based NN (GNN) for solving a TV problem can not be eliminated and

remains at a relative high level. In [81] it is shown that, when exploited to solve a TV problem, any traditional method that does not exploit the time-derivative information of TV coefficients can not converge to the theoretic solution with the residual error proportional to the value of the sampling gap.

In [86], Zhang et al. present a RNN for solving the TV Sylvester equation, in order to solve a TV problem in an error-free manner which is depicted in an implicit dynamical system and can be deemed as the seminal work on ZNN. They further generalize and summarize the design procedures of such a methodology, and analyze the convergence and stability of the corresponding ZNN model for TV matrix inversion in [87]. Notably, for solving a TV matrix inversion problem depicted in the form of

$$A(t)X(t) = I, \tag{3.37}$$

where $A(t) \in \mathbb{R}^{n\times n}$ is a smooth matrix with its derivative assumed to be known, $I \in \mathbb{R}^{n\times n}$ is the identity matrix and $X(t) \in \mathbb{R}^{n\times n}$ is the unknown matrix to be obtained.

The core in the design of ZNN model is to construct an error function $E(t) = A(t)X(t) - I$, which is evidently different from the performance index of gradient-related methods. Following that, the ZNN design formula comes to enforce the corresponding $E(t)$ to converge to zero:

$$\dot{E}(t) = -\gamma\Phi(E(t)), \tag{3.38}$$

where $\gamma > 0$ and $\Phi(\cdot)$ is a matrix array of AF $\phi(\cdot)$. In [88], the ZNN design formula is implemented for vector-valued TV problems, e.g., the system of linear equation $A(t)\mathbf{x}(t) = \mathbf{b}(t)$, with $A(t) \in \mathbb{R}^{m\times n}$, $\mathbf{x}(t) \in \mathbb{R}^n$ and $\mathbf{b}(t) \in \mathbb{R}^m$, and, as a consequence, the error function can be designed as $\mathbf{e}(t) = A(t)\mathbf{x}(t) - \mathbf{b}(t)$. Even for the scalar-valued TV problems [89], e.g., the TV 4-th root finding problem $x^4(t) = \alpha(t)$, with $\alpha(t) \in \mathbb{R}$ and $x(t) \in \mathbb{R}$, the error function can be designed as $e(t) = x^4(t) - \alpha(t)$. Note that the matrix-valued (or vector-valued) error function is a decoupled system and thus its ij-th (or i-th) subsystem, i.e., the scale-valued dynamical system $\dot{e}(t) = -\gamma\phi(e(t))$, can be used to analyze the corresponding convergence and stability. In essence, any ZNN model for solving any TV problem can be deemed as an equivalently expansion of the ZNN design formula (see Tab. 3.2).

Table 3.2: Continuous-time ZNN models constructed for solving TV problems.

Dynamic problem	Error function	ZNN model	Reference
4th root finding $x^4(t) = \alpha(t)$	$e(t) = x^4(t) - \alpha(t)$	$\dot{x}(t) = \frac{\dot{\alpha}(t) - \gamma\phi(x^4(t) - \alpha(t))}{4x^3(t)}$	[89]
Linear system $A(t)\mathbf{x}(t) = \mathbf{b}(t)$	$\mathbf{e}(t) = A(t)\mathbf{x}(t) - \mathbf{b}(t)$	$A(t)\dot{\mathbf{x}}(t) = -\dot{A}(t)\mathbf{x}(t) + \dot{\mathbf{b}}(t) - \gamma\Phi(A(t)\mathbf{x}(t) - \mathbf{b}(t))$	[90]
Matrix inversion $A(t)X(t) = I$	$E(t) = A(t)X(t) - I$	$A(t)\dot{X}(t) = -\dot{A}(t)X(t) - \gamma\Phi(A(t)X(t) - I)$	[87]
Matrix square roots finding $X^2(t) = A(t)$	$E(t) = X^2(t) - A(t)$	$X(t)\dot{X}(t) + \dot{X}(t)X(t) = -\gamma\Phi(X^2(t) - A(t)) - \dot{A}(t)$	[91]
Nonlinear equations $\mathbf{f}(\mathbf{x}(t), t) = 0$	$\mathbf{e}(t) = \mathbf{f}(\mathbf{x}(t), t)$	$\dot{\mathbf{x}}(t) = -J^{-1}(\mathbf{x}(t), t)(\gamma\Phi(\mathbf{f}(\mathbf{x}(t), t)) + \frac{\partial \mathbf{f}(\mathbf{x}(t), t)}{\partial t})$	[92]

Modified ZNN models have been frequently proposed by considering different internal and external factors since Zhang et al. proposed them in the 2000s. Especially, when nonlinear AFs are incorporated into the network models, stability research has gained significant progress. A brief review on the design of continuous-time ZNN models for various problems solving is presented in [93] and some new variations have taken place in [94].

Convergence and stability

The basic problems in the research of NNs are convergence and stability. In general, there are three ways for proving the convergence of ZNN models, i.e., proof based on Lyapunov theory [95, 96], ordinary differential equation (ODE) [87, 88], or Laplace transform [84, 97–99].

Nonlinear AFs

For the acceleration of the convergent speed of ZNN models, nonlinear AFs are used in their design and construction. Commonly, the following ones are often employed to construct ZNN models [91, 100]:

- the power-sum AF: $\phi(e_i) = \sum_0^N e_i^{2N-1}$, where $N > 1$.
- the power-sigmoid AF:
$$\phi(e_i) = \begin{cases} \frac{1+\exp(-\xi)}{1-\exp(-\xi)} \cdot \frac{1-\exp(-\xi e_i)}{1+\exp(-\xi e_i)}, & \text{if } |e_i| < 1 \\ e_i^p, & \text{if } |e_i| \geq 1 \end{cases}$$
where p is an odd integer and $\xi > 0$.
- and the hyperbolic sine AF: $\phi(e_i) = \frac{\exp(e_i m)}{2} - \frac{\exp(-e_i m)}{2}$, where m is an odd integer.

In order to prove the convergence of these nonlinear function activated ZNN models, a common approach is to construct a Lyapunov function candidate $V(t) = \mathbf{e}^{\mathrm{T}}(t)\mathbf{e}(t)/2$, and then compute its time derivative $\dot{V}$, which is smaller than that of linear function activated ZNN model. Hence, the conclusion is that nonlinear AF can be used to accelerate the convergence speed. Many existing results on the ZNN concern the case that AFs should be continuous and strictly monotonically increasing, which is a limitation and should be resolved in the future.

Finite-time convergence

Li et al. present a nonlinear function to accelerate the continuous-time ZNN to finite-time convergence for solving TV Sylvester equation in [101] and follow up with the online solution of dual NNs for solving quadratic programming problems in [102] as a result of the in-depth research on the ZNN model and inspired by the study on finite-time convergence in continuous autonomous system [103, 104]. Thereafter, many different finite-time AFs have been proposed and employed to various ZNN models, e.g, TV matrix pseudoinversion in complex domain [105], Lyapunov equation [106], equality-constrained quadratic optimization [107], linear complex matrix equation [108] and so on [109]. Observe that the residual error of a traditional ZNN model exponentially converges to zero, which means that the smaller the residual error is, the slower the convergent speed is. Consequently, an effective way to achieve finite-time convergence is to amplify the value of $\dot{e}(t)/e(t)$ to large enough when $e(t)$ approaches to zero. Furthermore, in [110], it is investigated that various ZNN models can be designed by exploiting different error functions for a TV problem solving, which can be accelerated to finite-time convergence.

Complex-valued ZNN models

Over the past few years, different ZNN models have received considerable studies in many scientific and engineering fields, which successfully tackles the estimation error problem in the real domain. In comparison to these ZNN models defined in the real domain, the research on complex-valued ZNN models shows advantage over conventional real-valued NNs in complex problems solving. The first proposition of a complex-valued ZNN model capable to solve the TV matrix-inversion problems in complex domain was made in [96]. Note that it uses only linear AFs for ensuring the global convergence of the NN. As mentioned earlier, many nonlinear function can be used to accelerate the convergence speed of ZNN model, which inspires Li et al. to explore nonlinear complex-valued AFs to accelerate the convergence of ZNN with guaranteed global convergence in [111]. They find two classes of AFs to achieve the global convergence of the complex-valued ZNN for solving the complex-valued TV Sylvester equation and then accelerate it to finite-time convergence. By exploiting different techniques to compute the TV complex-valued pseudoinversion problem Liao et al. propose five ZNN models in [112], which is further generalized to complex-valued matrix inversion in [113]. Qiao et al. present two complex-valued ZNN models for computing the Drazin inverse, which is accelerated to finite-time with proven upper bounds of the convergence time in [109]. Also, online solution of complex-valued systems of linear equations is investigated in the complex domain via a GNN model in [114].

Noise-tolerant ZNN models

For solving TV problems, many computational models such as the conventional ZNN models usually assume that the solving task is free of noises or that the noise reduction operation has been completed before the computation [115]. Point out that time is valuable for the online computation of TV problems and the preprocessing for denoising may consume extra time. In that case, the online computation is no longer required. In order to avoid that an integration-enhanced ZNN design formula is proposed in [98] for TV matrix inversion, which leverages the integration-control technique in control theoretical and is able to handle simultaneously the noises. In [99], such a noise-tolerant ZNN design formula is used to construct continuous-time ZNN model with nonlinear AF exploited for accelerating the noise-tolerant model. Authors in [97] analyze and design different ZNN models via a systematic approach from the control perspective for solving TV problems. The essence of the noise-tolerant ZNN models is to leverage the error integration information to eliminate the constant bias errors. The challenge in this approach is how to determine the suitable AFs for accelerating its convergence. By exploiting noise-tolerant ZNN design formula to solve different TV problems, various noise-tolerant ZNN models that exploit the time-derivative information of coefficients as well as the error integration can be constructed with their formulations shown in Tab. 3.3.

Table 3.3: Continuous-time noise-tolerant ZNN models constructed for solving TV problems.

Dynamic problem	**Noise-tolerant ZNN model**
4th root finding $x^4(t) = \alpha(t)$	$\dot{x}(t) = \frac{\dot{\alpha}(t) - \gamma(x^4(t) - \alpha(t))}{4x^3(t)} - \lambda \int_0^t (x^4(\delta) - \alpha(\delta))d\delta$
Linear system $A(t)\mathbf{x}(t) = \mathbf{b}(t)$	$A(t)\dot{\mathbf{x}}(t) = -\dot{A}(t)\mathbf{x}(t) + \dot{\mathbf{b}}(t) - \gamma(A(t)\mathbf{x}(t) - \mathbf{b}(t)) - \lambda \int_0^t (A(\delta)\mathbf{x}(\delta) - \mathbf{b}(\delta))d\delta$
Matrix inversion $A(t)X(t) = I$	$A(t)\dot{X}(t) = -\dot{A}(t)X(t) - \gamma(A(t)X(t) - I) - \lambda \int_0^t (A(\delta)X(\delta) - I)d\delta$
Matrix square roots finding $X^2(t) = A(t)$	$X(t)\dot{X}(t) + \dot{X}(t)X(t) = -\gamma(X^2(t) - A(t)) - \dot{A}(t) - \lambda \int_0^t (X^2(\delta) - A(\delta))d\delta$
Nonlinear equations $\mathbf{f}(\mathbf{x}(t), t) = 0$	$\dot{\mathbf{x}}(t) = -J^{-1}(\mathbf{x}(t), t)\ (\gamma(\mathbf{f}(\mathbf{x}(t), t)) + \frac{\partial \mathbf{f}(\mathbf{x}(t), t)}{\partial t} - \lambda \int_0^t (\mathbf{f}(\mathbf{x}(\delta), \delta))d\delta)$

3.5.2 Discrete-time ZNN

As mentioned before, for solving different problems there are various presentations of continuous-time ZNN models. Nevertheless, it is different for a digital computer to implement continuous-time ZNN models directly owing to the fact that step size in simulating continuous-time systems is variable and that the digital computer often requires constant time step [116, 117]. Also, it is always assumed that neurons communicated and responded instantaneously and without any delay for continuous-time ZNN models. However, time delay is unavoidable in digital circuits due to the existing of the sampling gap. So, researchers devote their effects to propose and

investigate discrete-time ZNN (DTZNN) models for online TV problems solving. In addition, some challenges of developing the DTZNN models from the continuous-time ones for the TV problems solving are listed as follows:
1. In terms of time, any TV problem can be considered as a causal system and thereby, the related computation should be conducted based on the existing data, i.e., the present and/or previous data, for computing the future solution because of the unavailability of future data. For example, for solving the TV matrix inversion problem 3.37 in a discrete manner, at the time instant t_k, we can use the known information, e.g., $A(t_k)$ and $\dot{A}(t_k)$, not the unknown information, e.g., $A(t_{k+1})$ and $\dot{A}(t_{k+1})$ for computing the inverse of $A(t_{k+1})$, i.e., $X(t_{k+1})$. So, a fundamental requirement for constructing the DTZNN model is that we can not use future data.
2. Proportionally, a attainable numerical differentiation formula for discretizing continuous-time ZNN model should only have one point ahead of the target point. Thus, the constructed DTZNN model has only one unknown point $X(t_{k+1})$ to be computed via the known data (e.g., $A(t_k)$ and $\dot{A}(t_k)$ when the numerical differentiation formula is used to discretize the continuous-time ZNN model. Hence, the backward and multiple-point central differentiation rules can not be used to construct DTZNN models, no matter how tiny the truncation error of each formula is. Also, only the forward differentiation formulas with one step ahead can be considered for the discretization of ZNN.
3. Computation consumes time inevitably at each time instant and time is important for the TV problems solving in practice. Hence, it is highly important for someone to know how to design a very simple DTZNN model with less calculation time.

The authors in [118] propose the first DTZNN model for constant matrix inversion, which bridges the gap between DTZNN model and the traditional Newton iteration. At the early stage, Euler forward difference is used to construct the discretize the continuous-time ZNN model [119, 120]. The DTZNN models generated by Euler forward difference are of the error pattern of $O(\tau^2)$, where τ denotes the sampling gap. For example, the Euler-type DTZNN model for TV matrix inversion is directly given as [121]:

$$X_{k+1} = X_k - \tau X_k \dot{A}_k X_k - h X_k (A_k X_k - I), \tag{3.39}$$

where k denotes the iteration index and $h = \tau\gamma > 0$. Also, by leaving out the term $\tau X_k \dot{A}_k X_k$ and letting $h = 1$, the Euler-type DTZNN model 3.39 reduces to

$$X_{k+1} = X_k - X_k (A_k X_k - I), \tag{3.40}$$

which is the Newton iteration. In essence, the traditional Newton iteration can be deemed as a special case of Euler-type DTZNN model 3.39. In the meantime, the link between Getz–Marsden dynamic system and DTZNN models is found in [122]. To achieving high accuracy in the discretization of ZNN models, a Taylor-type numerical differentiation formula is proposed in [121] for the first-order derivative approximation. It has a truncation error of $O(\tau^2)$ and formulated as

$$f'(t_k) = \frac{2f(t_{k+1}) - 3f(t_k) + 2f(t_{k-1}) - f(t_{k-2})}{2\tau} + O(\tau^2). \tag{3.41}$$

For TV matrix inversion a new Taylor-type DTZNN model can be developed as

$$X_{k+1} = -\tau X_k \dot{A}_k X_k - hX_k(A_k X_k - I) + \frac{3}{2}X_k - X_{k-1} + \frac{1}{2}X_{k-2}. \tag{3.42}$$

The above Taylor-type DTZNN model converges to the theoretical solution of the TV problem with the residual error being $O(\tau^3)$. In [123], a numerical difference rule is established for first-order derivative approximation with the truncation error of $O(\tau^3)$. Based on such a new formula, a five-step DTZNN model is proposed for TV matrix inversion, of which the residual error is $O(\tau^4)$. Observe that, the DTZNN models can be deemed as time delay systems, and so, a large value of h may lead the model to oscillate. A direct way to remedy the instability is to lessen the value of h. However, lessening the value of h would significantly slow the convergence of DTZNN models.

3.6 Neural Networks Solvers for Linear and Quadratic Programming Problems

In this section, TV linear programming (LP) problems and TV quadratic programming (QP) problems are approached by two continuous-time NN solvers. These solvers are the zeroing NN (ZNN) and the linear-variational-inequality primal-dual NN (LVI-PDNN). It is worth mentioning that two ZNN solvers are introduced, one based on the penalty function method and the other on the Karush-Kuhn-Tucker (KKT) conditions. The following are some of the section's general notations: the symbols $\mathbf{1}_n, \mathbf{0}_n$ denote a vector in $\mathbb{R}^n$ consisting of ones and zeros, respectively; $\mathbf{0}_{n\times n} \in \mathbb{R}^{n\times n}$ denotes a zero matrix of $n \times n$ dimensions; $I_n \in \mathbb{R}^{n\times n}$ denotes the identity $n \times n$ matrix; $\odot$ denotes the Hadamard (or element-wise) product and the superscript $()^{\odot}$ denotes the Hadamard exponential; $\bar{x}$ denotes a square diagonal matrix with the elements of vector x on the main diagonal.

3.6.1 The LP/QP ZNN Solver based on the Penalty Function Method

In view of its fundamental role playing in mathematical optimization, LP and QP problems have been investigated extensively through the last decades [124–126]. Consider the following TV QP (TVQP) problem:

$$\min_{x(t)} \quad x^{\mathrm{T}}(t)W(t)x(t)/2 + q^{\mathrm{T}}(t)x(t) \tag{3.43}$$

$$\text{s.t.} \quad K(t)x(t) = \rho(t) \tag{3.44}$$

$$D(t)x(t) \le g(t) \tag{3.45}$$

$$-\xi^- \le x(t) \le \xi^+ \tag{3.46}$$

where the coefficient matrices $K(t) \in \mathbb{R}^{m\times n}$ and $D(t) \in \mathbb{R}^{k\times n}$ are smoothly TV, while $W(t) \in \mathbb{R}^{n\times n}$ is smoothly TV, positive-definite and symmetric at any time instant $t \in [0, t_f] \subseteq [0, +\infty)$. At the mean time, coefficient vectors $q(t), \xi^-(t), \xi^+(t) \in \mathbb{R}^n$, $\rho(t) \in \mathbb{R}^m$ and $g(t) \in \mathbb{R}^k$ are all smoothly TV as well. Also, $x(t) \in \mathbb{R}^n$ is the desired solution to be found in real time t. It is important to mention that when $W(t) = \mathbf{0}_{n\times n}$, the TVQP problem of (3.43)-(3.46) becomes a TV LP problem. The three stages below can be followed to formulate a ZNN model for solving the TVQP problem of (3.43)-(3.46) based on [127, 128].

Stage 1: **(Reformulation of the TVQP problem)** The TVQP problem of (3.43)-(3.46) can be formulated as follows:

$$\min_{x(t)} \quad x^{\mathrm{T}}(t)W(t)x(t)/2 + q^{\mathrm{T}}(t)x(t) \tag{3.47}$$

$$\text{s.t.} \quad K(t)x(t) = \rho(t) \tag{3.48}$$
$$D(t)x(t) \leq g(t) \tag{3.49}$$
$$x(t) \leq \xi^{+} \tag{3.50}$$
$$-x(t) \leq -\xi^{-}, \tag{3.51}$$

Additionally, the TVQP problem of (3.47)-(3.51) can be reformulated as follows:

$$\min_{x(t)} \quad x^{\mathrm{T}}(t)W(t)x(t)/2 + q^{\mathrm{T}}(t)x(t) \tag{3.52}$$

$$\text{s.t.} \quad K(t)x(t) = \rho(t) \tag{3.53}$$
$$A(t)x(t) \leq b(t), \tag{3.54}$$

where $A(t) = \begin{bmatrix} D(t) \\ I_n \\ -I_n \end{bmatrix} \in \mathbb{R}^{(k+2n)\times n}$ and $b(t) = \begin{bmatrix} g(t) \\ \xi^{+} \\ -\xi^{-} \end{bmatrix} \in \mathbb{R}^{k+2n}$. Then, using the penalty function proposed in [127, 128], the problem of (3.52)-(3.54) may be reformulated as below:

$$\min_{x(t)} \quad x^{\mathrm{T}}(t)W(t)x(t)/2 + q^{\mathrm{T}}(t)x(t) + P(x(t)) \tag{3.55}$$

$$\text{s.t.} \quad K(t)x(t) = \rho(t), \tag{3.56}$$

where $P(x(t)) = p\sum_{i=1}^{2n+k} e^{-sN_i(t)}$ with $N_i(t) = b_i(t) - A_i(t)x(t)$, $i = 1,2,\ldots,2n+k$. Note that $p \geq 0$ is the penalty parameter and $s > 0$ is the design parameter. Notice that when the value of s increases, the value of $P(x(t))$ is more consistent with conditions (3.54), while if is too large, the calculations' time will increase. The parameter p may make the value of $P(x(t))$ close to zero when $N(t) = 0$.

Stage 2: **(Minimization conditions)** To solve the TVQP problem of (3.55)-(3.56), the following Lagrange function is determined:

$$L(x(t),\lambda(t),t) = x^{\mathrm{T}}(t)W(t)x(t)/2 + P(x(t)) + \lambda^{\mathrm{T}}(t)(K(t)x(t) - \rho(t)), \tag{3.57}$$

where $\lambda(t) \in \mathbb{R}^m$. Then, the first-order conditions are:

$$\begin{cases} \frac{\partial L(x(t),\lambda(t),t)}{\partial x(t)} = W(t)x(t) + q(t) + P_x(t) + K^{\mathrm{T}}(t)\lambda(t) = 0 \\ \frac{\partial L(x(t),\lambda(t),t)}{\partial \lambda(t)} = K(t)x(t) - \rho(t) = 0 \end{cases} \tag{3.58}$$

where $P_x(t) = ps\sum_{i=1}^{2n+k}\left(e^{-sN_i(t)}A_i^{\mathrm{T}}(t)\right) \in \mathbb{R}^n$.

Stage 3: **(ZNN solver)** Setting the following error matrix equation group:

$$\begin{cases} E_1(t) = W(t)x(t) + q(t) + P_x(t) + K^{\mathrm{T}}(t)\lambda(t) \\ E_2(t) = K(t)x(t) - \rho(t) \end{cases} \tag{3.59}$$

and then substituting $E_i(t), i = 1,2$, defined in (3.59) in place of $E(t)$ into (3.38), one obtains

$$\begin{cases} \dot{W}(t)x(t) + W(t)\dot{x}(t) + \dot{q}(t) + \dot{P}_x(t) + K^{\mathrm{T}}(t)\dot{\lambda}(t) + \dot{K}^{\mathrm{T}}(t)\lambda(t) = -\gamma E_1(t) \\ \dot{K}(t)x(t) + K(t)\dot{x}(t) - \dot{\rho}(t) = -\gamma E_2(t) \end{cases} \tag{3.60}$$

where $\dot{P}_x(t) = P_1(t)\dot{x}(t) + P_2(t)x(t) + P_3(t)$ with

$$\begin{aligned} P_1(t) &= ps^2 \sum_{i=1}^{2n+k} \left(e^{-sN_i(t)} A_i^{\mathrm{T}}(t) A_i(t)\right) \in \mathbb{R}^{n\times n}, \\ P_2(t) &= ps^2 \sum_{i=1}^{2n+k} \left(e^{-sN_i(t)} A_i^{\mathrm{T}}(t) \dot{A}_i(t)\right) \in \mathbb{R}^{n\times n}, \\ P_3(t) &= ps \sum_{i=1}^{2n+k} \left(e^{-sN_i(t)} \left(\dot{A}_i^{\mathrm{T}}(t) - sA_i^{\mathrm{T}}(t)\dot{b}(t)\right)\right) \in \mathbb{R}^{n}. \end{aligned} \tag{3.61}$$

It is worth mentioning that $\dot{A}(t) = \begin{bmatrix} -\dot{D}(t)^{\mathrm{T}} \\ \mathbf{0}_{n\times n} \\ \mathbf{0}_{n\times n} \end{bmatrix} \in \mathbb{R}^{(k+2n)\times n}$ and $\dot{b}(t) = \begin{bmatrix} -\dot{g}(t) \\ \mathbf{0}_n \\ \mathbf{0}_n \end{bmatrix} \in \mathbb{R}^{k+2n}$. As a result, (3.60) can be reformulated as follows:

$$\begin{cases} \left(W(t) + P_1(t)\right)\dot{x}(t) + K^{\mathrm{T}}(t)\dot{\lambda}(t) = -\gamma E_1(t) - \dot{W}(t)x(t) - \dot{q}(t) - \dot{K}^{\mathrm{T}}(t)\lambda(t) - P_2(t)x(t) - P_3(t) \\ K(t)\dot{x}(t) = -\gamma E_2(t) - \dot{K}(t)x(t) + \dot{\rho}(t) \end{cases} \tag{3.62}$$

Then setting

$$S(t) = \begin{bmatrix} W(t) + P_1(t) & K^{\mathrm{T}}(t) \\ K(t) & \mathbf{0}_{m\times m} \end{bmatrix}, \quad u(t) = \begin{bmatrix} -\gamma E_1(t) - \dot{W}(t)x(t) - \dot{q}(t) - \dot{K}^{\mathrm{T}}(t)\lambda(t) - P_2(t)x(t) - P_3(t) \\ -\gamma E_2(t) - \dot{K}(t)x(t) + \dot{\rho}(t) \end{bmatrix}, \tag{3.63}$$

where $S(t) \in \mathbb{R}^{(n+m)\times(n+m)}$ and $u(t) \in \mathbb{R}^{n+m}$, (3.62) can be reformulated as follows:

$$S(t)\dot{y}(t) = u(t), \tag{3.64}$$

where $\dot{y}(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{\lambda}(t) \end{bmatrix} \in \mathbb{R}^{n+m}$ and $S(t)$ is a nonsingular mass matrix. As a result, the proposed ZNN model for solving the TVQP problem of (3.55)-(3.56) is (3.64). It is worth mentioning that the solution $y(t)$ of (3.64) can be efficiently generated by employing an `ode` MATLAB solver. The convergence of the ZNN solver to the unique theoretical solution is proven in the following theorem 3.6.1.

Theorem 3.6.1 *Starting from any initial state $y(0) \in \mathbb{R}^{n+m}$, the state variable $y(t) = [x(t)^{*\mathrm{T}}, \lambda(t)^{*\mathrm{T}}]^{\mathrm{T}}$ of ZNN* (3.64) *globally converges to the unique theoretical solution $y^*(t) = [x(t)^{\mathrm{T}}, \lambda(t)^{\mathrm{T}}]^{\mathrm{T}}$. That is, $\lim_{t\to\infty}(y(t) - y^*(t)) = 0$. Furthermore, the theoretical solution $x^*(t)$ of TVQP* (3.55)-(3.56) *is the first n elements of $y^*(t)$.*

Proof. In order to obtain the solution $y(t)$ of TVQP (3.55)-(3.56), the error matrix equation group is defined as in (3.59), based on the ZNN design. After that, by adopting the linear design formula for zeroing (3.59), the model

(3.60) is obtained. From [87, Theorem 1], each error matrix equation in the error matrix equation group (3.60) converges to the theoretical solution when $t \to \infty$. As a result, the solution of (3.60) converges to the theoretical solution of TVQP (3.55)-(3.56) when $t \to \infty$. In addition, from the derivation process of (3.64), we know that it is actually another form of (3.60). The proof is thus completed. □

3.6.2 The LP/QP ZNN Solver based on the KKT Conditions

Using the same concept as the previous subsection, the three stages below can be followed to create a ZNN model for solving the TVQP problem of (3.43)-(3.46) based on the KKT conditions [129, 130].

Stage 1: **(Reformulation of the TVQP problem)** Consider the following reformulation of the TVQP problem of (3.47)-(3.51):

$$\begin{aligned} \min_{x(t)} \quad & x^{\mathrm{T}}(t)W(t)x(t)/2 + q^{\mathrm{T}}(t)x(t) && (3.65)\\ \text{s.t.} \quad & K(t)x(t) = \rho(t) && (3.66)\\ & A(t)x(t) + R^{\odot 2}(t) = b(t), && (3.67) \end{aligned}$$

where $A(t) = \begin{bmatrix} D(t) \\ I_n \\ -I_n \end{bmatrix} \in \mathbb{R}^{(k+2n)\times n}$, $b(t) = \begin{bmatrix} g(t) \\ \xi^{+} \\ -\xi^{-} \end{bmatrix} \in \mathbb{R}^{k+2n}$ and $R(t) \in \mathbb{R}^{k+2n}$.

Stage 2: **(Minimization conditions)** To solve the TVQP problem of (3.65)-(3.67), the following Lagrange function is determined:

$$L(x(t), \lambda_1(t), \lambda_2(t), R(t), t) = x^{\mathrm{T}}(t)W(t)x(t)/2 + \lambda_1^{\mathrm{T}}(t)(K(t)x(t) - \rho(t)) + \lambda_2^{\mathrm{T}}(t)(A(t)x(t) + R^{\odot 2}(t) - b(t)), \quad (3.68)$$

where $\lambda_i(t) \in \mathbb{R}^m, i = 1, 2$. Then, using the KKT conditions [131], we have

$$\begin{cases} \frac{\partial L(x(t),\lambda_1(t),\lambda_2(t),R(t),t)}{\partial x(t)} = W(t)x(t) + q(t) + K^{\mathrm{T}}(t)\lambda_1(t) + A^{\mathrm{T}}(t)\lambda_2(t) = 0 \\ \frac{\partial L(x(t),\lambda_1(t),\lambda_2(t),R(t),t)}{\partial \lambda_1(t)} = K(t)x(t) - \rho(t) = 0 \\ \frac{\partial L(x(t),\lambda_1(t),\lambda_2(t),R(t),t)}{\partial \lambda_2(t)} = A(t)x(t) + R^{\odot 2}(t) - b(t) = 0 \\ \frac{\partial L(x(t),\lambda_1(t),\lambda_2(t),R(t),t)}{\partial R(t)} = \lambda_2(t) \odot R(t) = 0 \end{cases} \quad (3.69)$$

Stage 3: **(ZNN solver)** Setting the following error matrix equation group:

$$\begin{cases} E_1(t) = W(t)x(t) + q(t) + K^{\mathrm{T}}(t)\lambda_1(t) + A^{\mathrm{T}}(t)\lambda_2(t) \\ E_2(t) = K(t)x(t) - \rho(t) \\ E_3(t) = A(t)x(t) + R^{\odot 2}(t) - b(t) \\ E_4(t) = \lambda_2(t) \odot R(t) \end{cases} \quad (3.70)$$

and then substituting $E_i(t), i = 1, \ldots, 4$, defined in (3.70) in place of $E(t)$ into (3.38), one obtains

$$\begin{cases} \dot{W}(t)x(t) + W(t)\dot{x}(t) + \dot{q}(t) + K^{\mathrm{T}}(t)\dot{\lambda}_1(t) + \dot{K}^{\mathrm{T}}(t)\lambda_1(t) + A^{\mathrm{T}}(t)\dot{\lambda}_2(t) + \dot{A}^{\mathrm{T}}(t)\lambda_2(t) = -\gamma E_1(t) \\ \dot{K}(t)x(t) + K(t)\dot{x}(t) - \dot{\rho}(t) = -\gamma E_2(t) \\ \dot{A}(t)x(t) + A(t)\dot{x}(t) + 2R(t)\dot{R}(t) - \dot{b}(t) = -\gamma E_3(t) \\ \dot{\lambda}_2(t)R(t) + \lambda_2(t)\dot{R}(t) = -\gamma E_4(t) \end{cases} \tag{3.71}$$

It is worth mentioning that $\dot{A}(t) = \begin{bmatrix} -\dot{D}(t)^{\mathrm{T}} \\ \mathbf{0}_{n\times n} \\ \mathbf{0}_{n\times n} \end{bmatrix} \in \mathbb{R}^{(k+2n)\times n}$ and $\dot{b}(t) = \begin{bmatrix} -\dot{g}(t) \\ \mathbf{0}_n \\ \mathbf{0}_n \end{bmatrix} \in \mathbb{R}^{k+2n}$. Then setting

$$\begin{aligned} S(t) &= \begin{bmatrix} W(t) & K^{\mathrm{T}}(t) & A^{\mathrm{T}}(t) & \mathbf{0}_{n\times(k+2n)} \\ K(t) & \mathbf{0}_{m\times m} & \mathbf{0}_{m\times(k+2n)} & \mathbf{0}_{m\times(k+2n)} \\ A(t) & \mathbf{0}_{(k+2n)\times m} & \mathbf{0}_{(k+2n)\times(k+2n)} & 2\bar{R}(t) \\ \mathbf{0}_{(k+2n)\times n} & \mathbf{0}_{(k+2n)\times m} & \bar{R}(t) & \bar{\lambda}_2(t) \end{bmatrix}, \\ u(t) &= \begin{bmatrix} -\gamma E_1(t) - \dot{W}(t)x(t) - \dot{q}(t) - \dot{K}^{\mathrm{T}}(t)\lambda_1(t) - \dot{A}^{\mathrm{T}}(t)\lambda_2(t) \\ -\gamma E_2(t) - \dot{K}(t)x(t) + \dot{\rho}(t) \\ -\gamma E_3(t) - \dot{A}(t)x(t) + \dot{b}(t) \\ -\gamma E_4(t) \end{bmatrix}, \end{aligned} \tag{3.72}$$

where $S(t) \in \mathbb{R}^{(5n+m+2k)\times(5n+m+2k)}$ and $u(t) \in \mathbb{R}^{5n+m+2k}$, (3.71) can be reformulated as follows:

$$S(t)\dot{y}(t) = u(t), \tag{3.73}$$

where $\dot{y}(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{\lambda}_1(t) \\ \dot{\lambda}_2(t) \\ \dot{R}(t) \end{bmatrix} \in \mathbb{R}^{5n+m+2k}$ and $S(t)$ is a nonsingular mass matrix. As a result, the proposed ZNN model for solving the TVQP problem of (3.65)-(3.67) is (3.64). It is worth mentioning that the solution $y(t)$ of (3.73) can be efficiently generated by employing an `ode` MATLAB solver. The convergence of the ZNN solver to the unique theoretical solution is proven in the following theorem 3.6.2.

Theorem 3.6.2 *Starting from any initial state $y(0) \in \mathbb{R}^{n+m}$, the state variable $y(t) = [x(t)^{*\mathrm{T}}, \lambda_1(t)^{*\mathrm{T}}, \lambda_2(t)^{*\mathrm{T}}, R(t)^{*\mathrm{T}}]^{\mathrm{T}}$ of ZNN (3.73) globally converges to the unique theoretical solution $y^*(t) = [x(t)^{\mathrm{T}}, \lambda_1(t)^{\mathrm{T}}, \lambda_2(t)^{\mathrm{T}}, R(t)^{\mathrm{T}}]^{\mathrm{T}}$. That is, $\lim_{t\to\infty}(y(t) - y^*(t)) = 0$. Furthermore, the theoretical solution $x^*(t)$ of TVQP (3.65)-(3.67) is the first n elements of $y^*(t)$.*

Proof. In order to obtain the solution $y(t)$ of TVQP (3.65)-(3.67), the error matrix equation group is defined as in (3.70), based on the ZNN design. After that, by adopting the linear design formula for zeroing (3.70), the model (3.71) is obtained. From [87, Theorem 1], each error matrix equation in the error matrix equation group (3.71)

converges to the theoretical solution when $t \to \infty$. As a result, the solution of (3.71) converges to the theoretical solution of TVQP (3.65)-(3.67) when $t \to \infty$. In addition, from the derivation process of (3.73), we know that it is actually another form of (3.71). The proof is thus completed. □

3.6.3 The LP/QP LVI-PDNN Solver

As a specific case of mathematical programming, solving LP/QP problems has been widely encountered in numerous scientific disciplines and industrial applications, such as robotic control [98], data regression [132], human movement analysis [133], pattern recognition [134] and signal processing [135]. In the past, the LP/QP problems investigated by researchers are usually only subject to less than two different kinds of constraints [132]. In addition, most of the researches are dedicated in investigating LP/QP problems that are based on static coefficients [125], which means that the methods and the algorithms that are designed for handling such a class of LP/QP problems are incompetent when being applied to TV cases.

It should be noted that due to its parallel distributed computing nature and hardware-implementation availability, the NN approach has been considered as a powerful tool for real-time computation for more than a decade [136–148]. For example, in [136], it is presented a simple, piecewise-linear, and global convergent to optimal solutions dual NN. A primal-dual NN (PDNN) with simple and piecewise-linear dynamics based on linear variational inequality (LVI) is introduced [148]. A linear-variational-inequality based primal-dual NN (LVI-PDNN) is globally convergent to the optimal solution(s) and obtain the capability of handling both LP and QP problems that are in the same/unified manner [143].

The general TVQP problem formulation investigated in [149] is presented as the equations:

$$\min_{x(t)} \quad x^{\mathrm{T}}(t)W(t)x(t)/2 + q^{\mathrm{T}}(t)x(t) \tag{3.74}$$

$$\text{s.t.} \quad J(t)x(t) = d(t) \tag{3.75}$$

$$A(t)x(t) \le b(t) \tag{3.76}$$

$$\xi^{-}(t) \le x(t) \le \xi^{+}(t), \tag{3.77}$$

where the coefficient matrices $J(t) \in \mathbb{R}^{m\times n}$ and $A(t) \in \mathbb{R}^{k\times n}$ are smoothly TV, while $W(t) \in \mathbb{R}^{n\times n}$ is smoothly TV, positive-definite and symmetric at any time instant $t \in [0, t_f] \subseteq [0, +\infty)$. At the mean time, coefficient vectors $q(t), \xi^{-}(t), \xi^{+}(t) \in \mathbb{R}^n$, $d(t) \in \mathbb{R}^m$ and $b(t) \in \mathbb{R}^k$ are all smoothly TV as well. In TV LP/QP (3.74)-(3.77), unknown vector $x(t) \in \mathbb{R}^n$ is to be solved in real time t.

In order to convert the TVQP problem of (3.74)-(3.77) into LVI-PDNN, we need to define some coefficients first. Those coefficients are as follows:

$$H(t) = \begin{bmatrix} W(t) & -J^{\mathrm{T}}(t) & A^{\mathrm{T}}(t) \\ J(t) & \mathbf{0}_{m\times m} & \mathbf{0}_{m\times k} \\ -A(t) & \mathbf{0}_{k\times m} & \mathbf{0}_{k\times k} \end{bmatrix}, \quad h(t) = \begin{bmatrix} q(t) \\ -d(t) \\ b(t) \end{bmatrix}.$$

Moreover, the definition of the primal-dual decision vector $y(t)$ along with the lower and upper bounds, to which is subject to, are depicted as

$$y(t) = \begin{bmatrix} x(t) \\ u(t) \\ v(t) \end{bmatrix}, \quad \varsigma^{-}(t) = \begin{bmatrix} \xi^{-}(t) \\ -\varpi\mathbf{1}_m \\ \mathbf{0}_k \end{bmatrix}, \quad \varsigma^{+}(t) = \begin{bmatrix} \xi^{+}(t) \\ +\varpi\mathbf{1}_m \\ +\varpi\mathbf{1}_k \end{bmatrix},$$

where

- constant $\varpi \gg 0$, being the numerical representation of $+\infty$, is sufficiently large for implementation purposes;
- $x(t) \in [\xi^-(t), \xi^+(t)]$ denotes the original decision variable vector of primal LP/QP (3.74)-(3.77) evidently;
- $u(t) \in \mathbb{R}^m$ is the dual decision variable vector corresponding to equality constraint (3.75);
- $v(t) \geq 0 \in \mathbb{R}^k$ is the dual decision variable vector corresponding to inequality constraint (3.76).

In [149], the definition of the projection operator is $P_\Omega(z(t)) = \left[P_{\Omega 1}(z_1(t)), P_{\Omega 2}(z_2(t)), \ldots, P_{\Omega n+m+k}(z_{n+m+k}(t))\right]^{\mathrm{T}}$. Such a projection operator is vector-input vector-output (or say, vector-valued), and projects from $z(t) \in \mathbb{R}^{n+m+k}$ onto set $\Omega = \{\mathbf{z}(t) | \varsigma^-(t) \leq \mathbf{z}(t) \leq \varsigma^+(t)\}$. Similarly, projection operator $P_\Omega(\cdot)$ is typically defined to be a function that is applied in an element-wise manner as follows:

$$P_{\Omega i}(z_i(t)) = \begin{cases} \varsigma^-(t), & z_i(t) < \varsigma^-(t) \\ z_i(t), & \varsigma^-(t) \leq z_i(t) \leq \varsigma^+(t), \\ \varsigma^+(t), & z_i(t) > \varsigma^+(t) \end{cases} \tag{3.78}$$

where $i = 1, 2, ..., m + n + k$.

Generalized LVI-PDNN Solution to LP/QP problems

The following dynamical system can be used to solve this TV QP problem

$$\dot{y}(t) = \gamma(I + H^{\mathrm{T}}(t))(P_\Omega(y(t) - (H(t)y(t) + h(t))) - y(t)), \tag{3.79}$$

where $P_\Omega(\cdot)$ is the projection operator (see [149]) and $\gamma > 0$ is known as the design parameter. Within hardware permission, the value of $\gamma >0$ should be set as the largest, or selected appropriately for simulation or experimental purposes.

While solving static QP problems, beginning with any $y(0) \in \mathbb{R}^{n+l+k}$ initial state, the LVI-PDNN state vector $y(t)$ converges to the equilibrium point y^*, wherein the first n elements are an optimal solution to the TVQP problems (3.74)-(3.77). Furthermore, the following inequality is true for the static QP's LVI-PDNN solution, [125]:

$$\left\| y - P_\Omega(y - (Hy + p)) \right\|_2^2 \geq \rho \left\| y - y^* \right\|_2^2, \tag{3.80}$$

where $\|\cdot\|_2$ corresponds to the vector's two-norm.

To gain a better understanding of LVI-PDNN's real-time convergence, the residual error is defined as

$$E(t) = y(t) - P_\Omega(y(t) - (H(t)y(t) + h(t))). \tag{3.81}$$

Based on the inequality (3.80), the convergence of the $y(t)$ state vector to the optimal $y^*(t)$ mathematical solution can be reached if $\|E(t)\|_2^2 \to 0$.

Convergence Analysis

In this subsection, we present, in a formal form, a convergence analysis of the LVI-PDNN model, based on the conceptual framework proposed in [125], by Zhang *et al.* We start with the static general problem, which handles LP/QP:

$$\min_{x} \quad x^{\mathrm{T}}Gx/2 + g^{\mathrm{T}}x \tag{3.82}$$

$$\text{subject to} \quad Dx = d \tag{3.83}$$

$$Bx \le b \tag{3.84}$$

$$\zeta^{-} \le x \le \zeta^{+}. \tag{3.85}$$

The proposed primal-dual NN from [125] could solve online (3.82)-(3.85) based on the equivalence of QP/LP, LVI and a system of piecewise linear equations. Then, in our case, equations (6), (7) from [125] can be reformulated as equations (3.86), (3.87), respectively, where:

$$y = \begin{bmatrix} x \\ \mu \\ \varrho \end{bmatrix}, \quad \varsigma^{-} = \begin{bmatrix} \zeta^{-} \\ -\varpi 1_{\nu} \\ 0 \end{bmatrix}, \quad \varsigma^{+} = \begin{bmatrix} \zeta^{+} \\ +\varpi 1_{\nu} \\ +\varpi 1_{\nu} \end{bmatrix}. \tag{3.86}$$

Here, ϖ represents a sufficiently large positive constant (or vector of suitable dimensions), and 1_{ν} denotes a vector of ones that is dimensioned appropriately. The coefficients in equation (5) are defined as [125]

$$H = \begin{bmatrix} G & -D^{\mathrm{T}} & B^{\mathrm{T}} \\ D & 0 & 0 \\ -B & 0 & 0 \end{bmatrix}, \qquad p = \begin{bmatrix} g \\ -d \\ b \end{bmatrix}. \tag{3.87}$$

In the following, we will use the same notation as in [125].

Theorem 3.6.3 ((LP/QP-LVI equivalence) [125], Theorem 1) *It is possible to reformulate the optimization problem* (3.82)-(3.85) *as: find a vector* $w^* \in \Omega$ *such that* $\forall\, w \in \Omega := \{w | \varsigma^{-} \le w \le \varsigma^{+}\} \subset \mathbb{R}^{n+l}$,

$$(w - w^*)^{\mathrm{T}}(Hw^* + p) \ge 0. \tag{3.88}$$

Theorem 3.6.4 ((PDNN convergence) [125], Theorem 2) *Starting from arbitrary initial state, the state vector* $w(t)$ *of the primal-dual NN* (3.79) *converges to the equilibrium* w^**, whose first* m *elements define the optimal solution* x^* *to the QP model* (3.82)-(3.85)*. In fact, the exponential convergence can be reached if there is a constant* $\rho > 0$ *satisfying*

$$\left\| w - P_{\Omega}(w - (Hw + p)) \right\|_2^2 \ge \rho \left\| w - w^* \right\|_2^2.$$

4

Intelligent Online Algorithms for Portfolio Analysis and Management

Continuing our previous work on time-varying (TV) financial optimization problems [3, 5], our research focuses on popular portfolio selection problems, which are the minimum-cost portfolio insurance (MCPI) problem, the Markowitz's mean-variance portfolio selection (MVPS) problem, the Black-Litterman portfolio optimization (BLPO) problem, and the tangency portfolio (TP) problem. These financial problems can be classified into two categories. The one category includes TV linear programming (LP) problems and TV quadratic programming (QP) problems, while the other category includes TV nonlinear programming (NLP) problems and TV integer linear programming (ILP) problems.

Creating TV versions of the aforementioned financial optimization problems and adding nonlinear constraints (NC), which refer to transaction costs (TC) and cardinality constraints (CC), the following TV financial optimization problems are presented in this chapter:

- the TV MCPI (TV-MCPI) problem [150, 151];
- the TV MVPS (TV-MVPS) problem [152, 153];
- the TV BLPO (TV-BLPO) problem [69];
- the multi-period (MP) MCPI under TC (MPMCPITC) problem [154];
- the TV MCPI under TC (TV-MCPITC) problem [155];
- the TV MVPS under TC and CC (TV-MVPSTC-CC) problem [156];

- the binary Markowitz-based portfolio selection (BMPS) problem [157];
- the TV TP under NC (TV-TPNC) problem [158].

More precisely, the TV-MCPI problem is defined and studied as a TV LP problem, while the TV-MVPS and the TV-BLPO problems are defined and studied as TV QP problems. In addition, the TV-MCPITC, TV-MVPSTC-CC, MPMCPITC and TV-TPNC problems are defined and studied as TV NLP problems, while the BMPS problem is defined and studied as a TV ILP problem.

Intelligent online optimization algorithms, which include modern neural network (NN) techniques and state-of-the-art metaheuristics, are employed to solve the aforementioned TV financial optimization problems using real-world datasets. More particularly, the TV LP/QP financial optimization problems are approached by two continuous-time (CT) NN solvers. These solvers are the zeroing NN (ZNN) and the linear-variational-inequality primal-dual NN (LVI-PDNN). Moreover, the TV NLP/ILP financial optimization problems are approached by popular metaheuristic optimization algorithms. These metaheuristic algorithms are variations of a nature inspired algorithm called Beetle Antennae Search (BAS), whose primary advantage is its low time consumption. The following diagram, in Fig. 4.1, connects the financial problems presented in this chapter with the mathematical methods that we intend to use to achieve our goal.

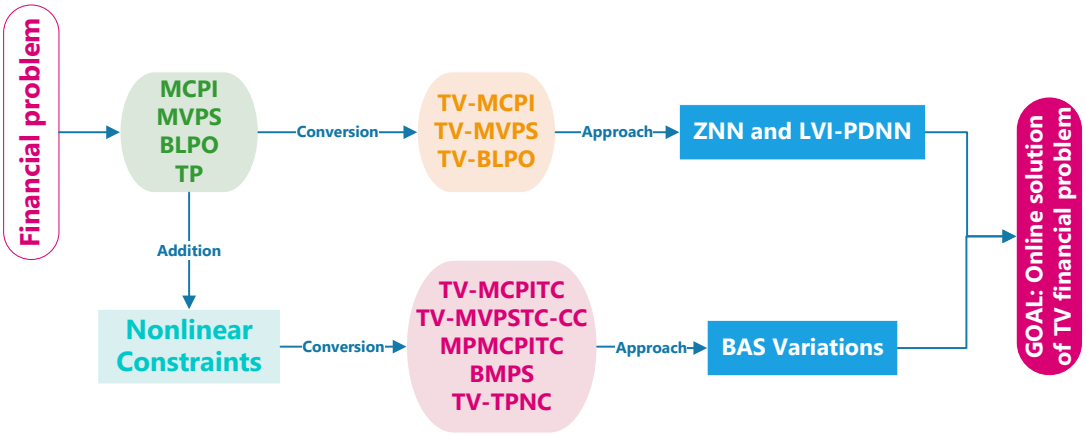


Figure 4.1: Flowchart depicting the connection between financial problems and mathematical methods.

The following are some of the chapter's general notations: the symbols $\mathbf{1}_n, \mathbf{0}_n$ denote a vector in $\mathbb{R}^n$ consisting of ones and zeros, respectively; $\mathbf{0}_{n\times n} \in \mathbb{R}^{n\times n}$ denotes a zero matrix of $n \times n$ dimensions; $I_n \in \mathbb{R}^{n\times n}$ denotes the identity $n \times n$ matrix; the operators $(\cdot)^{\mathrm{T}}$ and $(\cdot)^{\dagger}$ denote transposition and pseudo-inversion, respectively; $\odot$ denotes the Hadamard (or element-wise) product and the superscript $()^{\odot}$ denotes the Hadamard exponential; $\bar{x}$ denotes a square diagonal matrix with the elements of vector x on the main diagonal; zeros$(\cdot)$, mean$(\cdot)$, var$(\cdot)$ and cov$(\cdot)$ signify regular MATLAB routines.

4.1 The TV Minimum-Cost Portfolio Insurance Problem

This section has been organised in the following way. Subsection 4.1.1 describes the TV-MCPI problem, subsection 4.1.2 presents the ZNN's approach on the TV-MCPI problem and, in subsection 4.1.3, the LVI-PDNN's approach on the TV-MCPI problem is presented.

4.1.1 Minimum-Cost Portfolio Insurance

For financial models, reducing portfolio costs is always of major importance. One way to reduce portfolio costs is to minimize the cost of insurance (see [3, 43, 155, 156, 159, 160]). For instance, a TV optimization problem is defined in [3] for minimizing the cost of insurance in portfolios which constructs the portfolio that replicates a target payoff in a subset of states, in the case where the asset span is a lattice-subspace, and tackled with Riesz spaces theory. In [160], the author solve the problem of optimal expected growth in a random trade time model, taking into account the insurance of the portfolio in a low liquid market, to ensure the optimal constant proportion portfolio insurance approach in a simple form. Inhere, we present a TV analog of the corresponding static problem, which has been described and investigated in a number of papers, including [5, 161–164].

The continuous TV acceptation of the portfolio insurance problem introduced inhere is an innovative approach that integrates robust processes from NNs to provide online, thus more realistic, solution. With $C[a,b]$, we indicate the space of real-valued continuous functions specified at $[a,b]$ interval. The initial portfolio is a vector $\phi = [\phi_1, \phi_2, \ldots, \phi_n] \in \mathbb{R}^n$, where ϕ_i, $i = 1,2,\ldots,n$, is the number of shares of the ith security. The Euclidean space $\mathbb{R}^n$ is then referred to as portfolio space. If ϕ is a portfolio that is not zero, then its payoff is given by the formula

$$R(\phi) = \sum_{i=1}^{n} \phi_i x_i(t),$$

where $x_i(t) \in C[a,b]$, $i = 1,2,\ldots,n$, and $t \in [a,b] \subseteq [0,+\infty)$. The $R(\cdot)$ operator, called the payoff operator, is one-to-one, as defined in the previous formula. The payoff operator's range space is the vector subspace generated by the payoff vectors $x_i(t)$ in $C[a,b]$, which is referred to as $X(t) = [x_1(t), x_2(t), \ldots, x_n(t)]$. We shall refer to $X(t)$ as the space of marketed securities.

Let us suppose that $p(t) = [p_1(t), p_2(t), \ldots, p_n(t)]$ is a vector of TV security prices, where $p_i(t) \in C[a,b]$, $i = 1,2,\ldots,n$. Inhere, we presume that the costs of insurance of the portfolio include a standard price plus a risk rate of the assets. If the price rates related to the asset risk is denoted by δ and the fixed price by β, then the function of the fixed and linear insurance prices is composed as below:

$$p_i(t) = \beta + \delta \cdot \mathrm{Var}\Big[\frac{Y_i(t)}{\max(Y_i(t))}\Big],\ i = 1,2,\ldots,n, \tag{4.1}$$

where $Y_i(t) = [x_i(t), x_i(t-1), \ldots, x_i(t-c+1)]$, $\mathrm{Var}\big[Y_i(t)\big]$ denotes the variance of $Y_i(t)$. The number $c \leq t-1$, $c \in \mathbb{N}$, is a constant number and implies the time delays. Since $Y_i(t) \in \mathbb{R}^c$, it is the variance of its c in number normalized prices that measures the risk of the i asset. In this way, the insurance price of any asset relies on the level of risk it bears, where there is a possibility that the expected return will be different from the real return. Hence, the prices of insurance are becoming more realistic.

Considering that $\mathbf{k} = [k,k,\ldots,k] \in \mathbb{R}^n$ denotes a vector with n coordinates equal to $k \in \mathbb{R}$, the insured payoff

on the ϕ portfolio at the k "floor" and in the $p(t)$ price, is the supremum $R(\phi) \vee \mathbf{k}$. Consequently, given an initial portfolio ϕ and a floor $\mathbf{k}$, the TV minimum-cost insured portfolio $\eta(t) \in \mathbb{R}^n$ is the solution to the following cost minimization constrained problem:

$$\begin{aligned} \min_{\eta(t)} \quad & p^{\mathrm{T}}(t) \cdot \eta(t) \\ \text{s.t.} \quad & R(\eta(t)) \geq R(\phi) \vee \mathbf{k}. \end{aligned}$$

This problem can be written in TVLP format as well, by following [165], as bellow:

$$\min_{\eta(t)} \quad p^{\mathrm{T}}(t) \cdot \eta(t) \tag{4.2}$$

$$\text{s.t.} \quad -X^{\mathrm{T}}(t) \cdot \eta(t) \leq \min\{-X^{T}(t) \cdot \phi, -k\} \tag{4.3}$$

$$0 \leq \eta(t) \leq X^{\mathrm{T}}(t) \cdot \phi \cdot \left[\frac{1}{x_1(t)}, \ldots, \frac{1}{x_n(t)}\right], \tag{4.4}$$

where the right part of (4.4) is the maximum merit of each stock that an investor is allowed to keep at time t, by investing in each of them all the portfolio's ϕ payoff.

4.1.2 TV Minimum-Cost Portfolio Insurance Problem via ZNN

Bearing in mind its basic role in mathematical optimization, most aspects of LP have been thoroughly studied over the last decades. Solutions to LP problems have been commonly used in a wide range of research and industrial applications, see for example [132, 135, 166]. In the past, typically fewer than two different types of constraints are subject to LP problems studied by researchers [132]. However, most studies were dedicated to investigating LP problems on the basis of static coefficients, see [125], which means that methods and algorithms designed to address such a group of LP problems are inoperative when employed to TV situations.

To approach the TV-MCPI problem with an ZNN, we need to include the equations (4.2)-(4.4) to the coefficients of the ZNN in subsection 3.6.2. Consequently, we set the coefficients in (3.52)-(3.54) as follows:

$$W(t) = \mathbf{0}_{n\times n}, \quad q(t) = p(t), \quad K(t) = [\,], \quad \rho(t) = [\,], \quad x(t) = \eta(t),$$

$$A(t) = \begin{bmatrix} -X^{\mathrm{T}}(t) \\ I_n \\ -I_n \end{bmatrix} \in \mathbb{R}^{(2n+1)\times n}, \quad b(t) = \begin{bmatrix} \min\{-X^{T}(t) \cdot \phi, -k\} \\ X^{\mathrm{T}}(t) \cdot \phi \cdot \left[\frac{1}{x_1(t)}, \ldots, \frac{1}{x_n(t)}\right] \\ \mathbf{0}_n \end{bmatrix} \in \mathbb{R}^{2n+1}.$$

Following the implementation of stages 1 and 2 in subsection 3.6.2, stage 3 defines the following error matrix:

$$\begin{cases} E_1(t) = q(t) + A^{\mathrm{T}}(t)\lambda(t) \\ E_2(t) = A(t)x(t) + R^{\odot 2}(t) - b(t) \\ E_3(t) = \lambda(t) \odot R(t) \end{cases} \tag{4.5}$$

and then substituting $E_i(t), i = 1,2,3$, defined in (4.5) in place of $E(t)$ into (3.38), one obtains

$$\begin{cases} \dot{q}(t) + A^{\mathrm{T}}(t)\dot{\lambda}(t) + \dot{A}^{\mathrm{T}}(t)\lambda(t) = -\gamma E_1(t) \\ \dot{A}(t)x(t) + A(t)\dot{x}(t) + 2R(t)\dot{R}(t) - \dot{b}(t) = -\gamma E_2(t) \\ \dot{\lambda}(t)R(t) + \lambda(t)\dot{R}(t) = -\gamma E_3(t) \end{cases} \tag{4.6}$$

It is worth mentioning that

$$\dot{A}(t) = \begin{bmatrix} -\dot{X}^{\mathrm{T}}(t) \\ \mathbf{0}_n \\ \mathbf{0}_n \end{bmatrix} \in \mathbb{R}^{(2n+1)\times n}, \quad \dot{b}(t) = \begin{bmatrix} \min\{-\dot{X}^T(t)\cdot\phi, 0\} \\ \mathbf{0}_n \\ \mathbf{0}_n \end{bmatrix} \in \mathbb{R}^{2n+1}$$

Then setting

$$\begin{aligned} S(t) &= \begin{bmatrix} \mathbf{0}_{n\times n} & A^{\mathrm{T}}(t) & \mathbf{0}_{n\times(2n+1)} \\ A(t) & \mathbf{0}_{(2n+1)\times(2n+1)} & 2\bar{R}(t) \\ \mathbf{0}_{(2n+1)\times n} & \bar{R}(t) & \bar{\lambda}(t) \end{bmatrix}, \\ u(t) &= \begin{bmatrix} -\gamma E_1(t) - \dot{q}(t) - \dot{A}^{\mathrm{T}}(t)\lambda(t) \\ -\gamma E_2(t) - \dot{A}(t)x(t) + \dot{b}(t) \\ -\gamma E_3(t) \end{bmatrix}, \end{aligned} \tag{4.7}$$

where $S(t) \in \mathbb{R}^{(5n+2)\times(5n+2)}$ and $u(t) \in \mathbb{R}^{5n+2}$, (4.6) can be reformulated as follows:

$$S(t)\dot{y}(t) = u(t), \tag{4.8}$$

where $\dot{y}(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{\lambda}(t) \\ \dot{R}(t) \end{bmatrix} \in \mathbb{R}^{5n+2}$ and $S(t)$ is a nonsingular mass matrix. Note that the model of (4.8) is sufficient for solving the TV LP problem of (4.2)-(4.4), but because the creation of $p(t)$ and $X(t)$ in our approach is reliant on function handle in terms of MATLAB code, as will be shown in chapter 5, the following model of (4.9) would be easier to manage under an `ode` MATLAB solver. As a result, the proposed ZNN model for solving the TV LP problem of (4.2)-(4.4) is the following:

$$\dot{y}(t) = (S^{\mathrm{T}}(t)S(t))^{-1}S^{\mathrm{T}}(t)u(t), \tag{4.9}$$

which is equivalent of (4.8) except for costs of calculating the inverse of the matrix $S^{\mathrm{T}}(t)S(t)$. It is worth mentioning that the solution $y(t)$ of (4.9) can be efficiently generated by employing an `ode` MATLAB solver. The convergence of the ZNN solver to the unique theoretical solution is proven in the following theorem 4.1.1.

Theorem 4.1.1 *Starting from any initial state $y(0) \in \mathbb{R}^{5n+2}$, the state variable $y(t)$ of ZNN* (4.9) *globally converges to the unique theoretical solution $y^*(t)$ of TV LP problem of* (4.2)-(4.4). *That is,* $\lim_{t\to\infty}(y(t) - y^*(t)) = 0$.

Proof. In order to obtain the solution $x(t)$ of TV LP problem of (4.2)-(4.4), the error matrix equation is defined as

in (4.5), based on the ZNN design. After that, by adopting the linear design formula for zeroing (4.5), the model (4.6) is obtained. From [87, Theorem 1], the error matrix equation (4.6) converges to the theoretical solution when $t \to \infty$. As a result, the solution of (4.6) converges to the theoretical solution of TV LP problem of (4.2)-(4.4) when $t \to \infty$. In addition, from the derivation process of (4.9), we know that it is actually another form of (4.6). The proof is thus completed. □

4.1.3 TV Minimum-Cost Portfolio Insurance Problem via LVI-PDNN

The NN approach has been proved to be a powerful real-time computation instrument for over ten years due to the availability of hardware implementation and its parallel distributed computing nature [136–139, 141–143, 148]. For example, [136] presents a simple, piecewise-linear and global convergent approach to optimal solutions with dual NNs.

In [148], the authors introduced a primal-dual NN with a piecewise-linear dynamic, inline with linear variational inequality (LVI). In [143] one can find a LVI-PDNN with the ability to handle both linear and QP problems on the same/unified way. Globally, the LVI-PDNN converges to the optimum solution(s). In [165], LVI-PDNN was applied simultaneously to find real-time solutions to TV LP problems, liable to equality, inequality and boundary constraints.

To approach the TV-MCPI problem with an LVI-PDNN, we need to include the equations (4.2)-(4.4) to the coefficients of the LVI-PDNN in subsection 3.6.3. Consequently, we set the coefficients

$$H(t) = \begin{bmatrix} 0 & -X(t) \\ X^{\mathrm{T}}(t) & 0 \end{bmatrix}, \quad h(t) = \begin{bmatrix} p(t) \\ \min\{A(t) \cdot \phi, -k\} \end{bmatrix},$$

and the primal-dual decision vector $y(t)$, with the lower and upper boundaries to which it is liable, as follows:

$$y(t) = \begin{bmatrix} \eta(t) \\ \upsilon(t) \end{bmatrix}, \quad \varsigma^{-}(t) = \begin{bmatrix} \mathbf{0}_n \\ 0 \end{bmatrix}, \quad \varsigma^{+}(t) = \begin{bmatrix} X^{\mathrm{T}}(t) \cdot \phi \cdot \left[\frac{1}{x_1(t)}, \ldots, \frac{1}{x_n(t)}\right] \\ +\varpi \end{bmatrix},$$

where

- the constant $\varpi \gg 0$ is the numerical representation of $+\infty$, large enough for implementation purposes;
- $\varsigma^{-}(t) \leq y(t) \leq \varsigma^{+}(t)$ is the basic parameter decision vector of the primal TVLP (4.2)-(4.4);
- $\upsilon(t) \geq 0 \in \mathbb{R}$ is the dual decision variable vector of the inequality constraint (4.3).

The following dynamical system of (3.79) can be used to solve this TV LP problem:

$$\dot{y}(t) = \gamma(I + H^{\mathrm{T}}(t))(P_{\Omega}(y(t) - (H(t)\,y(t) + h(t))) - y(t)), \tag{4.10}$$

where $\gamma > 0$ is known as the design parameter, and $P_{\Omega}(\cdot)$ is the projection operator as declared in (3.78), defined as Note that the solution $y(t)$ of (4.10) can be efficiently generated by employing an `ode` MATLAB solver.

4.2 The TV Mean-Variance Portfolio Selection Problem

In finance terms, a collection of all stocks or assets held by a public or private institute is known as a portfolio. The portfolio selection problem refers to the optimal distribution of budget on the available stocks such that the expected mean-return is maximized (profit), and the risk is minimized. The factor to measure risk is the variance of the portfolio return, smaller the variance lower will be the risk. This approach was introduced few decades ago by Markowitz's modern portfolio theory [167]. The modern portfolio theory also assumes a perfect market without taxes or transaction costs where short sales are disallowed, but securities are infinitely divisible and can therefore be traded in any (non-negative) fraction.

Over the last decades the Markowitz's modern portfolio theory is studied extensively such as in [2, 168–171]. For example, the authors in [169] investigate a problem of CT mean-variance portfolio selection with stochastic parameters under a no-bankruptcy limit. In [171], the problem of dynamic portfolio selection is conceived as a Markowitz problem of optimizing mean-variance. They conclude that the single-period Markowitz QP algorithm can be used with appropriate modifications in the covariance and linear constraint matrices to solve the problem of multi-period asset allocation.

4.2.1 Definition of the TV-MVPS Financial Problem

Our approach to the mean-variance portfolio selection (MVPS) problem is a TV analog of the corresponding static problem defined and studied in a number of papers, such as [2, 167–172]. The MVPS is a financial optimization problem for assembling a portfolio of assets such that its risk is minimized under a target expected return. As far as we are aware of, our TV version of the mean-variance portfolio selection (TV-MVPS) problem is a novel approach that comprises robust techniques from NNs to provide online, thus more realistic, solution.

The space of marketed securities is $X = [x_1, x_2, \dots, x_n] \in \mathbb{R}^{m\times n}$ where $x_i \in \mathbb{R}^m$ is the security i, $i = 1, 2, \dots, n$, and comprises from the last m observations of its price. In the static MVPS problem the expected return of the marketed space is $r = [r_1, r_2, \dots, r_n] \in \mathbb{R}^n$ where $r_i = \sum_{j=1}^{m} x_i(j)/m \in \mathbb{R}$ is the expected return of the security i, $i = 1, 2, \dots, n$. The expected return of the portfolio is $r_p \in [\min(r), \max(r)] \subseteq \mathbb{R}$ and the variance of the marketed space is $\sigma^2 = \sum_{i=1}^{n}\sum_{i=1}^{n} x_i x_j \sigma_{ij}$ where $\sigma_{ij} = \rho_{ij}\sigma_i\sigma_j$ is the variance and ρ_{ij} is the correlation of i and j securities and σ_i is the variance of i security. That is, $\sigma^2 = X^{\mathrm{T}}CX$ where $C \in \mathbb{R}^{n\times n}$ is the covariance matrix of the marketed space X.

In the TV-MVPS we define the number $\tau \leq m - 1$, $\tau \in \mathbb{N}$, where τ is a constant number and it denotes the 'number of time periods'. The τ is used for the calculation of the simple moving average. A moving average (MA) is a calculation for analyzing data points by creating a series of averages of the complete data set of different subsets. In technical analysis of financial data such as stock prices, returns or volumes of trading the moving average is used as a technical indicator that combines price points of an instrument over a specified time frame divided by the number of data points τ in order to give a single trend line. Hence, a moving average is primarily a lagging indicator and, for that reason, it is one of the most popular tools for technical analysis. The unweighted mean of the previous τ data is called simple moving average (SMA). For the observation prices $x_i(t+1), x_i(t+2), \dots, x_i(t+1+\tau)$ of the security i, $i = 1, 2, \dots, n$, the formula of the simple moving average is $SMA_{t+1} = \sum_{j=t+1}^{t+1+\tau} x_i(j)/\tau$. In the case where evaluating consecutive values and a new value, $x_i(t)$, comes into the calculation, the oldest value, $x_i(t+1+\tau)$, drops out. That is, $SMA_t = SMA_{t+1} + (x_i(t) - x_i(t+1+\tau))/\tau$. The

chosen period depends on the type of interest movement, for example short, moderate, or long-term. Short-term averages respond quickly to changes in the price of the underlying, while long-term averages are slow to react. Moving average levels can be viewed in financial terms as support in a falling market or resistance in a rising market. In general, there exist several types of moving averages (see [173]). In this section, we use only one type, the simple moving average (*SMA*). All the rest types of moving averages can be applied to TV-MVPS similarly to *SMA*.

The TV-MVPS comprises from $m-\tau$ in number consecutive values of an *MA* with τ in number observations for each time period. The time $t \in [1, m-\tau]$ denotes the new value that it comes into the calculation of the *MA*. Hence, the expected return of the marketed space is $r(t) = [r_1(t), r_2(t), \ldots, r_n(t)] \in \mathbb{R}^n$ where

$$r_i(t) = \sum_{j=t}^{t+\tau} x_i(j)/\tau \in \mathbb{R}$$

is the expected return of the security i, $i = 1, 2, \ldots, n$. Obviously, the $r_i(t)$ is an *SMA* and the TV-MVPS problem is built-up on the $r(t)$ for every t. So, the expected return of the portfolio is $r_p(t) \in [\min(r(t)), \max(r(t))] \subseteq \mathbb{R}$, the variance of the marketed space is

$$\sigma^2(t) = \sum_{i=1}^{n} \sum_{i=1}^{n} x_i(t : t+\tau) x_j(t : t+\tau) \sigma_{ij}(t),$$

where $\sigma_{ij}(t) = \rho_{ij}(t)\sigma_i(t)\sigma_j(t)$ is the variance and $\rho_{ij}(t)$ is the correlation of $x_i(t : t+\tau)$ and $x_j(t : t+\tau)$ and $\sigma_i(t)$ is the variance of $x_i(t : t+\tau)$. That is,

$$\sigma^2(t) = X(t : t+\tau, :)^{\mathrm{T}} C(t) X(t : t+\tau, :),$$

where $C(t) = cov(X(t : t+\tau, :)) \in \mathbb{R}^{n \times n}$ is the covariance matrix of the marketed space $X(t : t+\tau, :)$ at time t. The optimal mean-variance portfolio is $\eta(t) = [\eta_1(t), \eta_2(t), \ldots, \eta_n(t)]$ where $\eta_i(t)$ is the solution of subsection's 4.2.1 or 4.2.1 optimization problem for the security i, $i = 1, 2, \ldots, n$.

The purpose of the number τ is to keep steady the number of observations in the TV-MVPS for each t in $X(t : t+\tau, :)$ while t is moving across the interval $[1, m-\tau]$. Hence, the outcome of the TV-MVPS for each t can be comparable with all the rest outcomes of every other $t \in [1, m-\tau]$ under the same number of observations. Note that, the expected return of the marketed space $r_i(t)$ is a *MA* and it also has the properties of a *MA*. That is, the bigger the τ of the TV-MVPS is the smoother the $r_i(t)$ will be when t is moving across the interval $[1, m-\tau]$, because it filters out the 'noise' from random short-term price fluctuations. Moreover, it affects the optimal mean-variance portfolio $\eta(t)$ in the same way.

We convert the discrete TV-MVPS problem to CT by interpolated the $r(t)$ and the $C(t)$ into continuous functions with any method of preferences. Consequently, $r(t),\ C(t) \in C[0, m-\tau-1]$ where the space $C[0, m-\tau-1]$ is the space of all continuous real functions on the interval $[0, m-\tau-1]$. The optimal mean-variance portfolio is $\eta(t) = [\eta_1(t), \eta_2(t), \ldots, \eta_n(t)]$ where $\eta_i(t)$ is the online solution of subsection's 4.2.1 or 4.2.1 optimization problem produced by the LVI-PDNN of section 4.2.3.

TV-MVPS with specific expected return target

The TV mean-variance portfolio selection for a specific target r_p is the solution to the following risk minimization and expected return maximization constrained problem:

$$\min_{\eta(t)} \quad \sum_i \sum_j \eta_i(t) \cdot \eta_j(t) \cdot \sigma_{ij}(t) \tag{4.11}$$

$$\text{s.t.} \quad \sum_i \eta_i(t) \cdot r_i(t) = r_p(t) \tag{4.12}$$

$$\sum_i \eta_i(t) = 1 \tag{4.13}$$

$$\eta_i(t) \in \mathbb{R}_0^+, \ \forall i, \tag{4.14}$$

where (4.11) is the variance $\sigma^2(t)$ of the portfolio $\eta(t)$.

This problem can also be written in the TV QP (TVQP) problem form, by following [149], as follows:

$$\min_{\eta(t)} \quad \eta^{\mathrm{T}}(t) \cdot C(t) \cdot \eta(t) \tag{4.15}$$

$$\text{s.t.} \quad [\mathbf{1}_n \ \ r(t)]^{\mathrm{T}} \cdot \eta(t) = [1 \ \ r_p(t)]^{\mathrm{T}} \tag{4.16}$$

$$\mathbf{0}_n \leq \eta(t) \leq \mathbf{1}_n, \tag{4.17}$$

where $C(t)$ is the covariance matrix of $X(t)$.

TV-MVPS with all possible expected returns

In addition, the TV mean-variance portfolio selection for all possible targets r_p (see [174]) is the solution to the following risk minimization and expected return maximization constrained problem:

$$\min_{\eta(t)} \quad \sum_i \sum_j \eta_i(t) \cdot \eta_j(t) \cdot \sigma_{ij}(t) \tag{4.18}$$

$$\text{s.t.} \quad \sum_i \eta_i(t) \cdot r_i(t) \geq r_p(t) \tag{4.19}$$

$$\sum_i \eta_i(t) = 1 \tag{4.20}$$

$$\eta_i(t) \in \mathbb{R}_0^+, \ \forall i, \tag{4.21}$$

where (4.18) is the variance $\sigma^2(t)$ of the portfolio $\eta(t)$.

By following [149], this problem can also be written in the TVQP problem form as follows:

$$\min_{\eta(t)} \quad \eta^{\mathrm{T}}(t)\cdot C(t)\cdot \eta(t) \tag{4.22}$$

$$\text{s.t.} \quad -r^{\mathrm{T}}(t)\cdot \eta(t) \le -r_p(t) \tag{4.23}$$

$$\mathbf{1}_n^{\mathrm{T}}\cdot \eta(t) = 1 \tag{4.24}$$

$$\mathbf{0}_n \le \eta(t) \le \mathbf{1}_n. \tag{4.25}$$

4.2.2 TV Mean-Variance Portfolio Selection Problem via ZNN

In this subsection, the TV-MVPS problem with specific expected return target, as well as with all possible expected return targets, is approached by the ZNN solver.

TV-MVPS problem with specific expected return target via ZNN

To approach the TV-MVPS problem with specific expected return target into an ZNN, we need to include the equations (4.15)-(4.17) to the coefficients of the ZNN in subsection 3.6.1. Consequently, we set the coefficients in (3.52)-(3.54) as follows:

$$W(t) = C(t), \quad q(t) = [\,], \quad K(t) = [\mathbf{1}_n \; r(t)]^{\mathrm{T}}, \quad \rho(t) = [1 \; r_p(t)]^{\mathrm{T}}, \quad x(t) = \eta(t),$$
$$A(t) = \begin{bmatrix} I_n \\ -I_n \end{bmatrix} \in \mathbb{R}^{(2n)\times n}, \quad b(t) = \begin{bmatrix} \mathbf{1}_n \\ \mathbf{0}_n \end{bmatrix} \in \mathbb{R}^{2n}.$$

Following the implementation of stages 1 and 2 in subsection 3.6.1, stage 3 defines the following error matrix equation group:

$$\begin{cases} E_1(t) = W(t)x(t) + P_x(t) + K^{\mathrm{T}}(t)\lambda(t) \\ E_2(t) = K(t)x(t) - \rho(t) \end{cases} \tag{4.26}$$

and then substituting $E_i(t), i = 1,2$, defined in (4.26) in place of $E(t)$ into (3.38), one obtains

$$\begin{cases} \dot{W}(t)x(t) + W(t)\dot{x}(t) + \dot{P}_x(t) + K^{\mathrm{T}}(t)\dot{\lambda}(t) + \dot{K}^{\mathrm{T}}(t)\lambda(t) = -\gamma E_1(t) \\ \dot{K}(t)x(t) + K(t)\dot{x}(t) - \dot{\rho}(t) = -\gamma E_2(t) \end{cases} \tag{4.27}$$

where $\dot{P}_x(t) = P_1(t)\dot{x}(t) + P_2(t)x(t) + P_3(t)$ with the penalty functions $P_i(t), i = 1,2,3$, as presented in (3.61) and $N_i(t) = b_i(t) - A_i(t)x(t)$, $i = 1,2,\ldots,2n$. It is worth mentioning that $\dot{A}(t) = \mathbf{0}_{(2n)\times n}$, $\dot{b}(t) = \mathbf{0}_{2n}$. As a result, (4.27) can be reformulated as follows:

$$\begin{cases} (W(t) + P_1(t))\,\dot{x}(t) + K^{\mathrm{T}}(t)\dot{\lambda}(t) = -\gamma E_1(t) - \dot{W}(t)x(t) - \dot{K}^{\mathrm{T}}(t)\lambda(t) - P_2(t)x(t) - P_3(t) \\ K(t)\dot{x}(t) = -\gamma E_2(t) - \dot{K}(t)x(t) + \dot{\rho}(t) \end{cases} \tag{4.28}$$

Then setting

$$S(t) = \begin{bmatrix} W(t) + P_1(t) & K^{\mathrm{T}}(t) \\ K(t) & \mathbf{0}_{2\times 2} \end{bmatrix}, \quad u(t) = \begin{bmatrix} -\gamma E_1(t) - \dot{W}(t)x(t) - \dot{K}^{\mathrm{T}}(t)\lambda(t) - P_2(t)x(t) - P_3(t) \\ -\gamma E_2(t) - \dot{K}(t)x(t) + \dot{\rho}(t) \end{bmatrix}, \tag{4.29}$$

where $S(t) \in \mathbb{R}^{(n+2)\times(n+2)}$ and $u(t) \in \mathbb{R}^{n+2}$, (4.28) can be reformulated as follows:

$$S(t)\dot{y}(t) = u(t), \tag{4.30}$$

where $\dot{y}(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{\lambda}(t) \end{bmatrix} \in \mathbb{R}^{n+2}$ and $S(t)$ is a nonsingular mass matrix. Note that the model of (4.30) is sufficient for solving the TV QP problem of (4.15)-(4.17), but because the creation of $C(t)$ and $r(t)$ in our approach is reliant on function handle in terms of MATLAB code, as will be shown in chapter 5, the following model of (4.31) would be easier to manage under an `ode` MATLAB solver. As a result, the proposed ZNN model for solving the TV QP problem of (4.15)-(4.17) is the following:

$$\dot{y}(t) = S^{-1}(t)u(t), \tag{4.31}$$

which is equivalent of (4.30) except for costs of calculating the inverse of the matrix $S(t)$. It is worth mentioning that the solution $x(t)$ of (4.31) can be efficiently generated by employing an `ode` MATLAB solver. The convergence of the ZNN solver to the unique theoretical solution is proven in the following theorem 4.2.1.

Theorem 4.2.1 *Starting from any initial state $y(0) \in \mathbb{R}^{n+2}$, the state variable $y(t)$ of ZNN* (4.31) *globally converges to the unique theoretical solution $y^*(t)$ of TV QP problem of* (4.15)-(4.17). *That is, $\lim_{t\to\infty}(y(t) - y^*(t)) = 0$. Furthermore, the theoretical solution $x^*(t)$ of TV QP problem of* (4.15)-(4.17) *is the first n elements of $y^*(t)$.*

Proof. In order to obtain the solution $x(t)$ of TV QP problem of (4.15)-(4.17), the error matrix equation is defined as in (4.26), based on the ZNN design. After that, by adopting the linear design formula for zeroing (4.26), the model (4.27) is obtained. From [87, Theorem 1], each error matrix equation in the error matrix equation group (4.27) converges to the theoretical solution when $t \to \infty$. As a result, the solution of (4.27) converges to the theoretical solution of TV QP problem of (4.15)-(4.17) when $t \to \infty$. In addition, from the derivation process of (4.31), we know that it is actually another form of (4.27). The proof is thus completed. □

TV-MVPS problem with all possible expected return targets via ZNN

To approach the TV-MVPS problem with all possible expected return targets into an ZNN, we need to include the equations (4.22)-(4.25) to the coefficients of the ZNN in subsection 3.6.1. Consequently, we set the coefficients in (3.52)-(3.54) as follows:

$$W(t) = C(t), \quad q(t) = [\,], \quad K(t) = \mathbf{1}_n^{\mathrm{T}}, \quad \rho(t) = 1, \quad x(t) = \eta(t),$$
$$A(t) = \begin{bmatrix} -r^{\mathrm{T}}(t) \\ I_n \\ -I_n \end{bmatrix} \in \mathbb{R}^{(2n+1)\times n}, \quad b(t) = \begin{bmatrix} -r_p(t) \\ \mathbf{1}_n \\ \mathbf{0}_n \end{bmatrix} \in \mathbb{R}^{2n+1}.$$

Following the implementation of stages 1 and 2 in subsection 3.6.1, stage 3 defines the following error matrix equation group:

$$\begin{cases} E_1(t) = W(t)x(t) + P_x(t) + K^{\mathrm{T}}(t)\lambda(t) \\ E_2(t) = K(t)x(t) - \rho(t) \end{cases} \tag{4.32}$$

and then substituting $E_i(t), i = 1,2$, defined in (4.32) in place of $E(t)$ into (3.38), one obtains

$$\begin{cases} \dot{W}(t)x(t) + W(t)\dot{x}(t) + \dot{P}_x(t) + K^{\mathrm{T}}(t)\dot{\lambda}(t) + \dot{K}^{\mathrm{T}}(t)\lambda(t) = -\gamma E_1(t) \\ \dot{K}(t)x(t) + K(t)\dot{x}(t) - \dot{\rho}(t) = -\gamma E_2(t) \end{cases} \tag{4.33}$$

where $\dot{P}_x(t) = P_1(t)\dot{x}(t) + P_2(t)x(t) + P_3(t)$ with the penalty functions $P_i(t), i = 1,2,3$, as presented in (3.61) and $N_i(t) = b_i(t) - A_i(t)x(t)$, $i = 1,2,\ldots,2n+1$. It is worth mentioning that

$$\dot{A}(t) = \begin{bmatrix} -\dot{r}^{\mathrm{T}}(t) \\ \mathbf{0}_n \\ \mathbf{0}_n \end{bmatrix}, \quad \dot{b}(t) = \begin{bmatrix} -\dot{r}_p(t) \\ \mathbf{0}_n \\ \mathbf{0}_n \end{bmatrix}, \quad \dot{K}(t) = \mathbf{0}_n^{\mathrm{T}}, \quad \dot{\rho}(t) = 0.$$

As a result, (4.33) can be reformulated as follows:

$$\begin{cases} \left(W(t) + P_1(t)\right)\dot{x}(t) + K^{\mathrm{T}}(t)\dot{\lambda}(t) = -\gamma E_1(t) - \dot{W}(t)x(t) - P_2(t)x(t) - P_3(t) \\ K(t)\dot{x}(t) = -\gamma E_2(t) \end{cases} \tag{4.34}$$

Then setting

$$S(t) = \begin{bmatrix} W(t) + P_1(t) & K^{\mathrm{T}}(t) \\ K(t) & 0 \end{bmatrix}, \quad u(t) = \begin{bmatrix} -\gamma E_1(t) - \dot{W}(t)x(t) - P_2(t)x(t) - P_3(t) \\ -\gamma E_2(t) \end{bmatrix}, \tag{4.35}$$

where $S(t) \in \mathbb{R}^{(n+1)\times(n+1)}$ and $u(t) \in \mathbb{R}^{n+1}$, (4.28) can be reformulated as follows:

$$S(t)\dot{y}(t) = u(t), \tag{4.36}$$

where $\dot{y}(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{\lambda}(t) \end{bmatrix} \in \mathbb{R}^{n+1}$ and $S(t)$ is a nonsingular mass matrix. Note that the model of (4.36) is sufficient for solving the TV QP problem of (4.22)-(4.25), but because the creation of $C(t)$ and $r(t)$ in our approach is reliant on function handle in terms of MATLAB code, as will be shown in chapter 5, the following model of (4.37) would be easier to manage under an `ode` MATLAB solver. As a result, the proposed ZNN model for solving the TV QP problem of (4.22)-(4.25) is the following:

$$\dot{y}(t) = S^{-1}(t)u(t), \tag{4.37}$$

which is equivalent of (4.36) except for costs of calculating the inverse of the matrix $S(t)$. It is worth mentioning that the solution $y(t)$ of (4.37) can be efficiently generated by employing an `ode` MATLAB solver. The convergence

of the ZNN solver to the unique theoretical solution is proven in the following theorem 4.2.2.

Theorem 4.2.2 *Starting from any initial state $y(0) \in \mathbb{R}^{n+1}$, the state variable $y(t)$ of ZNN (4.31) globally converges to the unique theoretical solution $y^*(t)$ of TV QP problem of (4.22)-(4.25). That is, $\lim_{t\to\infty}(y(t)-y^*(t))=0$. Furthermore, the theoretical solution $x^*(t)$ of TV QP problem of (4.22)-(4.25) is the first n elements of $y^*(t)$.*

Proof. In order to obtain the solution $x(t)$ of TV QP problem of (4.22)-(4.25), the error matrix equation is defined as in (4.32), based on the ZNN design. After that, by adopting the linear design formula for zeroing (4.32), the model (4.33) is obtained. From [87, Theorem 1], each error matrix equation in the error matrix equation group (4.33) converges to the theoretical solution when $t \to \infty$. As a result, the solution of (4.33) converges to the theoretical solution of TV QP problem of (4.22)-(4.25) when $t \to \infty$. In addition, from the derivation process of (4.37), we know that it is actually another form of (4.33). The proof is thus completed. □

4.2.3 TV Mean-Variance Portfolio Selection Problem via LVI-PDNN

Regarding its fundamental role in mathematical optimization, over the past decades, most aspects of QP have been thoroughly studied. Several methods/algorithms for solving the fundamental static QP problem have been proposed [131]. Such a QP problem has two common general type of solutions. One general type of solution is the numerical algorithms conducted on digital computers and was commonly used to solve static QP problems on a small scale. Nevertheless, in the case of large-scale real-time applications, these numerical algorithms can lead to a decline in performance due to their serial-processing nature [175]. Commonly the less the arithmetic operations are, the less computationally expensive the cube of the Hessian matrix dimension m will be. The other general type of solution is the application of parallel processing which has driven the algorithmic development [176]. Therefore, the comprehensive and thorough research of the recurrent NN (RNN) has developed and investigated various dynamic and analog solvers. The approximation by neural-dynamic is now considered one of the strong alternatives to QP problems in real-time computing, due to its parallel distributed nature and easiness of hardware implementation [177].

TV-MVPS problem with specific expected return target via LVI-PDNN

To convert the TV-MVPS problem with specific expected return target (SERT) into an LVI-PDNN, we need to include the equations (4.15)-(4.17) to the coefficients of the LVI-PDNN from [149]. According to the TVQP problem of subsection 4.2.1, if we set

- $G(t) = C(t)$
- $B(t) = b(t) = [\,]$
- $D(t) = [\mathbf{1}_n \;\; r(t)]^{\mathrm{T}}$
- $\zeta^-(t) = \mathbf{0}_n$
- $x(t) = \eta(t)$
- $g(t) = [\,]$
- $d(t) = [1 \;\; r_p(t)]^{\mathrm{T}}$
- $\zeta^+(t) = \mathbf{1}_n$,

then the coefficients of the LVI-PDNN can be written as follows:

$$H(t) = \begin{bmatrix} G(t) & -D^{\mathrm{T}}(t) \\ D(t) & \mathbf{0}_{2\times 2} \end{bmatrix}, \quad h(t) = \begin{bmatrix} g(t) \\ -d(t) \end{bmatrix}.$$

Furthermore, the definition of the primal-dual decision vector $y(t)$ can be written as follows, along with the lower and upper boundaries to which it is subject:

$$y(t) = \begin{bmatrix} x(t) \\ \mu(t) \end{bmatrix} \quad \varsigma^{-}(t) = \begin{bmatrix} \zeta^{-}(t) \\ -\varpi \mathbf{1}_l \end{bmatrix}, \quad \varsigma^{+}(t) = \begin{bmatrix} \zeta^{+}(t) \\ +\varpi \mathbf{1}_l \end{bmatrix}, \text{ where}$$

- the constant $\varpi \gg 0$ is the numerical representation and $+\infty$ replacement, large enough for implementation purposes;
- $\zeta^{-}(t) \leq x(t) \leq \zeta^{+}(t)$ clearly denotes the basic parameter decision vector of the primal TVQP (4.15)-(4.17);
- $\mu(t) \in \mathbb{R}^l$ is the dual decision variable vector of the equality constraint (4.16).

TV-MVPS problem with all possible expected return targets via LVI-PDNN

To convert the TV-MVPS problem with all possible expected return targets (APERT) into an LVI-PDNN, we need to include the equations (4.22)-(4.25) to the coefficients of the LVI-PDNN from [149]. According to the TVQP problem of subsection 4.2.1, if we set

- $G(t) = 2C(t)$
- $d(t) = 1$
- $g(t) = [\,]$
- $x(t) = \eta(t)$
- $B(t) = -r^{\mathrm{T}}(t)$
- $\zeta^{-}(t) = \mathbf{0}_n$
- $D(t) = \mathbf{1}_n^{\mathrm{T}}$
- $b(t) = -r_p(t)$
- $\zeta^{+}(t) = \mathbf{1}_n$

then the coefficients of the LVI-PDNN can be written as follows:

$$H(t) = \begin{bmatrix} G(t) & -D^{\mathrm{T}}(t) & B^{\mathrm{T}}(t) \\ D(t) & \mathbf{0}_{n \times n} & \mathbf{0}_n \\ -B(t) & \mathbf{0}_n^{\mathrm{T}} & 0 \end{bmatrix}, \quad h(t) = \begin{bmatrix} g(t) \\ -d(t) \\ b(t) \end{bmatrix}.$$

Furthermore, the definition of the primal-dual decision vector $y(t)$ can be written as follows, along with the lower and upper boundaries to which it is subject:

$$y(t) = \begin{bmatrix} x(t) \\ \mu(t) \\ \varrho(t) \end{bmatrix}, \quad \varsigma^{-}(t) = \begin{bmatrix} \zeta^{-}(t) \\ -\varpi \mathbf{1}_l \\ \mathbf{0}_k \end{bmatrix}, \quad \varsigma^{+}(t) = \begin{bmatrix} \zeta^{+}(t) \\ +\varpi \mathbf{1}_l \\ +\varpi \mathbf{1}_k \end{bmatrix}, \text{ where}$$

- the constant $\varpi \gg 0$ is the numerical representation and $+\infty$ replacement, large enough for implementation purposes;
- $\zeta^{-}(t) \leq x(t) \leq \zeta^{+}(t)$ clearly denotes the basic parameter decision vector of the primal TVQP (4.22)-(4.25);
- $\mu(t) \in \mathbb{R}^l$ is the dual decision variable vector of the equality constraint (4.24).
- $\varrho(t) \in \mathbb{R}^k$ is the dual decision variable vector of the inequality constraint (4.23).

4.2.4 Generalized LVI-PDNN Solution

The following dynamical system can be used to solve these TV QP problems:

$$\dot{y}(t) = \gamma(I + H^{\mathrm{T}}(t))(P_{\Omega}(y(t) - (H(t)y(t) + h(t))) - y(t)), \tag{4.38}$$

where $P_{\Omega}(\cdot)$ is the projection operator (see [149]) and $\gamma > 0$ is known as the design parameter. Note that the solution $y(t)$ of (4.38) can be efficiently generated by employing an `ode` MATLAB solver.

4.3 The TV Black-Litterman Portfolio Optimization Problem

It has been argued that the mean-variance (MV) optimal portfolio is nonsensical. Small changes in expected returns (ERs) input into an optimization solver may often result in large swings in portfolio positions, which results in asset weightings that are far too high in some cases. Because the Black-Litterman (BL) model uses a Bayesian analytic framework to process data [178], the portfolio manager merely has to construct views in this framework, and the model will convert these into asset return estimates [179]. As a result, the BL model is intriguing in theory and practice as a portfolio development tool and has been intensively examined during the previous few decades [70, 180, 181].

4.3.1 Exploiting the Black-Litterman Framework

This subsection goes over the definition of the CT BL portfolio optimization (CTBLPO or TV-BLPO) problem in extensive detail.

Discrete-Time MV portfolio selection Problem

From both a practical and theoretical standpoint, the MV portfolio selection (MPS) problem [2, 156, 158, 167, 172, 182] is significant. The MPS is a financial optimization problem that involves organizing the portfolio's assets in such a way that the portfolio's risk is minimized while meeting a target ER.

Consider the market space $M(t) = [\mu_1(t), \mu_2(t), \ldots, \mu_n(t)] \in \mathbb{R}^n$ where $\mu_i(t) \in \mathbb{R}$ is the asset i, $i = 1, 2, \ldots, n$, return at time $t = 1, 2, \ldots, m$. Considering the number $\theta \in \mathbb{N}$, that signifies the delays (or past values), we set $p(t) = [p_1(t), p_2(t), \ldots, p_n(t)] \in \mathbb{R}^n$ the market's space ER, where $p_i(t) = \sum_{k=0}^{\theta-1} \mu_i(t-k)/\theta \in \mathbb{R}$ is the asset's i, $i = 1, 2, \ldots, n$, ER at time t. Thus, $p_i(t)$ becomes a simple moving average (SMA) [173], i.e., the unweighted mean of the past θ data. A moving average is generally a lagging indicator that is one of the most often used technical analysis tools. Moreover, $C(t) \in \mathbb{R}^{n \times n}$ signifies the market's space covariance matrix, and $p_g(t) \in [\min(p(t)), \max(p(t))] \subseteq \mathbb{R}$ is the portfolio's target ER. As a result, for all the feasible targets p_g [152, 174], the discrete-time MPS (DTMPS) problem may be formulated as a TVQP problem as below:

$$\min_{\eta(t)} \quad \eta^{\mathrm{T}}(t) \cdot C(t) \cdot \eta(t) \tag{4.39}$$

$$\text{s.t.} \quad -p^{\mathrm{T}}(t) \cdot \eta(t) \leq -p_g(t) \tag{4.40}$$

$$\mathbf{1}_n^{\mathrm{T}} \cdot \eta(t) \leq 1, \tag{4.41}$$

$$\mathbf{0}_n \leq \eta(t) \leq \mathbf{1}_n, \tag{4.42}$$

where $\eta(t) \in \mathbb{R}^n$ signifies the optimal portfolio.

Continuous-Time BL Portfolio Optimization Problem

Markowitz's MV method gives a solution to the DTMPS problem only after the assets' ERs and covariances have been determined. Despite MPS is a huge theoretical breakthrough, it has run into a snag in practice: while the covariances of a few assets can be estimated accurately, reasonable ERs estimations are tough to come by. By not requiring the investor to input estimates of ER, BL avoids this problem. It presupposes that the initial ERs are required so that the distribution of equilibrium assets matches what we observe in the markets. It is solely up to the investor to state how his or her ERs differ from that of the markets, as well as his or her level of confidence in the various hypotheses. The BL approach uses this information to determine the best asset allocation.

The inclusion of investor information is at the basis of the BL model. These perspectives may vary from the assumed equilibrium returns, and they are now permitted to interact in the portfolio selection problem. As a result, a new diversified vector of ERs emerges. It is worth mentioning that a neutral equilibrium portfolio is used in the BL model for previous ERs. The concept is based on the General Equilibrium principle, which states that if the entire portfolio is in equilibrium, then any substitute portfolio must be as well. It may be employed with any utility function, giving it a wide range of applications. In reality, practitioners frequently use the Quadratic Utility (QU) function and assume a risk-free asset, reducing the equilibrium model to the Capital Asset Pricing Model (CAPM). The CAPM market's portfolio is the neutral portfolio in this case.

In this study, employing the QU function, unconstrained MV, and CAPM as provided in [183], the equilibrium excess returns $r(t)$ can be calculated as follows:

$$r(t) = \zeta(t) C(t) u(t), \tag{4.43}$$

where $u(t)$ is a vector of investment weights for each asset and $\zeta(t)$ signifies the risk aversion parameter. Considering $u(t)$ as the solution of the following TVQP problem:

$$\min_{u(t)} \quad u^{\mathrm{T}}(t) \cdot C(t) \cdot u(t) \tag{4.44}$$

$$\text{s.t.} \quad \mathbf{1}_n^{\mathrm{T}} \cdot u(t) = 1 \tag{4.45}$$

$$\mathbf{0}_n \leq u(t) \leq \mathbf{1}_n, \tag{4.46}$$

the following formula is used to calculate the risk aversion parameter:

$$\zeta(t) = \frac{\mathrm{SR}(t)}{\sqrt{u^{\mathrm{T}}(t)C(t)u(t)}}, \tag{4.47}$$

where $\mathrm{SR}(t)$ signifies the Sharpe Ratio at time t. The $\mathrm{SR}(t)$ expresses how much additional return someone obtains for enduring more volatility in order to retain a riskier asset, and can be expressed as below:

$$\mathrm{SR}(t) = \frac{E\big(p(t) - p_f(t)\big)}{\sigma(t)}, \tag{4.48}$$

where $\sigma(t)$ and $E\big(p(t) - p_f(t)\big)$ are the standard deviation and the mean of $\big(p(t) - p_f(t)\big)$, respectively. In addition, $p_f(t)$ signifies the theoretical rate of return on a zero-risk investment. In actuality, there is no such thing as a risk-free rate of return since every investment entails some level of risk. As a result, we set $p_f(t) = 0$ in our situation.

For calculating the investor's covariance matrix $\Upsilon(t)$, there are numerous methods [184–186]. Nevertheless, the market covariance $C(t)$ is frequently included in $\Upsilon(t)$ in reality. Thus, we consider

$$\Upsilon(t) = \mathrm{diag}\big((1-\tau)\mathrm{P}(t)C(t)\mathrm{P}^{\mathrm{T}}(t)\big), \tag{4.49}$$

where the matrix $\mathrm{P}(t)$ specifies the view-related assets at time t, and $\tau \in (0,1) \subset \mathbb{R}$ is the parameter that specifies the overall weighting of passive over active investment views. To calculate the new collection of equilibrium returns $p_{bl}(t)$, the formula below is applied [187, 188]:

$$p_{bl}(t) = \big[Q^{-1}(t) + \mathrm{P}^{\mathrm{T}}(t)\Upsilon^{-1}(t)\mathrm{P}(t)\big]^{-1}\big[(Q^{-1}(t)r(t) + \mathrm{P}^{\mathrm{T}}(t)\Upsilon^{-1}(t)Z^{\mathrm{T}}(t)\big], \tag{4.50}$$

where $Q(t) = \tau C(t)$ and $Z(t)$ signifies the vector containing the investor's views. In addition, the new collection of equilibrium returns' covariance $C_{bl}(t)$ is calculated as bellow [187, 188]:

$$C_{bl}(t) = \big[Q^{-1}(t) + \mathrm{P}^{\mathrm{T}}(t)\Upsilon^{-1}(t)\mathrm{P}(t)\big]^{-1}. \tag{4.51}$$

Substituting the $C(t)$, $p(t)$ and $p_g(t)$ of DTMPS problem with $C_{bl}(t)$ and $p_{bl}(t)$ the discrete-time BL portfolio optimization (DTBLPO) problem may be expressed in the next TVQP problem, for all feasible targets $p_g \in [\min(p_{bl}(t)), \max(p_{bl}(t))] \subseteq \mathbb{R}$:

$$\min_{\eta(t)} \quad \eta^{\mathrm{T}}(t) \cdot C_{bl}(t) \cdot \eta(t) \tag{4.52}$$

$$\text{s.t.} \quad \mathbf{1}_n^{\mathrm{T}} \cdot \eta(t) = 1 \tag{4.53}$$

$$-p_{bl}^{\mathrm{T}}(t) \cdot \eta(t) \leq -p_g(t) \tag{4.54}$$

$$\mathbf{0}_n \leq \eta(t) \leq \mathbf{1}_n. \tag{4.55}$$

The optimal BL portfolio is $\eta(t) = [\eta_1(t), \eta_2(t), \ldots, \eta_n(t)]$, where $\eta_i(t)$ is the DTBLPO problem's solution of the

asset i, $i = 1, 2, \ldots, n$, at time t. It is worth mentioning that (4.50) can be reformulated as below:

$$(1-\tau)\Big[C^{-1}(t)\big(p_{bl}(t) - r(t)\big)\Big] + \tau\Big[\mathrm{P}^{\mathrm{T}}(t)\Big(\mathrm{diag}\big(\mathrm{P}(t)C(t)\mathrm{P}^{\mathrm{T}}(t)\big)\Big)^{-1}\big(\mathrm{P}(t)p_{bl}(t) - Z^{\mathrm{T}}(t)\big)\Big] = 0. \tag{4.56}$$

If the parameter $\tau \to 0$, then $p_{bl}(t) \to p(t)$ (i.e., passive investment views are given the most weight), and if $\tau \to 1$, then $\mathrm{P}(t)p_{bl}(t) \to Z^{\mathrm{T}}(t)$ (i.e., active investment views are given the most weight). As a result, the parameter $\tau \in (0,1) \subset \mathbb{R}$ specifies the overall weighting of passive over active investment views.

Similar to [43, 152, 155, 156], interpolating the $p_{bl}(t)$ and the $C_{bl}(t)$ into continuous functions with any preferences' method, the DTBLPO problem is converted into the CTBLPO (or TV-BLPO) problem. As a result, $p_{bl}(t),\ C_{bl}(t) \in C[0, m-\theta-1]$, where $C[0, m-\theta-1]$ signifies the space of all real continuous functions on the interval $[0, m-\theta-1]$, and the TVQP problem of (4.52)-(4.55) becomes a continuous TVQP problem.

4.3.2 Continuous-Time BL Portfolio Optimization Problem via ZNN

To approach the TV-BLPO problem with all possible expected return targets into an ZNN, we need to include the equations (4.52)-(4.55) to the coefficients of the ZNN in subsection 3.6.1. Consequently, we set the coefficients in (3.52)-(3.54) as follows:

$$\begin{gathered} W(t) = C_{bl}(t), \quad q(t) = [\,], \quad K(t) = \mathbf{1}_n^{\mathrm{T}}, \quad \rho(t) = 1, \quad x(t) = \eta(t) \\ A(t) = \begin{bmatrix} -p_{bl}^{\mathrm{T}}(t) \\ I_n \\ -I_n \end{bmatrix} \in \mathbb{R}^{(1+2n)\times n}, \quad b(t) = \begin{bmatrix} -p_g(t) \\ \mathbf{1}_n \\ \mathbf{0}_n \end{bmatrix} \in \mathbb{R}^{1+2n}. \end{gathered}$$

Following the implementation of stages 1 and 2 in subsection 3.6.1, stage 3 defines the following error matrix equation group:

$$\begin{cases} E_1(t) = W(t)x(t) + P_x(t) + K^{\mathrm{T}}(t)\lambda(t) \\ E_2(t) = K(t)x(t) - \rho(t) \end{cases} \tag{4.57}$$

and then substituting $E_i(t), i = 1, 2$, defined in (4.57) in place of $E(t)$ into (3.38), one obtains

$$\begin{cases} \dot{W}(t)x(t) + W(t)\dot{x}(t) + \dot{P}_x(t) + K^{\mathrm{T}}(t)\dot{\lambda}(t) + \dot{K}^{\mathrm{T}}(t)\lambda(t) = -\gamma E_1(t) \\ \dot{K}(t)x(t) + K(t)\dot{x}(t) - \dot{\rho}(t) = -\gamma E_2(t) \end{cases} \tag{4.58}$$

where $\dot{P}_x(t) = P_1(t)\dot{x}(t) + P_2(t)x(t) + P_3(t)$ with the penalty functions $P_i(t), i = 1, 2, 3$, as presented in (3.61) and $N_i(t) = b_i(t) - A_i(t)x(t)$, $i = 1, 2, \ldots, 2n+1$. It is worth mentioning that

$$\dot{A}(t) = \begin{bmatrix} -\dot{p}_{bl}^{\mathrm{T}}(t) \\ \mathbf{0}_n \\ \mathbf{0}_n \end{bmatrix}, \quad \dot{b}(t) = \begin{bmatrix} -\dot{p}_g(t) \\ \mathbf{0}_n \\ \mathbf{0}_n \end{bmatrix}, \quad \dot{K}(t) = \mathbf{0}_n^{\mathrm{T}}, \quad \dot{\rho}(t) = 0.$$

As a result, (4.58) can be reformulated as follows:

$$\begin{cases} (W(t)+P_1(t))\,\dot{x}(t)+K^{\mathrm{T}}(t)\dot{\lambda}(t)=-\gamma E_1(t)-\dot{W}(t)x(t)-P_2(t)x(t)-P_3(t) \\ K(t)\dot{x}(t)=-\gamma E_2(t) \end{cases} \tag{4.59}$$

Then setting

$$S(t)=\begin{bmatrix} W(t)+P_1(t) & K^{\mathrm{T}}(t) \\ K(t) & 0 \end{bmatrix}, \quad u(t)=\begin{bmatrix} -\gamma E_1(t)-\dot{W}(t)x(t)-P_2(t)x(t)-P_3(t) \\ -\gamma E_2(t) \end{bmatrix}, \tag{4.60}$$

where $S(t)\in\mathbb{R}^{(n+1)\times(n+1)}$ and $u(t)\in\mathbb{R}^{n+1}$, (4.59) can be reformulated as follows:

$$S(t)\dot{y}(t)=u(t), \tag{4.61}$$

where $\dot{y}(t)=\begin{bmatrix}\dot{x}(t)\\ \dot{\lambda}(t)\end{bmatrix}\in\mathbb{R}^{n+1}$ and $S(t)$ is a nonsingular mass matrix. Note that the model of (4.61) is sufficient for solving the TV QP problem of (4.39)-(4.42), but because the creation of $C(t)$ and $r(t)$ in our approach is reliant on function handle in terms of MATLAB code, as will be shown in chapter 5, the following model of (4.62) would be easier to manage under an `ode` MATLAB solver. As a result, the proposed ZNN model for solving the TV QP problem of (4.39)-(4.42) is the following:

$$\dot{y}(t)=S^{-1}(t)u(t), \tag{4.62}$$

which is equivalent of (4.61) except for costs of calculating the inverse of the matrix $S(t)$. It is worth mentioning that the solution $y(t)$ of (4.62) can be efficiently generated by employing an `ode` MATLAB solver. The convergence of the ZNN solver to the unique theoretical solution is proven in the following theorem 4.3.1.

Theorem 4.3.1 *Starting from any initial state $y(0)\in\mathbb{R}^{n+1}$, the state variable $y(t)$ of ZNN (4.31) globally converges to the unique theoretical solution $y^*(t)$ of TV QP problem of (4.39)-(4.42). That is, $\lim_{t\to\infty}(y(t)-y^*(t))=0$. Furthermore, the theoretical solution $x^*(t)$ of TV QP problem of (4.39)-(4.42) is the first n elements of $y^*(t)$.*

Proof. In order to obtain the solution $x(t)$ of TV QP problem of (4.39)-(4.42), the error matrix equation is defined as in (4.57), based on the ZNN design. After that, by adopting the linear design formula for zeroing (4.57), the model (4.58) is obtained. From [87, Theorem 1], each error matrix equation in the error matrix equation group (4.58) converges to the theoretical solution when $t\to\infty$. As a result, the solution of (4.58) converges to the theoretical solution of TV QP problem of (4.39)-(4.42) when $t\to\infty$. In addition, from the derivation process of (4.62), we know that it is actually another form of (4.58). The proof is thus completed. □

4.3.3 Continuous-Time BL Portfolio Optimization Problem via LVI-PDNN

To convert the TV-BLPO problem into an LVI-PDNN, we need to include the equations (4.39)-(4.42) to the coefficients of the LVI-PDNN from [149]. According to the TVQP problem of subsection 4.2.1, if we set

- $G(t) = 2C_{bl}(t)$
- $d(t) = 1$
- $g(t) = [\,]$
- $x(t) = \eta(t)$
- $B(t) = -p_{bl}^{\mathrm{T}}(t)$
- $\zeta^{-}(t) = \mathbf{0}_n$
- $D(t) = \mathbf{1}_n^{\mathrm{T}}$
- $b(t) = -p_g(t)$
- $\zeta^{+}(t) = \mathbf{1}_n$

then the coefficients of the LVI-PDNN can be written as follows:

$$H(t) = \begin{bmatrix} G(t) & -D^{\mathrm{T}}(t) & B^{\mathrm{T}}(t) \\ D(t) & \mathbf{0}_{n\times n} & \mathbf{0}_n \\ -B(t) & \mathbf{0}_n^{\mathrm{T}} & 0 \end{bmatrix}, \quad h(t) = \begin{bmatrix} g(t) \\ -d(t) \\ b(t) \end{bmatrix}.$$

Furthermore, the definition of the primal-dual decision vector $y(t)$ can be written as follows, along with the lower and upper boundaries to which it is subject:

$$y(t) = \begin{bmatrix} \eta(t) \\ \mu(t) \\ \varrho(t) \end{bmatrix}, \quad \varsigma^{-}(t) = \begin{bmatrix} \zeta^{-}(t) \\ -\varpi\,\mathbf{1}_l \\ \mathbf{0}_k \end{bmatrix}, \quad \varsigma^{+}(t) = \begin{bmatrix} \zeta^{+}(t) \\ +\varpi\,\mathbf{1}_l \\ +\varpi\,\mathbf{1}_k \end{bmatrix}, \text{ where}$$

- the constant $\varpi \gg 0$ is the numerical representation of $+\infty$, large enough for implementation purposes;
- $\zeta^{-}(t) \leq x(t) \leq \zeta^{+}(t)$ is the basic parameter decision vector of the primal TVQP (4.39)-(4.42);
- $\mu(t) \in \mathbb{R}^l$ is the dual decision variable vector of the equality constraint (4.41).
- $\varrho(t) \in \mathbb{R}^k$ is the dual decision variable vector of the inequality constraint (4.40).

The following dynamical system can be used to solve this TV QP problem:

$$\dot{y}(t) = \gamma(I + H^{\mathrm{T}}(t))(P_{\Omega}(y(t) - (H(t)y(t) + h(t))) - y(t)), \tag{4.63}$$

where $P_{\Omega}(\cdot)$ is the projection operator (see [149]) and $\gamma > 0$ is known as the design parameter. Note that the solution $y(t)$ of (4.63) can be efficiently generated by employing an `ode` MATLAB solver.

4.4 The Multi-Period Portfolio Insurance under Transaction Costs Problem

One way to decrease a portfolio's spending is to reduce its insurance costs (see [189–191]). For example, a novel strategy that generalizes the CPPI approach is presented in [190]. The main purpose of this strategy is to guarantee the investment target or floor while engaging in asset performance and, at the same time, minimizing the portfolio's downside risk. It is also shown that the strategy accounts for the investor's behavioral aspects, such as skewed likelihoods, risk-averse benefit behavior, and risk-seeking loss behavior. In [191], the authors analyze the efficiency of option-based and constant percentage PI strategies for a defined contribution fund targeting a minimum rate of retirement annuity income covered from inflation. They conclude that their option-based approach typically leads to higher cumulative retirement savings, whereas the constant ratio approach offers a more robust downside risk security to stock market jumps/volatilities.

In this section, we give a concise introduction to the MCPI problem and we define its multi-period version, named MPMCPI. Furthermore, we also define the MPMCPI under transaction costs (MPMCPITC) problem that

minimizes as well the transaction costs and, simultaneously, seeks to reach full payoff while keeping the payoff over a price floor with respect to time-period which they belong. Note that our approach to the PI problem is resembling the problem described and investigated in several publications, such as [3, 5, 155, 161–164].

The linear space $X = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^{m\times n}$, is called the space of marketed securities, where $x_i \in \mathbb{R}^m$ denotes i-security, $i = 1, 2, \ldots, n$, with information from m successive price records. A vector $\theta = (\theta_1, \theta_2, \ldots, \theta_n)$ of $\mathbb{R}^n$, is called a portfolio, if θ_i implies the i-th security's number of shares. For the classical two period model, by considering θ, which is not zero, as the initial portfolio at the 1-st period, its payoff at the 2-nd period is the outcome of the following formula:

$$G(\theta) = \sum_{i=1}^{n} \theta_i x_i, \tag{4.64}$$

where $x_i \in \mathbb{R}^m,\ i = 1, 2, \ldots, n$, and G is the payoff operator. Given a "floor" price $\phi \in \mathbb{R}$, the insured payoff on the portfolio θ in the insurance price p and at the "floor" ϕ is the supremum $G(\theta) \vee \phi \cdot \mathbf{1}_m$, where " $\vee$ " signifies the operator of supremum. Based on that, the minimum-cost insured portfolio η is the solution to the MCPI problem below:

$$\min_{\eta} \quad p^{\mathrm{T}} \cdot \eta \tag{4.65}$$

$$\text{s.t.} \quad X \cdot \eta \geq G(\theta) \vee \phi \cdot \mathbf{1}_m. \tag{4.66}$$

The problem of (4.65)-(4.66) can be formulated in a linear programming (LP) form as below:

$$\min_{\eta} \quad p^{\mathrm{T}} \cdot \eta \tag{4.67}$$

$$\text{s.t.} \quad -X \cdot \eta \leq \min\{-G(\theta), -\phi \cdot \mathbf{1}_m\} \tag{4.68}$$

$$\mathbf{0}_n \leq \eta \leq X(1,:) \cdot \theta \cdot \left[\frac{1}{x_1(1)}, \ldots, \frac{1}{x_n(1)}\right]^{\mathrm{T}}, \tag{4.69}$$

In addition, the right part of (4.69) is the greatest quantity of each stock that an investor is able to hold by putting all the payout of the portfolio into each stock. Therein, $x_i(1)$ implies the first element of the vector x_i, $i = 1, \ldots, n$.

4.4.1 MPMCPI under Transaction Costs

We consider the MPMCPI problem to be a multi-period version of the MCPI problem. Considering that m is the observations' number of one period then β denotes the observations' number of one sub-period. According to this, the following must hold, $m = \sum_{t=0}^{[m/\beta]} \beta t$, where t is the number of the sub-periods and $[x]$ denotes the integer part of x. Consequently, we transform the space $X \in \mathbb{R}^{m\times n}$ to $X(t) = [x_1(t), x_2(t), \ldots, x_n(t)] \in \mathbb{R}^{\beta\times n}$, where $x_i(t) \in \mathbb{R}^{\beta}$, comprises of the prices of the i-security, over time t.

Our model consists of $[m/\beta] + 1$ time-periods, and we consider $\eta(0) = \theta$, and $\eta(t),\ t = 1, 2, \ldots, [m/\beta]$, as the requested ones, we have that $\eta(t) \in \mathbb{R}^n$ for $t = 0, 1, \ldots, [m/\beta]$ time-periods. Moreover, there exists two variants of operator G, the G_1, that works as the G operator, and the G_2, that excludes the previous period's insurance costs. Consequently, we have

$$G_1(\eta(t-1)) = \sum_{i=1}^{n} \eta_i(t-1) x_i(t)$$

and

$$G_2(\eta(t-1)) = \sum_{i=1}^{n} \eta_i(t-1)x_i(t) - \sum_{i=1}^{n} \eta_i(t-1)p_i(t-1),$$

where $p(t) = [p_1(t), p_2(t), \ldots, p_n(t)] \in \mathbb{R}^n$ is a vector of multi-period insurance prices and $\sum_{i=1}^{n} \eta_i(t-1)p_i(t-1)$ is the insurance cost for the preceding period. Notice that, for consistency reasons with equation (4.64), it must hold $\sum_{i=1}^{n} \eta_i(0)p_i(0) = 0$, hence, for $t = 1$ we have $G_1 = G_2 = G$.

Given a "floor" price $\phi(t) \in \mathbb{R}$, the insured payoff on $\eta(t-1)$ in the insurance price $p(t)$ and at the floor $\phi(t)$ is the supremum $G_2(q(t-1)) \vee \phi(t)$. According to this, the multi-period minimum-cost insured portfolio $\eta(t)$ solves the following MPMCPI problem:

$$\min_{\eta(t)} \quad p^{\mathrm{T}}(t) \cdot \eta(t) \tag{4.70}$$

$$\text{s.t.} \quad G_1(\eta(t)) \geq G_2(\eta(t-1)) \vee \phi(t) \cdot \mathbf{1}_m. \tag{4.71}$$

The problem of (4.70)-(4.71) can be formulated in a LP form as below:

$$\min_{\eta(t)} \quad p^{\mathrm{T}}(t) \cdot \eta(t) \tag{4.72}$$

$$\text{s.t.} \quad -X(t) \cdot \eta(t) \leq \min\{-\text{payoff}, -\phi(t) \cdot \mathbf{1}_m\} \tag{4.73}$$

$$\mathbf{0}_n \leq \eta(t) \leq \hat{X}(t) \cdot \eta(t-1) \cdot \left[\frac{1}{\hat{x}_1(t)}, \ldots, \frac{1}{\hat{x}_n(t)}\right], \tag{4.74}$$

where $\text{payoff} = (X(t) - p^{\mathrm{T}}(t-1)) \cdot \eta(t-1)$ and $\hat{X}(t) = X((t-1)\beta + 1, :)$. Note that $\hat{X}(t) = [\hat{x}_1(t), \ldots, \hat{x}_n(t)] \in \mathbb{R}^n$.

Furthermore, we assume that the portfolio's costs are separable as proposed in [192] and we use their multi-period version as introduced in [155]. Thus, we define $\kappa(t)$ to be the sum of the transaction costs related to the atomic exchanges, that is,

$$\kappa(t) = \sum_{i=1}^{n} \kappa_i(t), \tag{4.75}$$

where $\kappa_i(t)$ denotes the i transaction cost in time period t. Our strategy is enhanced to manage composite transaction costs. Considering that ζ^-,ζ^+ are the cost prices associated with the sale and buy of assets i, and δ^-, δ^+, are the fixed costs associated with the sale and buy of assets i, the fixed along with the linear multi-period transaction cost is the following:

$$\kappa_i(t) = \begin{cases} 0, & \eta_i(t) = \eta_i(t-1) \\ \delta_i^+ + \zeta_i^+(\eta_i(t) - \eta_i(t-1))x_i(t), & \eta_i(t-1) < \eta_i(t) \\ \delta_i^- + \zeta_i^-(\eta_i(t-1) - \eta_i(t))x_i(t), & \eta_i(t) < \eta_i(t-1) \end{cases} \tag{4.76}$$

Notice that this function is explicitly non-convex, with the exception of zero fixed cost condition.

By adding the transaction cost function (4.76) into the problem of (4.72)-(4.74), the MPMCPITC problem can be formulated in a NLP form as follows:

$$\min_{\eta(t)} \quad p^{\mathrm{T}}(t)\cdot\eta(t)+\kappa(t) \tag{4.77}$$

$$\text{s.t.} \quad -X^{\mathrm{T}}(t)\cdot\eta(t)\le\min\{\kappa(t)-\text{payoff},-\phi(t)\cdot\mathbf{1}_m\} \tag{4.78}$$

$$\mathbf{0}_n\le\eta(t)\le\hat{X}(t)\cdot\eta(t-1)\cdot\left[\frac{1}{\hat{x}_1(t)},\ldots,\frac{1}{\hat{x}_n(t)}\right]. \tag{4.79}$$

Notice that we exclude from the portfolio's payoff the prior insurance and transaction costs. In this way, the MPMCPITC problem becomes more genuine. Also, note that (4.77) is the objective function of the MPMCPITC problem.

4.4.2 Intelligent Algorithms

Intelligent algorithms are practical alternative techniques for tackling and solving a variety of challenging problems. Nature has been an inspiration for several meta-heuristic algorithms being introduced. Without being told, but by experience, it has managed to solve problems. The principal inspiration behind the early meta-heuristic algorithms was natural selection and survival of the fittest. Via various communication styles, different species interact with each other. In this section, we present the BAS algorithm in detail and we give some brief introduction to the firefly and bat algorithms. Furthermore, our approach on the MPMCPITC problem is presented.

Beetle Antennae Search Algorithm

The searching behavior of a beetle is described by a metaheuristic algorithm, namely BAS, see [39]. Such a technique permits novel optimisation algorithms to be developed (see [40–44]).

At i-th time, consider the location of beetle as a vector, x_i, $i=1,2,\ldots$, and signify the odour concentration at position x to be $f(x)$ defined as a fitness function. The maximum of $f(x)$ refers to the source odour point. The searching behavior model is described by the random path of the beetle searching as described in the following:

$$b=\frac{\mathrm{rnd}(g,1)}{\|\mathrm{rnd}(g,1)\|},$$

$$B=\frac{b}{2^{-52}+\|b\|}, \tag{4.80}$$

where $\mathrm{rnd}(\cdot)$ implies a random function, and g denotes the dimensions of position. For imitating the movements of the beetle's antennae, the searching behaviors of right-hand (x_R) and left-hand (x_L) side are formulated as follows:

$$x_R=x_i+Bd_i, \tag{4.81}$$

$$x_L=x_i-Bd_i, \tag{4.82}$$

where d is the sensing diameter of antennae equivalent to the exploit ability. Furthermore, the behavior of detecting can be formulated as follows:

$$x_i=x_{i-1}+B\delta_i\mathrm{sign}(f(x_R)-f(x_L)), \tag{4.83}$$

where δ is the search step size that accounts for convergence speed after a decreasing i function, and sign$(\cdot)$ represents a sign function. Lastly, the update rules of d and δ are the followings:

$$d_i = 0.988 d_{i-1} + 0.001, \tag{4.84}$$

$$\delta_i = 0.988 \delta_{i-1}. \tag{4.85}$$

Algorithm 2 Algorithm of beetle antennae search (BAS).

Input: Write an objective function $f(x_i)$, where $x_i = [x_1, x_2, \ldots, x_n]$, and initialize the parameters x_0, d_0, δ_0, i.

while $i < K_{max}$ **OR** (stop criterion) **do**
    Set the vector unit B in line with (4.80)
    Investigate in variable space with two antennae in line with (4.81) and (4.82)
    Update the state variable x_i in line with (4.83)
    if $f(x_i) < f_{best}$ **then**
        Set $f_{best} = f(x_i)$ and $x_{best} = x_i$
    end if
    Update d and δ in line with (4.84) and (4.85), respectively
    Set $i = i + 1$
end while

Output: x_{best}, f_{best}.

Popular Meta-heuristic Algorithms

In MPMCPITC problem, the effectiveness of BAS is compared with two well known meta-heuristic algorithms as well as the GA MATLAB function. These algorithms are the firefly algorithm (FA) [28] and the bat algorithm (BA) [193] (see chapter 2). Of course there are several other algorithm options in the literature we might have used but in this problem we put the BA and FA against BAS because of their efficiency and originality.

Multi-Objective Optimization

The MPMCPITC is an optimization problem which not has only a single objective, since we want to minimize a portfolio's insurance and transaction costs when seeking to reach the highest payoff while at the same time maintaining the payoff above a floor price. In this case, we are dealing with a multi-objective optimization problem. Note that all of the meta-heuristic algorithms discussed in this section are specifically applicable to unconstrained optimization as we use their regular form. For this purpose, it is important to use some external approaches that can maintain alternatives in a feasible area. It is worth mentioning that there are many methods to achieve that but, in the MPMCPITC problem, we use the penalty function method presented in [9] (see Section 2.1.2).

The penalty function method alter the objective function of the problem by adding a penalty function s. Considering that $f(x) = p^{\mathrm{T}}(t) \cdot \eta(t) + \kappa(t)$, as in (4.77), we optimize $Z(x,R)$ instead of $f(x)$, where

$$Z(x,R) = f(x) + s(R, r(x)). \tag{4.86}$$

In (4.86), R denotes a set of penalties and $r(x)$ represents the inequality constraint functions, (4.78) and (4.79). Since there is more than one penalty function presented in [9], we choose to use

$$s = R\langle r_j(x)\rangle^2, \tag{4.87}$$

where $\langle\cdot\rangle$ implies the bracket-operator with $\langle z\rangle = z$, if z is negative, else $\langle z\rangle = 0$. Furthermore, $r_j(x)$ is a penalty condition handling the j-th inequality constraint. As the infeasible signs are imported with a negative price, the bracket operator is an external penalty condition. This operator is specifically employed to resolve inequality constraints.

In Algorithm 3, the algorithmic process that manages MPMCPITC's equality/inequality constraint is shown. Note that this algorithm include MATLAB code.

Algorithm 3 Penalty function approach on the MPMCPITC problem.

Input: The penalty parameter R; f the objective function (4.77); A and b the left and right part of (4.78), respectively; the lower limit η^- and upper limit η^+ of (4.79).

1: Set $s = R(\text{sum}((A > b).(b - A)^2 + (\eta^- > \eta).(\eta^- - \eta)^2 + (\eta > \eta^+).(\eta - \eta^+)^2))$

2: Set $Z = f + s$

Output: The outcome of Z.

The Main Algorithm for the MPMCPITC Problem

First, we introduce the supplementary Algorithm 4 that creates the variables of the MPMCPITC problem of (4.77)-(4.79). Note that all the algorithms in this section include MATLAB code.

Algorithm 4 Algorithm for the construction of (4.77)-(4.79) problem's variables.

Input: The marketed space X; the insurance prices p; the time t; the delay parameter β; the floor ϕ; the previous portfolio $\eta_{-1} = \eta(t-1)$; and the previous insurance and transaction costs $y_{-1} = y(t-1)$.

1: **function** $[\eta^-,\eta^+,A,b,P]$ = problem$(X,p,\phi,\eta_{-1},y_{-1},t,\beta)$

2: Set $n = \text{size}(X,2)$ and $A = X((t-1)\beta + 1 : t\beta,:)$

3: Set payoff $= -A\eta_{-1} + y_{-1}$

4: Set $b = \min(\text{payoff}, -\phi(t)\text{ones}(\beta,1))$

5: Set $P = p(t)$ and $\eta^- = \text{zeros}(n,1)$

6: Set $\eta^+ = A(1,:)\eta_{-1}./A(1,:)'$

7: **end function**

Output: The η^-,η^+,A,b,P at time t.

In the MPMCPITC, we presume that the costs of insurance of the portfolio include a standard price plus a risk rate of the assets. If the price rates related to the asset risk is denoted by ξ and the fixed price by λ, then the function of the fixed and linear insurance prices is composed as below:

$$p_i = \lambda + \xi \cdot \mathrm{Var}\Big[\frac{x_i}{\max(x_i)}\Big],\ i = 1,2,\ldots,n, \tag{4.88}$$

where $\mathrm{Var}[Y]$ implies the variance of Y. Since $x_i \in \mathbb{R}^\beta$, it is the variance of its β normalized prices that measures the risk of the i asset.

The insurance price of any asset relies on the level of risk it bears, where there is a possibility that the expected return will be different from the real return. Hence, the prices of insurance are becoming more realistic.

The following Algorithm 5 is the main algorithm which includes the complete procedure for solving the MPMCPITC problem of (4.77)-(4.79) along with the construction of the insurance prices vector.

Algorithm 5 Main algorithm for the solution of MPMCPITC problem.

Input: The marketed space $X = [x_1, x_2, \ldots, x_n]$ which is a matrix of n time series as column vectors of m prices; the delay parameter $\beta \le m$, $\beta \in \mathbb{N}$; the price rates ξ and the fixed price λ of (4.88); the portfolio $\theta \in \mathbb{R}^n$ and the floor vector $\phi \in \mathbb{R}^\beta$.

1: Set $[m, n] = \mathrm{size}(X)$, $s = [m/\beta]$ and $p = \mathrm{zeros}(s, n)$
2: **for** $i = 1 : \beta : m$ **do**
3: Set $Y = X(i : i + \beta - 1, :)$
4: Set $p((i-1)/\beta + 1, :) = \lambda + \xi \mathrm{var}(Y./\max(Y))$
5: **end for**
6: Set $\eta = \mathrm{zeros}(n, s)$ and $f_\eta = \mathrm{zeros}(1, s)$
7: $[\eta^-, \eta^+, A, b, P]$ = problem $(X, p, \phi, \theta, 0, 1, \beta)$
8: Set $\eta(:, 1)$ the optimal solution of the penalty function of Algorithm 3 by using the variables η^-, η^+, A, b, P of the previous step via any intelligent algorithm
9: Set $f_\eta(1)$ the outcome of (4.77) for $\eta(:, 1)$
10: **for** $t = 2 : s$ **do**
11: $[\eta^-, \eta^+, A, b, P]$ = problem $(X, p, \phi, x(:, t-1), y(:, t-1), t, \beta)$
12: Set $\eta(:, t)$ the optimal solution of the penalty function of Algorithm 3 by using the variables η^-, η^+, A, b, P of the previous step via any intelligent algorithm
13: Set $f_\eta(t)$ the outcome of (4.77) for $\eta(:, t)$
14: **end for**

Output: The optimal solution of the MPMCPITC problem, $\eta(t)$ and $f_\eta(t)$.

4.5 The TV Minimum-Cost Portfolio Insurance under Transaction Costs Problem

In this section, we define the TV-MCPI problem as well as the TV portfolio selection under transaction costs (TV-PSTC) problem. The TV-MCPI problem minimizes the insurance costs of a portfolio while keeping its payoff

above a floor price. The TV-PSTC problem minimizes the transaction costs of a portfolio while trying to achieve maximum payoff. By combining the TV-MCPI and TV-PSTC, we also define the TV-MCPITC problem. The TV-MCPITC problem minimizes the insurance costs and the transaction costs of a portfolio while trying to achieve maximum payoff and keeping that payoff above a floor price at the same time. Then, a modified beetle antennae algorithm (BAS) algorithm for solving the TV-MCPITC problem is presented.

4.5.1 Definition of the TV-MCPI Financial Problem

Our approach to the portfolio insurance problem is a TV analog of the corresponding static problem defined and studied in a number of papers, such as [3, 5, 161, 163, 164]. As far as we are aware of, our TV version of the portfolio insurance problem is a novel approach that comprises modern meta-heuristic optimization techniques to provide an online, thus more realistic, solution to the TV-MCPITC problem.

The space of marketed securities is $X = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^{m\times n}$ where $x_i \in \mathbb{R}^m$ is the security i, where $i = 1, 2, \ldots, n$, and comprises data from the last m observations of its price. In the discrete TV-MCPI problem, we convert the space of marketed securities to a TV vector $X(t) = [x_1(t), x_2(t), \ldots, x_n(t)] \in \mathbb{R}^n$ where $x_i(t) \in \mathbb{R}$ is the security i, $i = 1, 2, \ldots, n$, and the time $t \in [1, m]$ denotes the new value that it comes during the calculation of the solution to the TV-MCPI problem.

A portfolio is a vector $\eta_p = (\eta_{p_1}, \eta_{p_2}, \ldots, \eta_{p_n})$ of $\mathbb{R}^n$ where η_{pi} is the number of shares of the ith security. In a two period model, if $\eta_p = (\eta_{p_1}, \eta_{p_2}, \ldots, \eta_{p_n})$ is a portfolio that is not zero at the time $t = 0$, then its payoff at the time $t = 1$ is given by the formula

$$G(\eta_p) = \sum_{i=1}^{n} \eta_{p_i} x_i(1), \tag{4.89}$$

where $x_i(1) \in \mathbb{R},\ i = 1, 2, \ldots, n$. The G operator is one-to-one and is called the payoff operator. In our m period model, we consider the portfolio $\eta_p = \eta(0)$ as the initial portfolio and $\eta(t) \in \mathbb{R}^n,\ t \in [1, m]$, as the requested one. Here we have two versions of the G operator. One version is the operator G_1 which is identical to the operator G and the other version is G_2 which removes the insurance costs of the previous period. Hence, we have

$$G_1(\eta(t-1)) = \sum_{i=1}^{n} \eta(t-1) x_i(t)$$

and

$$G_2(\eta(t-1)) = \sum_{i=1}^{n} \eta(t-1) x_i(t) - \sum_{i=1}^{n} \eta(t-1) p(t-1),$$

where $p(t) = (p_1(t), p_2(t), \ldots, p_n(t)) \in \mathbb{R}^n$ is a vector of TV security prices and $\sum_{i=1}^{n} \eta(t-1) p(t-1)$ is the insurance cost of the previous period. Note that it must hold $\sum_{i=1}^{n} \eta(0) p(0) = 0$ in order to be true the Equation (4.89). Consequently, for $t = 1$ we have that $G_1 = G_2 = G$.

If we also have a "floor" price $\phi(t) \in \mathbb{R}$ then the insured payoff on the $\eta(t-1)$ portfolio at the $\phi(t)$ "floor" and in the $p(t)$ price is the supremum $G_2(\eta(t-1)) \vee \phi(t)$. The TV minimum-cost insured portfolio $\eta(t)$ at the floor $\phi(t)$ and in the price $p(t)$ is the solution to the following cost minimization constraint problem:

$$\begin{aligned} \min_{\eta(t)} \quad & p^{\mathrm{T}}(t)\cdot\eta(t) \\ \text{s.t.} \quad & G_1(\eta(t)) \geq G_2(\eta(t-1)) \vee \phi(t). \end{aligned}$$

This problem is the TV-MCPI and it can also be written in the following TV LP (TVLP) form:

$$\min_{\eta(t)} \quad p^{\mathrm{T}}(t)\cdot\eta(t) \tag{4.90}$$

$$\text{s.t.} \quad -X^{\mathrm{T}}(t)\cdot\eta(t) \leq \min\{-\text{payoff}, -\phi(t)\} \tag{4.91}$$

$$0 \leq \eta(t) \leq X^{\mathrm{T}}(t)\cdot\eta(t-1)\cdot\left[\frac{1}{x_1(t)},\ldots,\frac{1}{x_n(t)}\right], \tag{4.92}$$

where payoff $= (X^T(t) - p^{\mathrm{T}}(t-1))\cdot\eta(t-1)$.

Similar to [43, 152, 155, 156], we convert the discrete TV-MCPI problem to CT form by interpolating the $X(t)$, $p(t)$ and the $\phi(t)$ into continuous functions with any method of preferences. Consequently, $X(t),\ p(t),\ \phi(t) \in C[0, m-1]$, where $C[0, m-1]$ is the space of all continuous real functions on the interval $[0, m-1]$. The optimal minimum-cost insured portfolio is equal to $\eta(t) = [\eta_1(t),\ldots,\eta_n(t)]$, where $\eta(t)$ is the online solution.

Insurance Pricing

The amount of money an individual or business must pay for an insurance policy is termed as insurance premium. The insurance premium for any asset also depends on the degree of risk that it carries. Risk represents the probability that the actual return may differ from the expected return. In order to be more realistic, we assume that the insurance costs of our portfolio comprise of a fixed charge plus a rate of the variance (risk) of the assets. Let θ be the price rates associated with the risk of assets and ζ be the fixed price. The fixed-plus-linear TV insurance prices function is given by

$$p(t) = \zeta + \theta\mathrm{Var}\left[\frac{Y}{\max(Y)}\right], \tag{4.93}$$

where $Y = X(t-\tau : t, :)$. The number $\tau \leq m-1$, $\tau \in \mathbb{N}$, is a constant number and it denotes the 'number of time periods'. The risk of the asset is calculated by the variance of its last τ normalized prices.

4.5.2 Definition of the TV-PSTC Financial Problem

Transaction costs, such as brokerage commissions, bid-ask spreads, taxes or even fund loads, can be used to model a variety of costs. In this section, we consider the transaction costs to be separable as presented in [192]. That is, the sum of the transaction costs associated with individual trades is equal to

$$\kappa(t) = \sum_{i=1}^{n} \kappa_i(t), \tag{4.94}$$

where κ_i is the transaction cost function for asset i. In reality, transaction costs are nonconvex functions of the traded amount. Actually, the costs of either buying or selling would probably be concave. For example, a

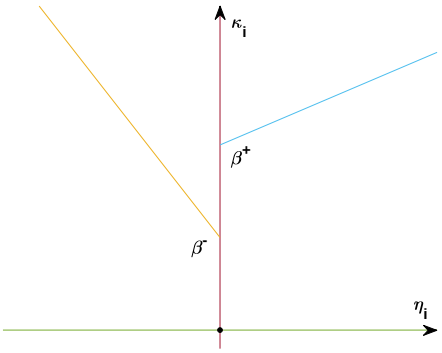


Figure 4.2: Fixed plus linear transaction costs $\kappa_i(t)$ as a function of transaction amount $(\eta_i(t)-\eta_i(t-1))x_i(t)$. There is no cost if there is not a transaction, i.e., $\kappa_i(t)=0$.

fixed price for any non-zero exchange is ordinary, and there could exist one or more breakpoints at which the transaction cost per share may depreciate. We assume a simplistic model that involves fixed plus linear costs. But our approach is expanded to handle more complex transaction cost functions. Let α^+, α^- be the cost rates related to buying and selling asset i and β^+, β^- the fixed costs associated with buying and selling asset i. The fixed-plus-linear TV transaction cost function is given by

$$\kappa_i(t)=\begin{cases} 0, & \eta_i(t)=\eta_i(t-1) \\ \beta_i^+ + \alpha_i^+(\eta_i(t)-\eta_i(t-1))x_i(t), & \eta_i(t)>\eta_i(t-1) \\ \beta_i^- + \alpha_i^-(\eta_i(t-1)-\eta_i(t))x_i(t), & \eta_i(t)<\eta_i(t-1) \end{cases} \tag{4.95}$$

where $x_i(t)$ is the price of the i stock at time t. Fig. 4.2 shows the transaction cost function. This function is clearly nonconvex, except in the case of zero fixed costs.

The TV-PSTC problem

A related problem of Subsection 4.5.1 is the minimization of the total transaction costs s.t. portfolio constraints. Among all possible transactions that produce portfolios achieving a given expected return and satisfying portfolio constraints, we would like to select those which generate the smallest total cost. Hence, the TV-PSTC problem is written as follows:

$$\min_{\eta(t)} \quad \kappa(t) \tag{4.96}$$

$$\text{s.t.} \quad \sum_i \eta_i(t)\cdot r_i(t) \geq r_p(t) \tag{4.97}$$

$$\eta_i(t) \in \mathbb{R}_0^+, \ \forall i, \tag{4.98}$$

where $\kappa(t)$ is defined in (4.94) and $\mathbb{R}_0^+$ denotes non-negative real numbers.

4.5.3 The TV-MCPITC Financial Problem

Combining the TV-PSTC problem and insurance pricing, the TV-MCPITC problem is formulated as follows:

$$\min_{\eta(t)} \quad p^{\mathrm{T}}(t)\cdot\eta(t)+\kappa(t) \tag{4.99}$$

$$\text{s.t.} \quad G_1(\eta(t))\geq(G_2(\eta(t-1))-\kappa(t-1))\vee\phi(t) \tag{4.100}$$

$$\eta_i(t)\in\mathbb{R}_0^+,\ \forall i. \tag{4.101}$$

This problem can also be written in the TV-NLP form as follows:

$$\min_{\eta(t)} \quad p^{\mathrm{T}}(t)\cdot\eta(t)+\kappa(t) \tag{4.102}$$

$$\text{s.t.} \quad -X^{\mathrm{T}}(t)\cdot\eta(t)\leq\min\{\kappa(t)-\text{payoff},-\phi(t)\} \tag{4.103}$$

$$0\leq\eta(t)\leq X^{\mathrm{T}}(t)\cdot\eta(t-1)\cdot\left[\frac{1}{x_1(t)},\ldots,\frac{1}{x_n(t)}\right], \tag{4.104}$$

where

$$\text{payoff}=(X^{T}(t)-p^{\mathrm{T}}(t-1))\cdot\eta(t-1).$$

Note that we remove the previous insurance costs from the portfolio's payoff and also remove the previous transaction cost. Hence, the TV-MCPITC problem becomes more realistic. The following algorithmic procedure defines the MATLAB function for generating the right hand side in (4.102).

Algorithm 6 Algorithmic procedure for Eq. (4.102).

Input: The insurance prices $p=p(t)$, the stock prices $A=A(t)$, the portfolio $\eta=\eta(t)$ and the previous $\eta_{-1}=\eta(t-1)$, the fixed costs β^-, β^+ and the costs rates α^-, α^+.

1: **function** f = minfunc(η, η_{-1}, p, A, β^-, β^+, α^-, α^+)

2: Set $f=p'\eta+sum((\eta>\eta_{-1}).(\beta^++\alpha^+(\eta-\eta_{-1}).A')+(\eta<\eta_{-1}).(\beta^-+\alpha^-(\eta_{-1}-\eta).A'))$

3: **return** f

4: **end function**

Output: The function in (4.102).

4.5.4 Penalty Function

The following penalty function is used (see Sections 2.1.2 and 4.4.2):

$$P(x,R)=f(x)+\Omega(R,g(x),h(x)), \tag{4.105}$$

where R is a set of penalty parameters, Ω is the penalty term chosen to favor the selection of feasible points over infeasible points, $f(x)$ is the function that we want to minimize, $g(x)$ and $h(x)$ are the inequality and equality constraint functions, respectively.

In this section, we approach the TV-MCPITC problem via a modified version of a BAS algorithm (see chapter 2). In our modified version of BAS, we make use of the aforementioned penalty functions. So, in our case, we only use the penalty function (4.105) as it is and the user sets the initial value of the penalty parameter R, which

stays constant through all the sequences that BAS generates. By adding the penalty function inside the BAS algorithm as presented in subsection 4.5.5, we are able to make a modified BAS algorithm even more efficient than the original. The reason is that the penalty function allows the BAS method to handle more efficiently any constraints (convex or nonconvex). Apart from that, the philosophy behind the BAS algorithm stays the same as presented in [39].

We combine the penalty function with the Bracket operator penalty,

$$\Omega = R\langle g_j(x)\rangle^2, \tag{4.106}$$

where $\langle k\rangle = k$, if k is negative, otherwise $\langle k\rangle = 0$. This term handles inequality constraints. The bracket operator is an exterior penalty term because it applies a positive value to the infeasible points. This operator is primarily used to handle the inequality constraints. Corresponding algorithmic procedure is presented as a MATLAB function defined in Algorithm 7.

Algorithm 7 Penalty function algorithm for TV-MCPITC.

Input: The requirements of Algorithm 6 plus the penalty parameter R, b the right part of Eq. (4.103) and the lower limit η^- and upper limit η^+ of Eq. (4.104), respectively.

1: **function** P = penfunc(η, η_{-1}, p, A, b, R, η^-, η^+, β^-, β^+, α^-, α^+)
2: Set $\Omega = R(\text{sum}((A\eta > b).(-A\eta + b)^2 + (\eta^- > \eta).(\eta^- - \eta)^2 + (\eta > \eta^+).(\eta - \eta^+)^2))$
3: Set P =minfunc$(\eta, \eta_{-1}, p, -A, \beta^-, \beta^+, \alpha^-, \alpha^+) + \Omega$
4: **return** P
5: **end function**

Output: The outcome of Penalty Function.

4.5.5 Modified BAS Algorithm for solving the TV-MCPITC Problem

In the following, we modified the BAS algorithm by using the two MATLAB functions presented in Algorithms 6 and 7. Namely, function `minfunc` implements Algorithm 6 and function `penfunc` implements Algorithm 7. Note that the dots in the input arguments of functions `minfunc` and `penfunc` in Algorithm 8 mean that the rest of the input arguments stay the same as they are declared in algorithms 6 and 7.

Algorithm 8 BAS algorithm for the TV-MCPITC problem.

Input: The requirements of Algorithms 6 and 7 plus the parameters d, δ, tol and $kmax$.

1: **function** [η,f_η] = basfunc(η_{-1}, p, A, b, R, η^-, η^+, β^-, β^+, α^-, α^+, d, δ, tol, $kmax$)
2: Set $y_1 = \eta_{-1}$ and y_2 = (nan)ones(size(η_{-1}))
3: Set $k = 0$ and len =length(η_{-1})
4: **while** $k < kmax$ **OR** $\left\|\text{penfunc}(y_2,\ldots) - \text{penfunc}(y_1,\ldots)\right\|_2 > tol$ **do**
5: Set b =rands(len, 1) and $b = \frac{b}{2^{-52}+\|b\|}$
6: Set $y_r = y_1 - db$ and $y_l = y_1 + db$

Set $y = |y_1 + \delta b(\text{sign}(\text{penfunc}(y_r,\ldots) - \text{penfunc}(y_l,\ldots)))|$
if penfunc$(y,\ldots)$ <penfunc$(y_1,\ldots)$ **then**
Set $y_2 = y_1$ and $y_1 = y$
end if
Set $d = 0.991d + 0.001$, $\delta = 0.991\delta$ and $k = k + 1$
end while
return $\eta = y_1$, f_η =minfunc$(y_1, \eta_{-1}, p, -A, \ldots)$
end function
Output: $\eta = \eta(t)$, f_η.

4.6 The TV Mean-Variance Portfolio Selection under Transaction Costs and Cardinality Constraint Problem

In this section, we define the TV-MVPS problem and describe the TV-PSTC problem. The TV-MVPS problem reduces the variance of a portfolio and, at the same time, it keeps the expected return over a target value. The TV-PSTC problem reduces the cost of transactions of a portfolio and, at the same time, it tries to achieve a maximum expected return over a target value. Furthermore, by merging the TV-MVPS and TV-PSTC problems and adding a cardinality constraint, we propose the TV-MVPSTC-CC problem too. The TV-MVPSTC-CC problem reduces the variance and the cost of transactions of a portfolio and, at the same time, it tries to achieve a maximum expected return and keeps the expected return over a target value in a specific number of the portfolio's securities. Then, a modified BAS algorithm for solving the TV-MVPSTC-CC problem is presented.

4.6.1 The TV-MVPS Problem Definition

We are dealing with the MVPS problem as a TV analog of the static case, which is studied in many articles (see [2, 169–172]). The MVPS is a financial optimization problem for assembling a portfolio of assets such that its risk is minimized under a target expected return. To the best of our knowledge, the TV version of the MVPS (TV-MVPS) problem presented inhere is a fresh approach that consists of robust techniques from neural networks to provide an online, thus more realistic, solution.

The marketed securities space is $X = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^{m\times n}$ where $x_i \in \mathbb{R}^m$ is the security i, $i = 1, 2, \ldots, n$, and consists of the last m observations of its price. In the static MVPS problem the expected return of the marketed space is $r = [r_1, r_2, \ldots, r_n] \in \mathbb{R}^n$ where $r_i = \frac{\sum_{j=1}^{m} x_i(j)}{m} \in \mathbb{R}$ is the expected return of the security i, $i = 1, 2, \ldots, n$. The expected return of the portfolio is $r_p \in [\min(r), \max(r)] \subseteq \mathbb{R}$ and the variance of the marketed space is $\sigma^2 = \sum_{i=1}^{n}\sum_{i=1}^{n} x_i x_j \sigma_{ij}$ where $\sigma_{ij} = \rho_{ij}\sigma_i\sigma_j$ is the variance and ρ_{ij} is the correlation of i and j securities and σ_i is the variance of i security. That is, $\sigma^2 = X^{\mathrm{T}}CX$ where $C \in \mathbb{R}^{n\times n}$ is the covariance matrix of the marketed space X.

In the TV-MVPS we define the number $\tau \le m - 1$, $\tau \in \mathbb{N}$, where τ implies the 'number of time periods'. τ is used for the calculation of the moving average and it is also known as window size. A moving average (MA) is a calculation for analyzing data points by creating a series of averages of the complete data set of different subsets. In technical analysis of financial data such as stock prices, returns or volumes of trading the moving average is used as a technical indicator that combines price points of an instrument over a specified time frame divided by the number of data points τ in order to give a single trend line. Hence, a moving average is primarily a lagging indicator and, for that reason, it is one of the most popular tools for technical analysis.

The exponentially weighted moving average of the previous τ data is called Exponential Moving Average (*EMA*). Analysts compute *EMA* as

$$EMA_t(\lambda) = \frac{\sum_{i=0}^{\tau-1} \lambda^i x(t-i)}{\sum_{i=0}^{\tau-1} \lambda^i} \tag{4.107}$$

The formula for *EMA* can be written in a recursive form that can greatly facilitate and accelerate the computation of *EMA* in practice

$$EMA_t(\tau) = (1-\lambda)x(t) + \lambda EMA_{t-1}(\tau). \tag{4.108}$$

In the formula above, $(1-\lambda)$ determines the weight of the last closing price in the computation of the current *EMA*, whereas λ determines the weight of the previous *EMA* in the computation of the current *EMA*. Commonly, the value of the parameter λ is given by

$$\lambda = 1 - \frac{2}{\tau+1}. \tag{4.109}$$

The chosen period depends on the type of interest movement, for example short, moderate, or long-term. Short-term averages respond quickly to changes in the price, while long-term averages are slow to react. Moving average levels can be viewed in financial terms as support in a falling market or resistance in a rising market. In general, there exist several types of moving averages (see [173]). In this section, we use only one type, the exponential moving average (*EMA*). All the rest types of moving averages can be applied to TV-MVPS similarly to *EMA*.

The TV-MVPS comprises from $m-\tau$ in number consecutive values of an *MA* with τ in number observations for each time period. The time $t \in [1, m-\tau]$ implies the new price that it comes into the reckoning of the *MA*. Hence, the expected return of the marketed space is $r(t) = [r_1(t), r_2(t), \ldots, r_n(t)] \in \mathbb{R}^n$ where

$$r_i(t) = \frac{\sum_{j=t}^{t+\tau-2} \lambda^{j+1-t} x_i(j-1)}{\sum_{j=1}^{\tau-1} \lambda^j} \in \mathbb{R} \tag{4.110}$$

is the expected return of the security i, $i = 1, 2, \ldots, n$. Obviously, the $r_i(t)$ is an *EMA* and the TV-MVPS problem is built-up on the $r(t)$ for every t. So, the expected return of the portfolio is $r_p(t) \in [\min(r(t)), \max(r(t))] \subseteq \mathbb{R}$, the variance of the marketed space is

$$\sigma^2(t) = \sum_{i=1}^{n} \sum_{i=1}^{n} x_i(t : t+\tau) x_j(t : t+\tau) \sigma_{ij}(t) \tag{4.111}$$

where $\sigma_{ij}(t) = \rho_{ij}(t)\sigma_i(t)\sigma_j(t)$ is the variance and $\rho_{ij}(t)$ is the correlation of $x_i(t : t+\tau)$ and $x_j(t : t+\tau)$ and $\sigma_i(t)$ is the variance of $x_i(t : t+\tau)$. That is,

$$\sigma^2(t) = X(t : t+\tau, :)^{\mathrm{T}} C(t) X(t : t+\tau, :) \tag{4.112}$$

where $C(t) = cov(X(t : t+\tau, :)) \in \mathbb{R}^{n \times n}$ is the covariance matrix of the marketed space $X(t : t+\tau, :)$ at time t. The optimal mean-variance portfolio is $\eta(t) = [\eta_1(t), \eta_2(t), \ldots, \eta_n(t)]$ where $\eta_i(t)$ is the solution of (4.116)-(4.118) optimization problem for the security i, $i = 1, 2, \ldots, n$.

The purpose of the number τ is to keep steady the number of observations in the TV-MVPS for each t in $X(t : t+\tau, :)$ while t is moving across the interval $[1, m-\tau]$. Hence, the outcome of the TV-MVPS for each t can

be comparable with all the rest outcomes of every other $t \in [1, m-\tau]$ under the same number of observations. Note that, the expected return of the marketed space $r_i(t)$ is an *MA* and it also has the properties of an *MA*. That is, the bigger the τ of the TV-MVPS is, the smoother the $r_i(t)$ will be when t is moving across the interval $[1, m-\tau]$, because it filters out the 'noise' from random short-term price fluctuations. Moreover, it affects the optimal mean-variance portfolio $\eta(t)$ in the same way.

Similar to [43, 152, 155, 156], by interpolating $r(t)$ and $C(t)$ into continuous functions, we have that $r(t),\ C(t) \in C[0, m-\tau-1]$. In this way, we modify the discrete TV-MVPS of (4.116)-(4.118) to continuous-time. Note that the space $C[\kappa, \lambda]$ denotes the space of all continuous (real) functions defined on the closed interval $[\kappa, \lambda]$.

The TV-MVPSTC for all possible targets r_p (see [174]) is the solution to the next risk minimization and expected return maximization constrained problem:

$$\min_{\eta(t)} \quad \sum_i \sum_j \eta_i(t) \cdot \eta_j(t) \cdot \sigma_{ij}(t) \tag{4.113}$$

$$\text{s.t.} \quad \sum_i \eta_i(t) \cdot r_i(t) \geq r_p(t) \tag{4.114}$$

$$\eta_i(t) \in \mathbb{R}_0^+,\ \forall i, \tag{4.115}$$

where (4.113) is the variance $\sigma^2(t)$ of the portfolio $\eta(t)$ and $\mathbb{R}_0^+$ implies nonnegative real numbers.

This problem can also be written in the TVQP problem form as follows:

$$\min_{\eta(t)} \quad \eta^{\mathrm{T}}(t) \cdot C(t) \cdot \eta(t) \tag{4.116}$$

$$\text{s.t.} \quad -r^{\mathrm{T}}(t) \cdot \eta(t) \leq -r_p(t) \tag{4.117}$$

$$\mathbf{0}_n \leq \eta(t) \leq +\varpi\, \mathbf{1}_n, \tag{4.118}$$

where $C(t)$ is the covariance matrix of $X(t)$ and the constant $\varpi \gg 0$ is the numerical representation and $+\infty$ replacement, large enough for implementation purposes.

4.6.2 The TV-MVPSTC with Cardinality Costraint Problem

Transaction costs can be used to illustrate a variety of costs, such as bid-ask spreads, fund loads, taxes or even brokerage commissions. As set out in [192], we assume that the transaction costs are separable in this section. More precisely, let α^-, α^+ be the cost prices associated with selling and buying asset i and β^-, β^+ the fixed costs prices related to selling and buying the asset i. The function of fixed-plus-linear TV transaction cost is defined as

$$\phi_i(t) = \begin{cases} 0 & ,\eta_i(t) = \eta_i(t-1) \\ \beta_i^+ + \alpha_i^+(\eta_i(t) - \eta_i(t-1))x_i(t) & ,\eta_i(t) > \eta_i(t-1) \\ \beta_i^- + \alpha_i^-(\eta_i(t-1) - \eta_i(t))x_i(t) & ,\eta_i(t) < \eta_i(t-1) \end{cases} \tag{4.119}$$

where $x_i(t)$ is the price of the i stock at time t. Note that, except for zero fixed costs, this function is nonconvex.

As presented in [155], a related problem to (4.116)-(4.118) is reducing the overall transaction costs that are liable to portfolio restrictions. Of all potential transactions which produce portfolios that achieve a granted

expected return and meet portfolio constraints, we do want to pick those that will produce the least overall cost. Consequently, the TV-PSTC problem is formulated in the following form:

$$\min_{\eta(t)} \quad \phi(t) \tag{4.120}$$

$$\text{s.t.} \quad \sum_i \eta_i(t) \cdot r_i(t) \geq r_p(t) \tag{4.121}$$

$$\eta_i(t) \in \mathbb{R}_0^+, \ \forall i, \tag{4.122}$$

where (4.120) is the sum of the transaction costs related to each of the individual trades.

In order to avoid over-diversification and, hence, reduce the return of portfolio to zero, we set a cardinality constraint to our problem. That is, for a given space of marketed securities X of n securities, we only diversify in $K < n$, $K \in \mathbb{N}$, securities. A constant number K implies the number of securities the investor can hold. To formulate the cardinality constraint mathematically, we consider the binary variables $z_1, z_2, \ldots, z_n$ to denote which security is present in the portfolio. These binary variables can either hold a value of zero or one. That is, $z_i = 0$ means that the investor does not hold the security i and $z_i = 1$ means that the investor holds the security i, $i = 1, \ldots, n$. Consequently, the cardinality constraint can be formulated as follows:

$$\sum_{i=1}^{n} z_i \leq K. \tag{4.123}$$

Combining the two aforementioned problems of (4.116)-(4.118) and (4.120)-(4.122) along with equation (4.123), the TV-MVPSTC-CC problem is formulated in the TVNLP form as follows:

$$\min_{\eta(t)} \quad \eta^{\mathrm{T}}(t) \cdot C(t) \cdot \eta(t) + \phi(t) \tag{4.124}$$

$$\text{s.t.} \quad -r^{\mathrm{T}}(t) \cdot \eta(t) \leq -r_p(t) \tag{4.125}$$

$$\sum_{i=1}^{n} z_i(t) \leq K \tag{4.126}$$

$$\mathbf{0}_n \leq \eta(t) \leq X^{\mathrm{T}}(t) \cdot \eta(t-1) \cdot \left[\frac{1}{x_1(t)}, \ldots, \frac{1}{x_n(t)} \right], \tag{4.127}$$

where $C(t)$ is the covariance matrix of $X(t)$. The algorithmic procedure which defines the MATLAB function in (4.124) is given in Algorithm 9 in Appendix.

Algorithm 9 Algorithmic procedure for (4.124).

Input: The covariance matrix $C = C(t)$, the stock prices $X = X(t)$, the portfolio $\eta = \eta(t)$ and the previous $\eta_{-1} = \eta(t-1)$, the fixed costs β^-, β^+ and the costs rates α^-, α^+.

1: **function** f = minfunc(η, η_{-1}, C, X, β^-, β^+, α^-, α^+)

2: Set $f = \eta' C\eta$+sum($(\eta > \eta_{-1})(\beta^+ + \alpha^+(\eta - \eta_{-1})X) + (\eta < \eta_{-1})(\beta^- + \alpha^-(\eta_{-1} - \eta)X)$)

3: **return** f

4: **end function**

Output: The outcome of the function in (4.124).

4.6.3 Penalty Parameter

The following penalty function is used and minimized (see Sections 2.1.2 and 4.4.2):

$$Q(y, R) = f(y) + \Omega(R, d(y), r(y)), \tag{4.128}$$

where Ω is the penalty term and $f(y)$ is the objective function. Moreover, R is a set of penalty parameters, $q(y)$ is the inequality constraint function and $r(y)$ is the equality constraint function.

As mentioned before, we tackle the TV-MVPSTC-CC problem through an evolved version of BAS algorithm. Since BAS is directly applicable only to unconstrained optimization, it is necessary to use some additional methods that will keep solutions in the feasible region. Our modified version of BAS incorporates the aforementioned penalty function. Therefore, we just use penalty function (4.128) and the initial value of the penalty parameter is settled by the user. Note that R remains constant over all the sequences generated by BAS. We can therefore build an even more effective modified BAS algorithm than the primary one by simply considering a penalty function as shown in (4.6.4). This approach helps BAS to manage convex or nonconvex constraints more effectively. Nevertheless, the concept of the BAS algorithm remains similar to the one set out in [39].

We merge the bracket operator penalty with the penalty function

$$\Omega = R\langle d_j(y)\rangle^2, \tag{4.129}$$

where $\langle s\rangle = s$, if s is negative, otherwise $\langle s\rangle = 0$. The bracket operator is an external penalty condition, since the infeasible points are inserted with a negative value. This operator is mainly used to deal with the constraints on inequality. The algorithmic procedure, that handles the equality/inequality constraints, is shown in Algorithm 10 as a MATLAB function.

Algorithm 10 Algorithmic procedure that handles the equality/inequality constraints.

Input: The requirements of Algorithm 9 plus the cardinality constraint number K, the penalty parameter R, the lower limit η^- and upper limit η^+ of (4.127).

1: **function** Q = penfunc(η, η_{-1}, C, r, r_p, X, K, R, η^-, η^+, β^-, β^+, α^-, α^+)
2: Set $\Omega = R(\text{sum}((r\eta > r_p)(-r\eta + r_p)^2 + (\eta^- > \eta)(\eta^- - \eta)^2 + (\eta > \eta^+)(\eta - \eta^+)^2)) + R^2(\text{sum}(x > 0) > K)$
3: Set Q =minfunc($\eta, \eta_{-1}, C, X, \beta^-, \beta^+, \alpha^-, \alpha^+$) + Ω
4: **return** Q
5: **end function**

Output: The outcome of Penalty Function.

4.6.4 BAS Approach on the TV-MVPSTC-CC Problem

Algorithm 11 describes the BAS approach on the TV-MVPSTC-CC problem. Furthermore, Algorithm 11 includes function `minfunc`, which implements Algorithm 9 as well as function `penfunc`, which implements Algorithm 10. Also, the dots in these functions' input arguments imply that the remainder of the input arguments remain as stated in their algorithms.

Algorithm 11 BAS approach on TV-MVPSTC-CC.

Input: The covariance matrix $C = C(t)$, the expected return $r = r(t)$ and the target r_p, the stock prices $X = X(t)$, the previous portfolio $\eta_{-1} = \eta(t-1)$, the fixed costs β^-, β^+ and the costs rates α^-, α^+, the cardinality constraint number K, the penalty parameter R and the lower limit η^- and upper limit η^+ of (4.127). In addition, the parameters d, δ, tol and $kmax$.

1: **function** $[\eta, f_\eta]$ = basfunc(η_{-1}, C, r, r_p, X, K, R, η^-, η^+, β^-, β^+, α^-, α^+, d, δ, tol, $kmax$)
2: Set $y_1 = \eta_{-1}$ and y_2 =nan.ones(size(η_{-1}))
3: Set $k = 0$ and len =length(η_{-1})
4: **while** $k < kmax$ **OR** $\left\|\text{penfunc}(y_2,\ldots) - \text{penfunc}(y_1,\ldots)\right\|_2 > tol$ **do**
5: Set b =rands(k, 1) and $b = \frac{b}{2^{-52}+\|b\|}$
6: Set $y_r = y_1 - db$ and $y_l = y_1 + db$
7: Set $y = |y_1 + \delta b(\text{sign}(\text{penfunc}(y_r,\ldots) - \text{penfunc}(y_l,\ldots)))|$
8: **if** penfunc($y,\ldots$) <penfunc($y_1,\ldots$) **then**
9: Set $y_2 = y_1$ and $y_1 = y$
10: **end if**
11: Set $d = 0.991d + 0.001$ and $\delta = 0.991\delta$
12: **end while**
13: **return** $\eta = y_1$, f_η =minfunc($y_1,\ldots$).
14: **end function**

Output: $\eta = \eta(t)$, f_η.

4.6.5 Penalty Algorithm for the Modified BAS Algorithm

We initially implement a supplementary algorithm 12 that creates the TVNLP problem of (4.124)-(4.127) variables. Furthermore, the vector $noep$ and the function `omega`, which are include in algorithm 12, are used in order to unclasp the problem of nonexistent observations between periods of the same division. A detailed analysis about the vector $noep$ and the function `omega` can be found in [155].

Algorithm 12 Algorithm for the TVNLP problem of subsection 4.6.2.

Input: The $data$, which is a time-series as a vector of n prices, the time t and the vector $noep$, which contains the number of observations in each period.

1: **function** $[\eta^-,\eta^+,A,r_p,X_p,C]$ = problem($noep$,t,f_C,f_r,X,η_{-1})
2: Set ω =omega($noep, t$), $C = f_C(\omega t)$, $r = f_r(\omega t)$, $A = -r'$ and $X_p = X(\omega t)$
3: Set $r_p = -\min(r)$, η^- =zeros(length(X_p), 1) and $\eta^+ = X_p'\eta_{-1}./X_p$
4: **end function**

Output: The η^-,η^+,A,X_p,C for the time t.

The innovation of Algorithm 13 in comparison with the BAS version presented in [155] is that the penalty

parameter R is adjusted automatically. That is the user does not have to intervene and adjust the value of R instead Algorithm 13 finds the suitable value of R.

Algorithm 13 Penalty-BAS algorithm.

Input: The requirements of Algorithm 12 plus the cardinality constraint number K.

function $[x_{best},f_{best}]$ = PBAS($noep$,t,f_C,f_r,X,η_{-1},K)
  Set $[\eta^-,\eta^+,A,r_p,X_p,C]$ = problem($noep$,t,f_C,f_r,X,η_{-1})
  Set $tol_1 = tol_2 = 1e-4$, $R_0 = 0.1$, $c = 100$, $T = 1$, $x = [\ ]$, $x(:,T) = \eta_{-1}$ and $R(T) = R_0$
  Set $s_0 = @(u)\text{sum}((-Au + r_p < 0).(-Au + r_p) + (\eta^- - u > 0).(\eta^- - u) + (u - \eta^+ > 0).(u - \eta^+) + (u - \eta_{-1} = 0).(u - \eta_{-1})) + (\text{sum}(u > 0) > K)$
  Set $s = s_0(x(:,T))$
  Set bee= $@(w_1, w_2)$basfunc(w_1, C, r, r_p, X_p, K, w_2, η^-, η^+, β^-, β^+, α^-, α^+, d, δ, tol, $kmax$)
  if $s > 0$ **then**
    Set $Q(1,T)$ = penfunc($x(:,T)$, η_{-1}, C, r, r_p, X_p, K, $R(1,T)$, η^-, η^+, β^-, β^+, α^-, α^+)
    Set $G(:,T)$ = bee($x(:,T)$,$R(1,T)$), $x(:,T+1) = G(:,T)$ and $T = T + 1$
    Set $Q(1,T)$ = penfunc($G(:,T-1)$, η_{-1}, C, r, r_p, X_p, K, $R(1,T-1)$, η^-, η^+, β^-, β^+, α^-, α^+)
    Set $x(:,T-1) = x(:,T)$
  else
    Set $Q(1,T)$ = penfunc($x(:,T)$, η_{-1}, C, r, r_p, X_p, K, 0, η^-, η^+, β^-, β^+, α^-, α^+)
    Set $G(:,T)$ = bee($x(:,T)$,0) and $T = T + 1$
    Set $Q(1,T)$ = penfunc($G(:,T-1)$, η_{-1}, C, r, r_p, X_p, K, 0, η^-, η^+, β^-, β^+, α^-, α^+)
    Set $x(:,T) = G(:,T-1)$
  end if
  Set err =abs($Q(1,T)$), $R(1,t) = cR(1,T-1)$ and $x_{best}(1,:) = G(:,T-1)$
  while $err > tol_2$ **do**
    Set $G(:,T)$ = bee($x(:,T)$,$R(1,T)$), $x(:,T+1) = G(:,T)$, $T = T + 1$ and $R(1,T) = cR(1,T-1)$
    Set $Q(1,T)$ = penfunc($G(:,T-1)$, η_{-1}, C, r, r_p, X_p, K, $R(1,T-1)$, η^-, η^+, β^-, β^+, α^-, α^+)
    Set err = abs($Q(1,T) - Q(1,T-1)$), $s = s_0(x(:,T))$ and $x_{best}(1,:) = G(:,T-1)$
    if abs(s) < tol_1 **then**
      break
    else if abs($x(:,T) - x(:,T-1)$) < $1e-4$ **then**
      break
    end if
  end while
  Set f_{best} = minfunc(x_{best}, η_{-1}, C, X_p, β^-, β^+, α^-, α^+)
end function

Output: The x_{best} and f_{best} for the time t.

4.6.6 The TVPBAS Algorithmic Method

Algorithm 14 is the main and most important algorithm, since it describes the complete procedure for solving the continuous TV-MVPSTC-CC problem in discrete time and produces its online solution. Note that Algorithm 14 can make use any of the interpolation algorithms presented in Appendix A.

Algorithm 14 The TVPBAS algorithmic method for solving the problem of TV-MVPSTC-CC.

Input: The data, which is a time-series as a vector of n prices, and the moving average's number of time periods $\tau \leq m-1$, $\tau \in \mathbb{N}$. The $tspan = [t_{start}, t_{end}]$, where $t_{start}, t_{end} \in$ [0,length(data)−1], the parameter γ which divides $tspan$ corresponding to γ in a number equal steps, and the vector $noep$ which contains the number of observations in each period. Furthermore, a given portfolio η_p and the cardinality constraint number K.

Construct C and r from Algorithm 24.

Construct f_C, f_r and f_X from Algorithm 27 or Algorithm 28.

function [x,y] = TVPBAS(η_p,γ,$tspan$,$noep$,f_C,f_r,f_X, K)

Set $t = tspan(1) : \frac{1}{\gamma} : tspan(2)$

Set n =length(η_p) and tot =length(t)

Set x =zeros(n, tot) and y =zeros($1, tot$)

Set $[x(:,1), y(1)]$= PBAS($noep$,t,f_C,f_r,X,η_p,K)

for $i = 2 : tot$ **do**

Set $[x(:,i), y(i)]$= PBAS($noep$,t,f_C,f_r,X,$x(:,i-1)$,K)

end for

end function

Output: The online solution of the TV-MVPSTC-CC Problem.

4.7 A binary Markowitz-based Portfolio Selection Problem

In this section, the binary version of BAS (BBAS), which is described in [43], is modified by adding a V-shaped transfer function as presented in [193, 194], to solve the BBAS problem of trapping in local minima. In this way, we introduce the modified V-shaped transfer function-based binary BAS (VSBAS) algorithm, which is a more efficient version of BBAS in the case of large input data. VSBAS is compared against BBAS, the binary bat algorithm (BBA), the binary genetic algorithm (BGA) and the modified V-shaped transfer function-based binary particle swarm optimization (VPSO) on a Markowitz-based portfolio selection problem using real-world large data input.

The portfolio selection problem (or portfolio optimization) alludes to the optimal distribution of budget on the available stocks according to some objective (see [1–3, 5, 195]). A common objective is the expected mean-return maximization, and the risk (variance) minimization. This objective was presented not many decades ago from Markowitz's Modern Portfolio Theory [167], and has been studied extensively ever since. Contemporary references on Markowitz-based portfolio selection problems are [168, 196, 197]. For instance, the authors of [196] explore the optimization of the portfolio under the Hidden Markov model's investor sentiment states and

over a different time period. By using two methods, namely the Markowitz and Bayesian mean-variance, their findings show that the Bayesian efficient frontier of regular and Islamic stock portfolios is influenced by the state of sentiment of the investor and the time period. In [197], a Markowitz-based model is considered in a privacy-conscious manner for outsourcing to a public cloud. The proposed model uses encryption operations for location-scrambling and value-alteration that can protect the privacy well of its input/output. The results therein display that when solving the proposed model, the investor will obtain a tremendous amount of computational benefit and cloud complexity consistent in comparison with that of the original model.

In recent years, many Markowitz-based portfolio selection problems have been proposed and studied, and they have been addressed using a variety of methods, including a multi-objective evolutionary algorithm (MOEA) in [198], a standard solver (Cplex) in [199], an augmented ϵ-constraint method (AUGMECON) in [200, 201], a multi-objective optimization genetic algorithm (MOO-GA) in [202], a non-dominated sorting genetic algorithm (NSGA-II) in [203], a BAS algorithm in [156]. The proposed approach has several new features when compared to previous research on Markowitz-based portfolio selection problems in general. New features and similarities with the most relevant papers are presented in Tab. 4.1.

Table 4.1: Interconnection with previously published papers that are most closely related

Reference	TV Model	Cardinality	Transaction Costs	Financial Market	Solution Approach
[198]	No	Yes	Yes	Hang Seng, S&P, Nikkei	MOEA
[199]	No	Yes	Yes	Hang Seng, S&P 500, Nikkei 225, Russell 3000	Cplex
[200]	No	Yes	Yes	Eurostoxx 50	AUGMECON
[201]	No	Yes	Yes	S&P 500	AUGMECON
[202]	No	No	Yes	Bovespa, S&P 500	MOO-GA
[203]	No	Yes	Yes	S&P 100	NSGA-II
[156]	Yes	Yes	Yes	US Market	BAS
Proposed Approach	Yes	Yes	Yes	DJIA, CAC 40, US Market	VSBAS

4.7.1 V-Shaped Binary Beetle Antennae Search Algorithm

In this section, the VSBAS is described as the combination of two factors, namely the BBAS and the transfer function method.

On the one hand, we have the binary BAS of [43], which can be analyzed as follows. Assume that the position of the beetle, at i-th time moment, is a vector x_i, $i = 1, 2, \ldots$ and the concentration of odour, at position x, is the objective function $f(x)$, where the minimum value of $f(x)$ is correlated with the source point of the odour. Then a random searching path of the beetle defines the model of searching behavior as follows:

$$B = \text{round}(\text{rnd}(g, 1)), \tag{4.130}$$

where round(·) and rnd(·) implies a round and a random function respectively, and g implies the position's dimensions. To mimic the searching behaviors of the beetle's antennae, the right (x_R) and left (x_L) antennae are formulated as follows:

$$x_R = \begin{cases} 1, & x_i + B > 1 \\ 0, & x_i + B < 0 \end{cases}, \tag{4.131}$$

$$x_L = \begin{cases} 1, & x_i - B > 1 \\ 0, & x_i - B < 0 \end{cases}. \tag{4.132}$$

Furthermore, considering the candidate optimal solution (x_C) as follows:

$$x_C = \begin{cases} x_R, & f(x_R) < f(x_L) \\ x_L, & f(x_R) > f(x_L) \end{cases}, \tag{4.133}$$

the behavior of detecting can be formulated as follows:

$$x_i = \begin{cases} x_C, & f(x_C) < f(x_{i-1}) \\ x_{i-1}, & f(x_C) > f(x_{i-1}) \end{cases}. \tag{4.134}$$

On the other hand, transfer functions are commonly employed in population-based evolutionary algorithms to solve the problem of trapping in local minima and thus to improve their optimization ability [204]. In addition, a v-shaped family of transfer functions has been proposed and investigated in [193, 194], with findings indicating that the v-shaped family of transfer functions has merit for use in binary algorithms. The transfer function determines the possibility of changing the elements of a position vector from 0 to 1 and vice versa. In this way, the particles are forced to move in a binary space. To choose a transfer function according to [204], the following principles should be taken into account, for mapping velocity values to likelihood values. First, the span of a transfer function must be limited in the $[0, 1]$ interval, as it reflects the likelihood that a particle will change its location. Second, for a large absolute value of velocity, a transfer function should have a high likelihood of changing the location. Particles with high absolute values for their velocity are obviously far from the ideal solution, so in the next iteration, they could adjust their locations. Third, there should also be a small likelihood of a transfer function shifting the location for a small absolute value of the velocity. Fourth, a transfer function's return value should increase as the velocity increases. For going back to their previous positions, particles that are heading away from the best solution must have a greater chance of shifting their position vectors. Fifth, a transfer function's return value must decline as the velocity decreases. These five principles ensure that a transfer function can map the search procedure to a binary search space in a continuous search space, while retaining comparable concepts of the search for a specific evolutionary algorithm.

In the case of population-based evolutionary approaches, these five principles can be applied as presented in [193, 194]. Considering that x_{best} is the best solution attained so far and x_c is a candidate solution, the velocity vector v is formulated as follows:

$$v_i(t+1) = v_i(t) + (x_c - x_{best})F_i \tag{4.135}$$

where F_i indicates the frequency of i-th particle at iteration t and is formulated as follows:

$$F_i = F_{min} + (F_{max} - F_{min})\text{rnd}(1). \tag{4.136}$$

Furthermore, a v-shaped transfer function V and position updating rule in g-th dimension can be formulated as follows:

$$V(v_i^g(t)) = \left|\frac{2}{\pi}\arctan\left(\frac{\pi}{2}v_i^g(t)\right)\right| \tag{4.137}$$

$$x_i^g(t+1) = \begin{cases} \left(x_i^g(t)\right)^{-1}, & \text{rnd}(1) < V(v_i^g(t+1)) \\ x_i^g(t), & \text{rnd}(1) \geq V(v_i^g(t+1)) \end{cases}, \tag{4.138}$$

where the superscript $()^{-1}$ indicates the complement of $x_i^g(t)$.

In the case of BBAS, we may not have a population-based evolutionary approach, but we have a single beetle with two antennae. What we are trying to do is to v-shaped the outcome of its antennae. To accomplish this, (4.135)-(4.138) must be modified in such a way that they match the BAS concept while still remaining functional. Thus, we can make the following three changes in the aforementioned approach. First, since the term frequency is incompatible with the BAS concept, (4.136) is discarded and F_i is replaced by rnd(1) in (4.135). Second, since we do not have particles in the BAS concept, the term i-th particle is removed from (4.135), (4.137) and (4.138). Third, in the concept of a population-based evolutionary algorithm, the population evolves over time t and (hopefully) reveals better solutions. In contrast to population-based evolutionary algorithm concepts, the BAS concept has no evolved population, so $t = 0$. Its worth noting that the initial velocity $v(0)$ is always equal to 0 in [193, 194]. As a result, the term t is removed from (4.135), (4.137) and (4.138). Based on that, we can rewrite the velocity formula of (4.135) as follows:

$$v = (x_c - x_{best})\text{rnd}(1), \tag{4.139}$$

and, hence, the v-shaped transfer function V of (4.137) and the position updating rule of (4.138) in g-th dimension can be simplified and formulated as follows:

$$V(v) = \left|\frac{2}{\pi}\arctan\left(\frac{\pi}{2}v^g\right)\right| \tag{4.140}$$

$$x^g = \begin{cases} \left(x_c^g\right)^{-1}, & \text{rnd}(1) < V(v) \\ x_c^g, & \text{rnd}(1) \geq V(v) \end{cases}. \tag{4.141}$$

Furthermore, as proposed in [193, 194], the balancing between these techniques is controlled by a parameter r. Consequently, (4.141) can be modified as follows:

$$x^g = \begin{cases} \left(x_c^g\right)^{-1}, & r < \text{rnd}(1) < V(v) \\ x_c^g, & r < \text{rnd}(1) \quad \text{and} \quad \text{rnd}(1) \geq V(v) \\ x_{best}^g, & r > \text{rnd}(1) \end{cases}. \tag{4.142}$$

The process of v-shaping the outcome of a beetle's antennae element by element is shown in (4.139)-(4.142), whereas the following Alg. 15, which contains MATLAB routines, describes the process of v-shaping the outcome of a beetle's antennae for all of its elements at once.

Algorithm 15 Binary V-Shaped transfer function for BBAS antennae.

Input: The best solution attained so far x_{best} and a candidate solution x_c.

function x = VSTF(x_{best}, x_c)
  Set g =length(x_c), $r = 0.1$ and b =rand(g, 1)
  Set $v = (x_c - x_{best}).b$ and V =abs((2/pi)atan((pi/2)v))
  Set $x_c(b < V) =\sim x_c(b < V)$ and $x_c(b > r) = x_{best}(b > r)$
  return $x = x_c$
end function

Output: The binary V-Shaped optimal solution x.

By adding the property of v-shaping the outcome of a beetle's antennae to BBAS, the problem of trapping in local minima can be handled efficiently, improving BBAS accuracy. The VSBAS approach is described in the following Alg. 16.

Algorithm 16 Algorithm of VSBAS.

Input: An objective function $f(x_i)$, where $x_i \in \mathbb{R}^n$, and initialize the parameters x_0 and i.

while $i < I_{max}$ **OR** (stop criterion) **do**
  Define the vector unit B according to (4.130)
  Explore the variable space with two kinds of antennae according to x_R of (4.131) and x_L of (4.132)
  V-Shaped x_R and x_L by setting x_R =VSTF(x_i, x_R) and x_L =VSTF(x_i, x_L), respectively
  Calculate the candidate optimal solution (x_C) in line with (4.133)
  Update x_i according to (4.134)
  if $f(x_i) < f_{best}$ **then**
    Set $f_{best} = f(x_i)$ and $x_{best} = x_i$
  end if
  Set $i = i + 1$
end while

Output: x_{best}: the optimal solution, f_{best}: the value of $f(x_{best})$.

4.7.2 The BMPS Problem

Assume that the market space is $X(t) = [x_1(t), x_2(t), \ldots, x_n(t)] \in \mathbb{R}^n$ where $x_j(t) \in \mathbb{R}$ is the asset j, $j = 1, 2, \ldots, n$, at time $t = 1, 2, \ldots, m$. Given the number $\beta \in \mathbb{N}$, which denotes the past values (or delays), we set $r(t) = [r_1(t), r_2(t), \ldots, r_n(t)] \in \mathbb{R}^n$ the expected return of the market space, where $r_j(t) = \frac{\sum_{h=0}^{\beta-1} x_j(t-h)}{\beta \max(x_j(t-\beta-1:t))} \in \mathbb{R}$ is the asset's j normalized expected return at time $t = \beta + 1, \beta + 2, \ldots, m$. In the sequel, max(·) denotes the maximum value and the colon operator (:) is being used to create vector subscripts. More precisely, a colon by itself in place of a subscript denotes all the elements of the corresponding row or column, while $(k_1 : k_2)$ denotes all the elements

from k_1 to k_2 of the corresponding row or column. Also, $C(t) \in \mathbb{R}^{n\times n}$ denotes the covariance matrix of the normalized market space $X(t)$ at time $t = \beta+1, \beta+2, \ldots, m$. That is, $C(t) = \mathrm{cov}\Big(X(t-\beta-1:t,:) \odot \frac{1}{\max(x_j(t-\beta-1:t,:))}\Big)$, where cov$(\cdot)$ denotes the covariance matrix and $\odot$ implies the Hadamard (or element-wise) product. In addition, $\eta(t) \in \mathbb{R}^n$ denotes the optimal portfolio of the time-varying BMPS problem and $\eta_j(t) \in \mathbb{R}$ denotes the j-th asset of the optimal portfolio $\eta(t)$.

The purpose of the number β is to keep steady the number of observations in the time-varying BMPS problem for each $t = \beta+1, \beta+2, \ldots, m$ in $X(t)$. As a result, $\eta(t)$ for each t can be comparable with all the rest outcomes of every other t under the same number of observations. It is worth noting that the market space's expected return $r_j(t)$ is a moving average and thus has the properties of a moving average. That is, the bigger the β is, the smoother the $r_j(t)$ will be when t is moving across the values $t = \beta+1, \beta+2, \ldots, m$, because it filters out the "noise" from random short-term price fluctuations. It also has the same effect on the optimal portfolio $\eta(t)$.

Furthermore, we set a cardinality limit on the BMPS problem to prevent over-diversification and, thus, minimize the return of the portfolio to zero. As a result, we only diversify between $\mu^- \le \mu^+ < n$, with $\mu^-, \mu^+ \in \mathbb{N}$, assets for a given market space X of n assets. That is, the maximum number of assets that an investor can hold is an integer between μ^- and μ^+. To mathematically formulate the cardinality limitation, the binary variables λ_j, $j = 1, \ldots, n$, are considered to imply which asset is present in the portfolio. These binary variables are capable of retaining either a value of zero or one. That is, $\lambda_j = 1$ if the investor holds the asset j (i.e. $\eta_j(t) \neq 0$), otherwise $\lambda_j = 0$. Therefore, the cardinality constraint can be formulated as follows:

$$\mu^- \le \sum_{j=1}^{n} \lambda_j \le \mu^+. \tag{4.143}$$

In addition, a number of costs, such as fund loads, bid-ask spreads, taxes, brokerage commissions etc. can be defined as transaction costs. As determined in [192], we let θ^-, θ^+ the fixed rates prices connected with selling and purchasing the asset j and ζ^-, ζ^+ be the cost rates connected with selling and purchasing asset j. In this way, the transaction costs connected with selling and purchasing asset are separable. The time-varying transaction cost function is formulated as follows

$$\gamma_j(t) = \begin{cases} 0 & ,\eta_j(t) = \eta_j(t-1) \\ \theta_j^+ + \zeta_j^+(\eta_j(t) - \eta_j(t-1))x_j(t) & ,\eta_j(t) > \eta_j(t-1) \\ \theta_j^- + \zeta_j^-(\eta_j(t-1) - \eta_j(t))x_j(t) & ,\eta_j(t) < \eta_j(t-1) \end{cases}. \tag{4.144}$$

Note that, except for zero fixed costs, (4.144) is nonconvex.

Finally, by setting $R(t) \in [\mu^- \min(r(t)), \mu^+ \max(r(t))] \subseteq \mathbb{R}$ as the target expected return of the portfolio, the time-varying BMPS problem can be formulated in an integer linear programming form as follows:

$$\min_{\eta(t)} \quad \eta^{\mathrm{T}}(t) \cdot C(t) \cdot \eta(t) + \gamma(t) \tag{4.145}$$

$$\text{s.t.} \quad -r^{\mathrm{T}}(t) \cdot \eta(t) \le -R(t) \tag{4.146}$$

$$\mu^- \le \sum_{j=1}^{n} \lambda_j(t) \le \mu^+ \tag{4.147}$$

$$\eta_j(t) = \{0, 1\}, \tag{4.148}$$

4.8 The TV Tangency Portfolio under Nonlinear Constraints Problem

The mean-variance optimization theory of Markowitz provides a mechanism for selecting assets (or securities) portfolios that trades off expected returns and risk of prospective portfolios. For a given level of risk, investors that utilize mean-variance resolution to maximize their expected return always prefer portfolios that are on the CML. If a feasible portfolio has the highest expected return among all portfolios with the same variance, or if it has the lowest variance among all portfolios with at least a specific expected return, it is said to be efficient. The efficient frontier of the portfolio universe is made up of a collection of efficient portfolios. The most efficient portfolio, dubbed the tangency portfolio, is found at the point where the CML intersects with the efficient frontier.

Any portfolio η with one or more risky assets and one risk-free asset may have a linear connection between its expected return r_p and its risk σ_p, according to Sharpe Ratio (SR) [205]. Mathematically, this can be stated in the following way:

$$r_p = r_f + S_\eta \sigma_p, \tag{4.149}$$

where r_f denotes the risk-free asset's return and S_η denotes the portfolio's SR, which is the risk premium per risk unit.

The tangency portfolio optimization given in [206] is the foundation of our approach to the TV-TPNC problem. A rationalistic risk averse investor's endowment will be divided, with a proportion γ invested in a risk-free asset and the rest $(1-\gamma)$ in a time-varying portfolio of risky assets $\eta(t)$, $t \subseteq \mathbb{N}$, whereas $S_p(t)$ is determined by the composition of $\eta(t)$, which is based on the common capital market hypothesis of one risk-free and many risky assets. Consider the market space $X(t) = [x_1(t), x_2(t), \ldots, x_n(t)] \in \mathbb{R}^n$ that contains n assets prices, the investor would choose the weights $\eta_i(t)$, for the assets $i = 1, 2, \ldots, n$, included in the portfolio $\eta(t) = [\eta_1(t), \eta_2(t), \ldots, \eta_n(t)] \in \mathbb{R}^n$ to optimize $S_p(t)$. It is worth emphasizing that γ reflects the investor's risk aversion, and that all investor's $\eta_i(t)$ must be the alike. As a result, the time-varying tangency portfolio $\eta(t)$ can be computed without considering the risk aversion or utility function of the investor.

Moreover, investors prefer portfolios with a lower number of different assets since handling portfolios with a big number of various assets may be time intensive [207]. A key consideration during the portfolio selection process is that most of a portfolio's risk diversification may be achieved with a small but well-selected collection of assets [206]. Mathematically, a cardinality constraint (CC) can be used to any portfolio optimization problem to achieve this. Thus, the fixed number K denotes the exact amount of assets an investor can own, avoiding over-diversification, while CC is expressed as the binary vector $D(t) = [D_1(t), D_2(t), \ldots, D_n(t)] \in \mathbb{R}^n$, which signify the assets in the portfolio and can have a value of 1 or 0, where $D_i(t) = 1$ signifies that the investor owns the asset i and $D_i(t) = 0$ signifies the opposite. Thus, the time-varying CC function can be formulated as follows:

$$D_i(t) = \begin{cases} 1, & \eta_i(t) > 0 \\ 0, & \eta_i(t) = 0 \end{cases} \tag{4.150}$$

MPT frequently considers an ideal market in which short sales are prohibited but shares are infinitely separable and hence may be sold in any non-negative partition, free of taxes and transaction costs (TC). TC can refer to a variety of expenses like fund loads, taxes, bid-ask spreads, brokerage charges, and so on. Inline with [192], we will consider θ^-, θ^+ the fixed charges prices generated from the sell and buy of an asset i, and ζ^-, ζ^+ the cost

charges generated from the sell and buy of an asset i. Thus, TC generated from the sell and buy of an asset are separate, and the time-varying TC function can be formulated as follows:

$$G_i(t) = \begin{cases} 0, & \eta_i(t) = \eta_i(t-1) \\ \theta^+ + \zeta^+(\eta_i(t) - \eta_i(t-1))x_i(t), & \eta_i(t) > \eta_i(t-1) \\ \theta^- + \zeta^-(\eta_i(t-1) - \eta_i(t))x_i(t), & \eta_i(t) < \eta_i(t-1) \end{cases} \tag{4.151}$$

and $G(t) = [G_1(t), G_2(t), \ldots, G_n(t)] \in \mathbb{R}^n$. Apart from the case of zero costs, (4.151) is nonconvex.

According to the aforementioned, if a market $X(t)$ of n assets exists, in which only K of them have to be included in $\eta(t)$, the TV-TPNC problem can be formulated as follows:

$$\max_{\eta(t)} \quad S_p(t) - \sum_{i=1}^{n} G_i(t) \tag{4.152}$$

s.t.

$$S_p(t) = \frac{r_p(t) - r_f(t)}{\sigma_p(t)} \tag{4.153}$$

$$r_p(t) = \sum_{i=1}^{n} \eta_i(t) r_i(t) \tag{4.154}$$

$$\sigma_p(t) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} \eta_i(t)\eta_i(t)\sigma_{ij}(t)} \tag{4.155}$$

$$\sum_{i=1}^{n} D_i(t) = K, \ \forall i, \tag{4.156}$$

where $r_i(t)$ signifies the expected return of asset i at time t, and $\sigma_{ij}(t)$ signifies the covariance among the expected returns of assets i and j at time t.

The following improvements are made to transform the TV-TPNC to an NLP problem and make it more realistic. We use past values (or delays) to construct the variance (risk), covariance matrix and expected return of the market $X(t)$. Representing the delays with the constant number $\beta \in \mathbb{N}$, we consider $r(t) = [r_1(t), r_2(t), \ldots, r_n(t)] \in \mathbb{R}^n$ the expected return of $X(t)$, where $r_i(t) = \sum_{z=0}^{\beta-1} (x_i(t-z))/\beta \in \mathbb{R}$ signifies the asset's i, $i = 1, 2, \ldots, n$, expected return at time t, and $C(t) \in \mathbb{R}^{n\times n}$ the covariance matrix of $X(t)$ based on β in number delays. In this way, we can set $r_p(t) = \eta^{\mathrm{T}}(t) r(t)$ and $\sigma_p(t) = \sqrt{\eta^{\mathrm{T}}(t) C(t) \eta(t)}$. It is worth noting that $X(t)$ contains both risk-free and risky assets. However, when it comes to investing, there is no such thing as an asset that is risk-free because nothing can be guaranteed 100 percent. As a result, in our model, risk-free assets are defined as market assets with a variance below a small fixed value α. Thus, setting $H(t) = [h_1(t), h_2(t), \ldots, h_n(t)]$, where $h_i(t) = 1$, if $\mathrm{Var}\big[h_i(t)\big] < \alpha$ with $\mathrm{Var}\big[J\big]$ signifying the variance of i, and $h_i(t) = 0$, otherwise, we have that $r_f(t) = \eta^{\mathrm{T}}(t)\big(H(t) \odot r(t)\big)$ with $\odot$ signifying the Hadamard (or element-wise) product. It is also worth noting that the price of each asset x_i is normalized inline with its β in number delays.

The TV-TPNC problem may be expressed in the next NLP formation based on the aforementioned analysis:

$$\min_{\eta(t)} \quad G^{\mathrm{T}}(t) \cdot \mathbf{1}_n - \frac{\eta^{\mathrm{T}}(t)\big(r(t) - H(t) \odot r(t)\big)}{\sqrt{\eta^{\mathrm{T}}(t) \cdot C(t) \cdot \eta(t)}} \tag{4.157}$$

s.t.

$$\eta^{\mathrm{T}}(t) \cdot \mathbf{1}_n = 1 \tag{4.158}$$

$$D^{\mathrm{T}}(t) \cdot \mathbf{1}_n = K \tag{4.159}$$

$$\mathbf{0}_n \le \eta(t) \le \mathbf{1}_n \tag{4.160}$$

4.8.1 The Semi-Integer Beetle Antennae Search Model

The computational procedures utilized to handle the given financial NLP problem in a brief period of time with great accuracy are the main emphasis of this paper. As a result, a hybrid algorithm called SIBAS is developed, which is based on a nature inspired algorithm called BAS, whose primary advantage is its low time consumption. SIBAS combines BAS and BBAS to better handle cardinality constrained NLP problems.

The SIBAS Algorithm

BAS is a nature inspired algorithm that finds the best solution to an optimization problem by mimicking the behavior of a beetle [39], while a binary type of BAS named BBAS was presented in [43]. Due to the fact that these algorithms are only applicable to optimization without constrains, a complementary procedure have to be used to keep solutions inside the acceptable range. The penalty method [9] is chosen as the supplementary procedure for manipulating nonconvex or convex constraints more effectively in this study.

The following penalty function is used and minimized (see Sections 2.1.2 and 4.4.2):

$$F(w) = f(w) + U(R, q(w)), \tag{4.161}$$

where $f(w)$ signifies the objective function. Also, $U(R, q(w))$ signifies the penalty term, where $q(w)$ denotes the inequality/equality constraint and R denotes a set of penalty parameters.

The SIBAS may be described as follows. At i-th time moment, consider the position of the beetle as a vector x_i, $i = 1, 2, \ldots$. Then, the gathering of odour is the objective functions $F_1(x)$ and $F_2(x)$ at position x. As a result, the minimum value of $F_1(x)$ and $F_2(x)$ is linked to the odour's source spot. Note the $F_1(x)$ is (4.161) with only the CC of the NLP problem, while $F_2(x)$ is (4.161) with all the rest inequality/equality constraints of the NLP problem. The model of seeking behavior is defined as follows by a random searching path of the beetle:

$$A = \text{round}(\text{rnd}(n, 1)), \tag{4.162}$$

where $\text{rnd}(\cdot)$ and $\text{round}(\cdot)$ signify a random and a round function respectively, while n signifies the position's dimensions. The right (x_R) and left (x_L) antennae are composed as bellow to replicate the seeking behaviors of the beetle's antennae:

$$x_R = \begin{cases} 1, & x_{i-1} + A > 1 \\ 0, & x_{i-1} + A < 0 \end{cases}, \tag{4.163}$$

$$x_L = \begin{cases} 1, & x_{i-1} - A > 1 \\ 0, & x_{i-1} - A < 0 \end{cases}. \tag{4.164}$$

Moreover, assuming the candidate optimal solution as bellow:

$$x_C = \begin{cases} x_R, & F_1(x_R) < F_1(x_L) \\ x_L, & F_1(x_R) > F_1(x_L) \end{cases}, \tag{4.165}$$

the behavior of detecting may be formulated as bellow:

$$x_i = \begin{cases} x_C, & F_1(x_C) < F_1(x_{i-1}) \\ x_{i-1}, & F_1(x_C) > F_1(x_{i-1}) \end{cases}. \tag{4.166}$$

Note that i signifies the iteration number. Given that y is the optimal solution of $F_1(\cdot)$, a new random seeking path is created for optimizing $F_2(\cdot)$. Hence, setting $g = \text{rnd}(n, 1)$ at position x_{i-1}, the random path is as bellow:

$$B = \frac{g}{2^{-52} + \|g\|}. \tag{4.167}$$

Imitating the antennae motions, we have:

$$x_L = x_{i-1} - dB, \quad x_R = x_{i-1} + dB, \tag{4.168}$$

where the detecting diameter of the antennae is denoted by d, which is related to the ability to exploit. In addition, considering the candidate optimal solution:

$$x_C = \|x_{i-1} + \delta B \text{sign}(F_2(x_R) - F_2(x_L))\| \odot y, \tag{4.169}$$

where the term δ refers to a size step that corresponds to the pace of convergence following an increase in i during the search. In this way, the optimal solution of $F_1(\cdot)$ is merged with the solution of $F_2(\cdot)$, while only specific elements of x_C are allowed to be modified. Hence, the behavior of detecting may be formulated as bellow:

$$x_i = \begin{cases} x_C, & F_2(x_C) < F_2(x_{i-1}) \\ x_{i-1}, & F_2(x_C) > F_2(x_{i-1}) \end{cases}. \tag{4.170}$$

Finally, the d and δ update rules are as follows:

$$\delta = 0.991\delta, \quad d = 0.991d + 0.001. \tag{4.171}$$

The SIBAS algorithm is shown in the diagram of Fig. 4.3.

SIBAS Approach on the TV-TPNC Problem and the Complete Process

Given the market dataset M, which comprises of assets prices time-series, the market space $X(t)$ along with the span of time-period t are determined based on the delays number β. In addition, $C(t)$ and $R(t) = r(t) - H(t) \odot r(t)$ can be constructed based on the analysis presented in this section. Setting the initial position of the beetle as the initial portfolio of the TV-TPNC problem as well as the penalty functions according to the analysis presented in subsection 4.8.1, the TV-TPNC problem of (4.157)-(4.160) can be solved with the SIBAS algorithm. More precisely, the penalty functions for the TV-TPNC problem of (4.157)-(4.160) can be written in MATLAB routines as follows:

$$F_1(\eta) = f(\eta) + R\text{^}2 * (\text{sum}(\eta > 0) \sim = K), \tag{4.172}$$

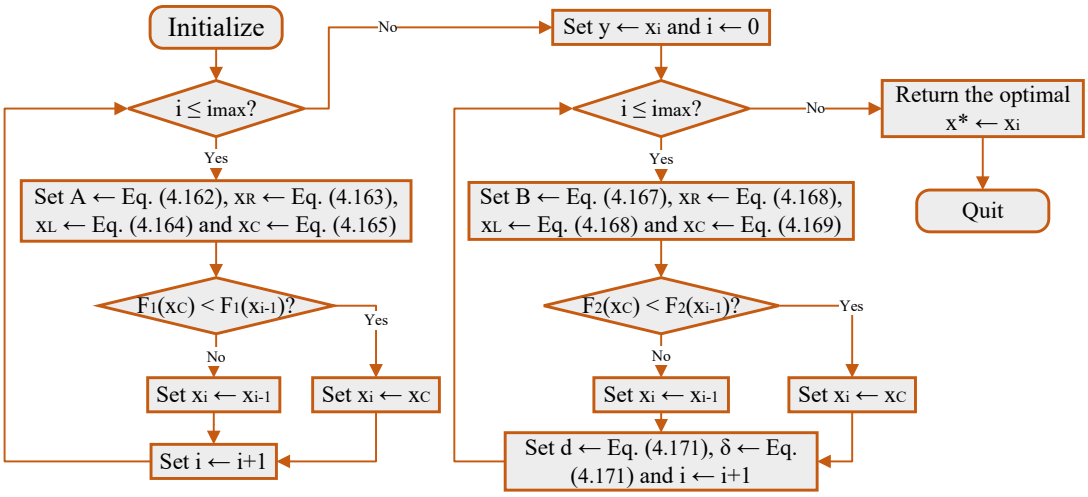


Figure 4.3: The SIBAS algorithm for cardinality constrained NLP problems.

$$\begin{aligned} F_2(\eta) = & f(\eta) + R * \text{sum}((\text{sum}(\eta) - 1 \sim = 0).(\text{sum}(\eta) - 1).\hat{}2 \\ & + (\eta < 0). * \eta.\hat{}2 + (\eta - 1 > 0). * (\eta - 1).\hat{}2), \end{aligned} \tag{4.173}$$

where sum$(\cdot)$ signifies the MATLAB routine for summing the elements of an input array and $f(\eta)$ is (4.157).

The complete process to solve the TV-TPNC problem of (4.157)-(4.160) using the SIBAS approach is presented in Alg. 17.

Algorithm 17 The complete process to solve the TV-TPNC problem of (4.157)-(4.160) using SIBAS.

Input: The market dataset M; the delays number β; the initial portfolio η_{in} and the parameter α.

1: Set $[m, n]$ =size(M), $t_{\text{end}} = m - \beta$, r =zeros(t_{end}, n), X =zeros(t_{end}, n) and $C\{t_{\text{end}}, 1\} = \{\,\}$
2: **for** $t = 1 : t_{\text{end}}$ **do**
3: Set $s = M(t : \beta + t - 1, :)$, $s = s./\max(s)$ and $X(t, :) = s(\beta, :)$
4: Set $C\{t, 1\}$ =cov(s) and $r(t, :)$ =mean(s) $-$ (var$(s) < \alpha$).$*$mean(s)
5: **end for**
6: Set η_{opt} =zeros(n, t_{end})
7: Set $\eta_{\text{opt}}(t)$ the optimal solution of SIBAS algorithm based on the initial portfolio η_{in}
8: **for** $t = 2 : t_{\text{end}}$ **do**
9: Set $\eta_{\text{opt}}(t)$ the optimal solution of SIBAS algorithm based on the previous portfolio $\eta_{\text{opt}}(t-1)$
10: **end for**
11: **return** $\eta_{\text{opt}}(t)$ for $t \in [1, t_{\text{end}}] \subseteq \mathbb{N}$

Output: The optimal solution $\eta_{\text{opt}}(t)$ of the TV-TPNC problem of (4.157)-(4.160).

5

Applications in Real-World Datasets

This chapter includes applications on the portfolio management problems presented in chapter 4 using real-world datasets. More precisely, this chapter is divided into three sections, the first of which contains useful algorithmic procedures for handling data input in continuous-time financial optimization problems, the second of which contains numerical experiments on portfolio management problems using NN solvers, and the third of which contains numerical experiments on portfolio management problems using metaheuristics.

5.1 Handling Data Input in Continuous-Time Portfolio Management Problems

In financial optimization models that we are dealing with, the data input is time-series. A time-series is a series of time-indexed data points that means our data input is discrete. When we deal with a continuous-time financial optimization problem, we need to convert those data inputs from discrete to continuous-time.

In [152, 155, 156], four popular interpolation methods are employed that are also offered by MathWorks, and they demonstrated how to use them with MATLAB routines to produce faster results in the case where input data are given in the form of time-series. These interpolation methods are the step function, the linear, the piecewise cubic Hermite (P.C.Hermite), and the cubic spline (C.Spline). A graphic illustration of these methods is given in Figs. 5.1a, 5.1b, 5.1c and 5.1d, respectively. The data that were used in Fig. 5.1 are the daily close prices of NASDAQ Composite (^IXIC) in the year 2019.

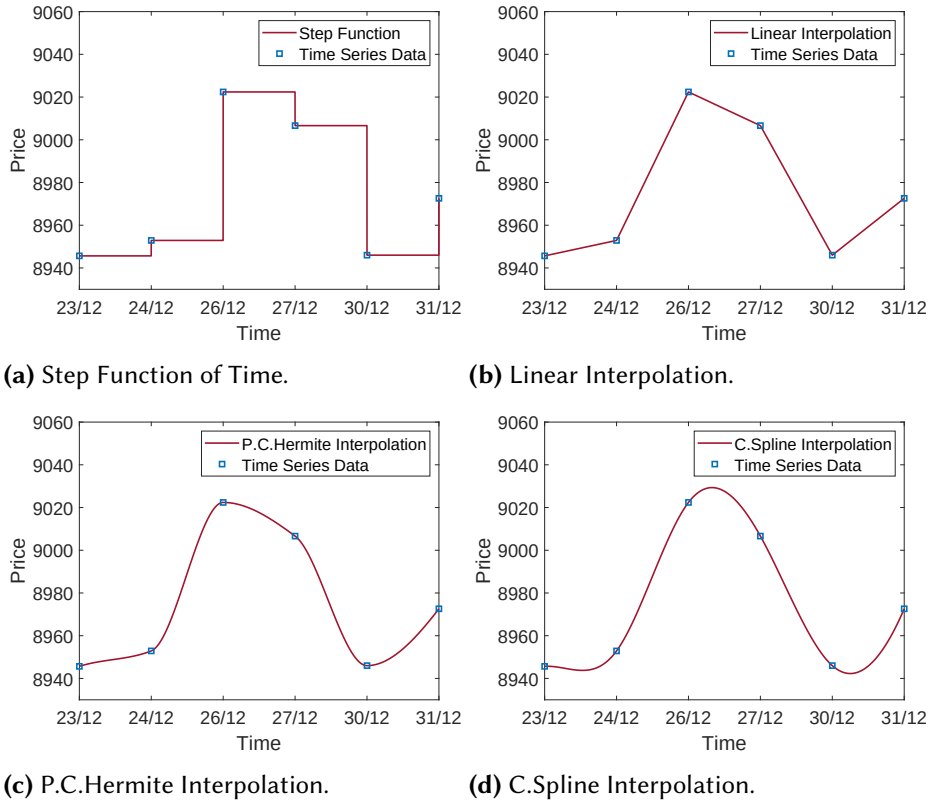
Converting time-series to continuous-time functions is clearly apparent in the following sections. As a result, several algorithms in this chapter make use of the custom MATLAB functions presented in [155, 156]. These custom MATLAB functions are the followings:

- `sfots` and `sfotss` for step function of time of arrays and matrices, respectively;

- `linots` and `linotss` for linear interpolation of arrays and matrices, respectively;
- `pchinots` and `pchinotss` for P.C.Hermite interpolation of arrays and matrices, respectively;
- `splinots` with `sp` and `splinotss` with `sps` for C.Spline interpolation of arrays and matrices, respectively.

The main advantage of these custom MATLAB functions over MathWorks commercial functions is that they can be handled more efficiently by the `ode` MATLAB solvers, lowering computational costs. It is worth mentioning that the algorithmic procedures of all these custom MATLAB functions can also be found in the Appendix A or downloaded from:

https://github.com/SDMourtas/TV-MVPSTC-CC.



(a) Step Function of Time.

(b) Linear Interpolation.

(c) P.C.Hermite Interpolation.

(d) C.Spline Interpolation.

Figure 5.1: Data interpolation methods.

Furthermore, in finance it is possible to split the time periods into daily weekly, monthly, quarterly, annual and their combinations. Yet their results may not be equal in number for two different time periods of the same division, which is due to the fact that financial markets may be close (special days of the calendar), the year may be leap, one month may have fewer days, etc. To solve the problem of missing observations between periods of the same division, we calculate the parameter ω for each t, which divides the observations into the time periods. For this purpose, the following Alg. 18 is proposed in [152, 155, 156]:

Algorithm 18 Algorithm procedure for splitting the observations.

Input: The time t and the vector $noep$, which contains the number of observations in each period.

procedure omega($noep, t$)
  if length($noep$) = 1 **then**
    return $\omega = noep$
  else if If $noep(1) = noep(2) = \cdots = noep(m)$ **then**
    return $\omega = noep/m$
  else
    Set T as the floor price of t and $\omega = 0$
    if $T > 0$ **then**
      for $i = 1 : T$ **do**
        Set $\omega = \omega + noep(i)$
      end for
    end if
    if $t \neq T$ **then**
      Set $\omega = (\omega + noep(T+1) \cdot (t - T))/t$
    else
      Set $\omega = \omega/t$
    end if
    return ω
  end if
end procedure

Output: The parameter ω which splits the observations to the time periods.

5.2 Portfolio Management through NN Solvers

Numerical experiments on the TV-MCPI, TV-MVPS, and TV-BLPO problems are presented in this section, along with a thorough discussion of their implementation.

5.2.1 Numerical Experiments on the TV-MCPI Problem

The TV-MCPI problem is a continuous-time financial optimization problem. Because we are attempting to compute the online solution of a continuous-time problem, the data input must be transformed from discrete-time to continuous-time. As a result, the interpolation functions are employed on $X(t), \dot{X}(t), p(t)$ and $\dot{p}(t)$. It is worth mentioning that $\dot{X}(t) = X(t+1) - X(t)$ and $\dot{p}(t) = p(t+1) - p(t)$. In addition, to solve the omitted observations problem between periods of the same division, we use the parameter ω from Alg. 18, which splits the observations to the time periods for each t inside the ZNN and LVI-PDNN. That is, we employ $X(\omega t), p(\omega t)$ instead of $X(t), p(t)$, and $\dot{X}(\omega t), \dot{p}(\omega t)$ instead of $\dot{X}(t), \dot{p}(t)$. Moreover, the `ode15s` MATLAB solver is employed on (4.9) and (4.10) to generate the online solution of the TV-MCPI problem. Also, in the following experiments,

the parameters settings are $\gamma = 100$, $\beta = 2$ and $\delta = 1000$, while the financial time-series used are taken from `https://finance.yahoo.com` and the exact data used can be downloaded from:

`https://github.com/SDMourtas/DATA/tree/main/TV-MCPI`.

Finally, the ZNN's and LVI-PDNN's solutions are checked, for comparison purposes, against the assumed theoretical solutions produced by the `quadprog` MATLAB function.

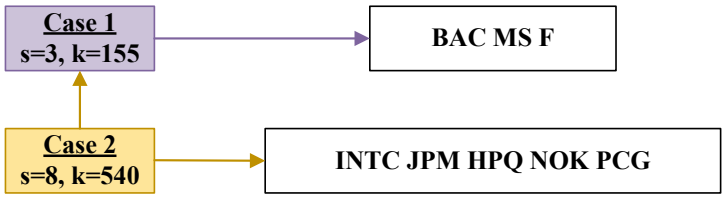
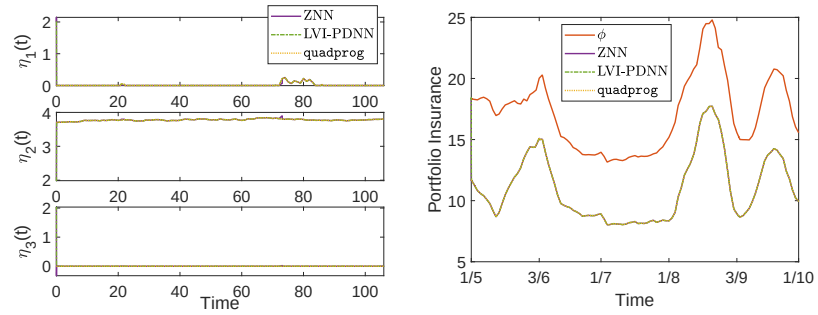


Figure 5.2: The portfolio cases stocks that have been utilized in the TV-MCPI problem experiments.

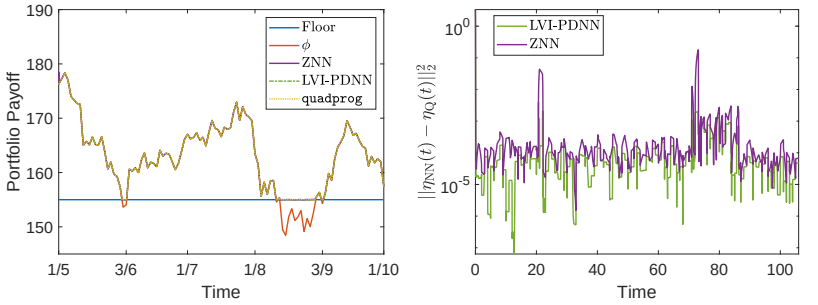
The experiments cover two different portfolio configuration cases. The portfolio cases are presented in Fig. 5.2, with the market space containing some of the most active stocks in the US market. In the j-th case, $j = 1, 2$, we assume the market $X = [x_1, x_2, \ldots, x_s]$, where X includes the daily close prices of the s stocks shown in Fig. 5.2. We employ linear data interpolation in the previously mentioned time series to convert them into functions of time, we set the time delay parameter $c = 21$ and we find the minimum-cost insured portfolio $\eta(t)$ for the time period 01/05/2019 to 01/10/2019. Furthermore, we divide our time-series into five monthly periods by setting $noep = [22, 20, 22, 22, 21]$, and we set $tspan = [0, 5]$ in the `ode15s` MATLAB solver. Note that the vector $noep$ contains the observations number for each month of the time period 01/05/2019 to 01/10/2019. Given a portfolio $\phi = 2\text{ones}(s, 1)$, a floor k, and starting from $y(0) = [\phi; 2\text{ones}(4s+2, 1)]$ in (4.9) and $y(0) = [\phi; 2]$ in (4.10), the results are presented in Figs. 5.3 and 5.4.

More precisely, Fig. 5.3a shows the minimum-cost insured portfolios $\eta(t)$ generated by the ZNN, LVI-PDNN and `quadprog`. We can observe, therein, that the portfolios are identical. Figs. 5.3b and 5.4a present the insurance costs of the initial portfolio ϕ and the portfolios $\eta(t)$, produced by the ZNN, LVI-PDNN and `quadprog`, and Figs. 5.3c and 5.4b present the portfolios $\eta(t)$ payoffs along with the floor price. We can observe that the insurance costs of $\eta(t)$ are rising only in the case where it needs to keep its payoff at the floor, but, by comparing the Figs. 5.3b and 5.4a to Figs. 5.3c and 5.4b, it is evident that the clear payoff of portfolio $\eta(t)$ is greater than the clear payoff of portfolio's ϕ. Also, using the parameter ω, which is especially useful when combining different time periods with unequal numbers of observations in each one, is a novel concept. So, by considering the ω parameter, our approach is more realistic. Another major finding is that, in all the tested cases, the portfolio insurance costs are significantly lower when we ask for a time-varying portfolio $\eta(t)$ instead of a constant portfolio ϕ. Figs. 5.3d and 5.4c depict the error between `quadprog` and NN solvers, produced during the ZNN and LVI-PDNN convergence, while $\eta_{\text{NN}}(t)$ and $\eta_{\text{Q}}(t)$ in the y-label of these figures correspond to the portfolio produced by the NN solvers (i.e., ZNN and LVI-DPNN) and the `quadprog`, respectively. Therein, the noise is expected because we are dealing with time-series and, considering the design parameter's γ low value, the error value is excellent. Note that, as the value of parameter γ increases, the performance of the ZNN and LVI-PDNN models improves and overall



(a) Portfolios convergence.

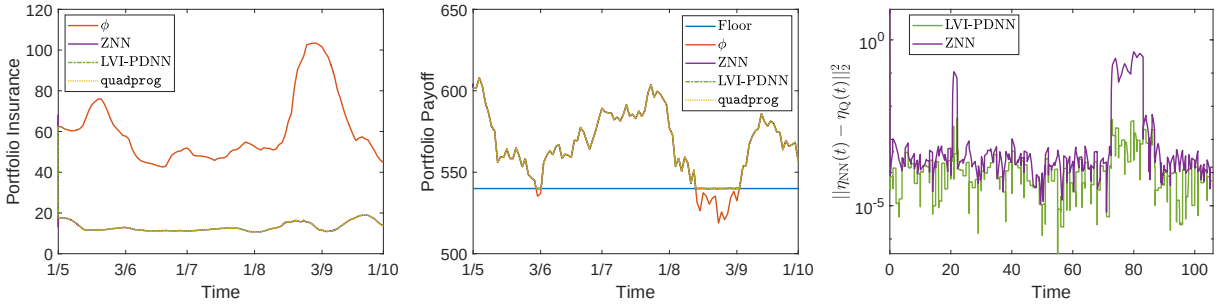
(b) Portfolios insurance costs.



(c) Portfolios payoff.

(d) Error between quadprog and NN solvers.

Figure 5.3: Portfolios convergence, payoff, insurance costs, and the error between quadprog and NN solvers in case 1.



(a) Portfolios insurance costs.

(b) Portfolios payoff.

(c) Error between quadprog and NN solvers.

Figure 5.4: Portfolios payoff, insurance costs, and the error between quadprog and NN solvers in case 2.

error value reduces even more. However, since the error between quadprog and LVI-DPNN is lower than the error between quadprog and ZNN, we can assume that the LVI-DPNN solver performs better than the ZNN. Comprehensively, the ZNN and LVI-PDNN worked splendidly in solving the TV-MCPI problem.

For the purpose of monitoring the performance between the employed custom MATLAB functions (namely `sfots`, `linots`, `pchinots`, `spl`, `splinots`) and the MATLAB functions of MathWorks, `ts2func` and `interp1`, we present Tab. 5.1. This table shows the average execution time of ZNN, LVI-PDNN and quadprog on each portfolio case, by using all the aforementioned MATLAB functions for step function of time, linear, P.C.Hermite and C.Spline interpolation. It is evident that the custom MATLAB functions, which manipulate only time-series, are the best alternatives in terms of computation time responses, as ZNN, LVI-PDNN and quadprog consume less time when used with the custom MATLAB functions than when used with the `interp1`, while producing the

same results. Notably, as the dimension of the portfolio increases, the ZNN and LVI-PDNN time consumption, when used with the function `interp1`, increases even more than the `quadprog` function, while the LVI-PDNN's time consumption, when used with the custom MATLAB functions, is lower than the `quadprog` function. Based on this and the analysis given in numerical experiments of this section, the efficacy and computational efficiency of the ZNN and LVI-PDNN solvers are proven.

Table 5.1: The ZNN, LVI-PDNN and quadprog time consumption for solving the TV-MCPI problem.

Interpolation Function	Case 1			Case 2		
	3 Stocks Portfolio			8 Stocks Portfolio		
	ZNN	LVI-PDNN	`quadprog`	ZNN	LVI-PDNN	`quadprog`
`sfots`	4 s	1.5 s	6.7 s	9 s	2 s	7 s
`ts2func`	4.5 s	2.6 s	6.7 s	9.5 s	3 s	7.2 s
`linots`	3.2 s	2.8 s	5 s	32 s	8 s	8 s
`interp1` (linear)	1.4 s	5 s	1.7 s	75 s	27 s	13 s
`pchinots`	2.1 s	4.4 s	2.4 s	128 s	24 s	28 s
`interp1` (P.C.Hermite)	2.2 s	12 s	2.1 s	320 s	67 s	32 s
`sp` & `splinots`	0.4 s	4 s	1 s	43 s	14 s	10 s
`interp1` (C.Spline)	0.8 s	12 s	1.2 s	151 s	75 s	20 s

5.2.2 Numerical Experiments on the TV-MVPS Problem

In the TV-MVPS problem, we use the expected return array and the covariance matrix of a portfolio, which comprises of time-series. That is, our data input is in discrete-time. As a result, the data input must be transformed from discrete-time to continuous-time. We accomplish this by employing interpolation functions. The following Alg. 19 shows how we construct the expected return array r and its first derivative $\dot{r}$, as well as the covariance matrix C and its first derivative $\dot{C}$. Furthermore, Alg. 19 makes use of the custom MATLAB functions `linots` and `linotss` for linear interpolation of arrays and matrices, respectively.

Note that we normalize the portfolio's data for each time period in order to have a correct covariance matrix for comparison purposes, while, without loss of generality, we multiply the covariance matrix C with the number 100, which causes the variance of the portfolio to be in %. Moreover, the `ode15s` MATLAB solver is employed on (4.31), (4.37) and (4.38) to generate the online solution of the TV-MVPS problem. Also, in the following experiments, the design parameter has been set to $\gamma = 100$, and the ZNN solver's penalty function settings have been set to $s = 3e4$ and $1e-8$. The financial time-series used are taken from `https://finance.yahoo.com` and the exact data used can be downloaded from:

`https://github.com/SDMourtas/DATA/tree/main/TV-MVPS`.

Finally, the ZNN's and LVI-PDNN's solutions are checked, for comparison purposes, against the assumed theoretical solutions produced by the `quadprog` MATLAB function.

The experiments cover two different portfolio configuration cases. The portfolio cases are presented in Fig. 5.5, with the market space containing some of the most active stocks in the US market. In the j-th case, $j = 1,2$, we assume the market $X = [x_1, x_2, \ldots, x_s]$, where X includes the daily close prices of the s stocks shown in Fig. 5.5

Algorithm 19 Algorithmic procedure for the TV-MVPS problem's data preparation.

Input: The marketed space $X = [x_1, x_2, \ldots, x_n]$ which is a matrix of n time series as column vectors of m prices, the moving average's number of time periods $\tau \leq m-1$, $\tau \in \mathbb{N}$.

procedure Data_Prep(X, τ)
  Set $[m, n]$ =size(X)
  Set r =zeros($m-\tau, n$)
  Set $C\{m-\tau, 1\} = \{\}$
  for $i = 1 : m-\tau$ **do**
    Set h =max($X(i:\tau+i-1,:)$)
    Set $C\{i,1\} = 100 * \text{cov}(X(i:\tau+i-1,:)./h)$
    Set $r(i,:)$ =mean($X(i:\tau+i-1,:)./h$)
  end for
  for $j = 1$:length(r) -1 **do**
    Set $\dot{C}\{j,1\} = C\{j+1,1\} - C\{j,1\}$
    Set $\dot{r}(j,:) = r(j+1,:) - r(j,:)$
  end for
  Set $C = @(t)\text{linotss}(C, t)$ and $r = @(t)\text{linots}(r, t)'$
  Set $\dot{C} = @(t)\text{linotss}(\dot{C}, t)$ and $\dot{r} = @(t)\text{linots}(\dot{r}, t)'$
end procedure

Output: The continuous-time function C that comprises of the covariance matrix of the normalized portfolio of each time t, and its first derivative $\dot{C}$, the continuous-time function r that comprises of the expected return of the normalized portfolio of each time t, and its first derivative $\dot{r}$.

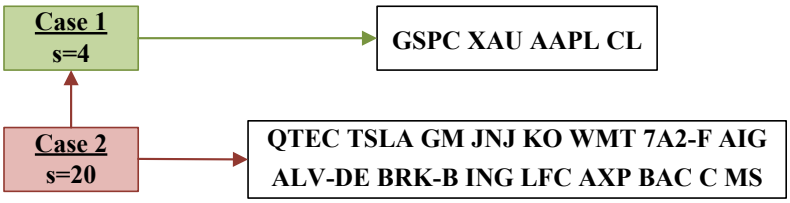
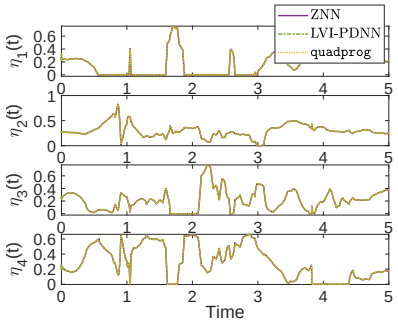


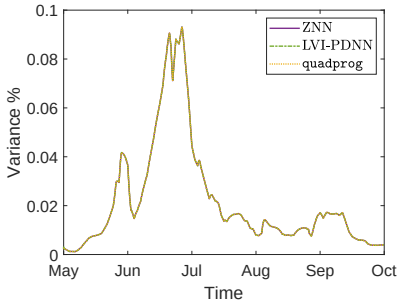
Figure 5.5: The portfolio cases stocks that have been utilized in the TV-MVPS problem experiments.

from 2/4/2019 to 1/10/2019 into $x_1, x_2, \ldots, x_s$, respectively. We employ linear data interpolation in the previously mentioned time series to convert them into functions of time, we set the time delay parameter $\tau = 20$ to calculate the expected returns r and covariance C of Alg. 19. The rest of our data is the period from 1/5/2019 to 1/10/2019 with 107 observations. In particular, May, July, August have 22 observations each, June has 20 observations, September and October have 21 observations together. To solve the omitted observations problem between periods of the same division, we use the parameter ω from Alg. 18, which splits the observations to the time periods for each t inside the ZNN and LVI-PDNN. That is, we employ $r(\omega t), C(\omega t)$ instead of $r(t), C(t)$, and $\dot{r}(\omega t), \dot{C}(\omega t)$ instead of $\dot{r}(t), \dot{C}(t)$. So, we divide our time-series into five monthly periods by setting $noep = [22, 20, 22, 22, 21]$, we use the linear data interpolation, and we set $tspan = [0, 5]$ in the `ode15s` MATLAB solver to find the optimal portfolio $\eta(t)$ for the time period 01/05/2019 to 01/10/2019. For each portfolio case, we examine two selections of portfolios. In the first selection, we set $r_p = \max(0.94 + 0.004t, \text{mean}(r(\omega(t))))$, and use the SERT

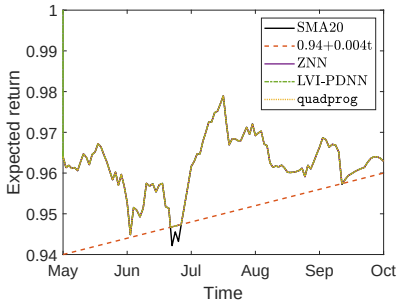
setup of subsection 4.2.3. In the second selection, we set $r_p = \min(r(\omega(t)))$, and use the APERT setup of subsection 4.2.3. Starting from $y(0) = \text{ones}(s+2,1)/s$ in (4.31) and (4.38), and $y(0) = \text{ones}(s+1,1)/s$ in (4.37), the results are presented in Figs. 5.6 and 5.7.



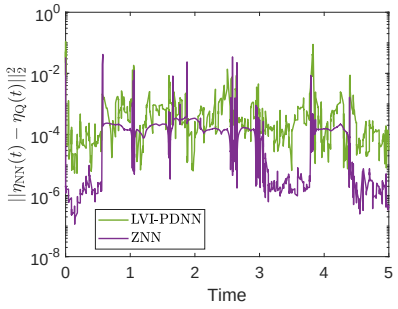
(a) Portfolios convergence under SERT setup.



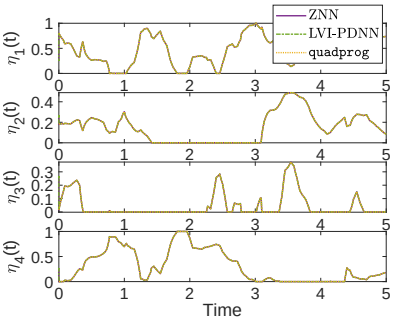
(b) Portfolios variance % under SERT setup.



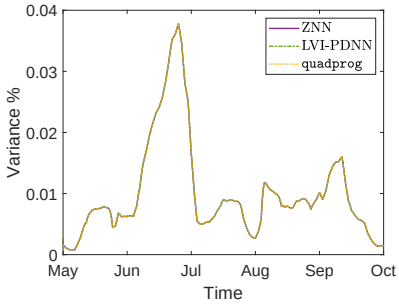
(c) Portfolios expected return under SERT setup.



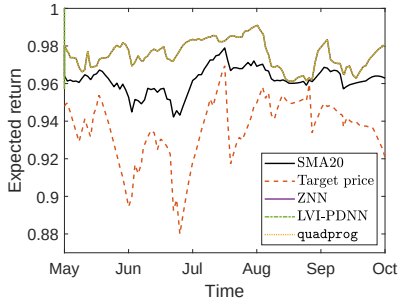
(d) Error between quadprog and NN solvers under SERT setup.



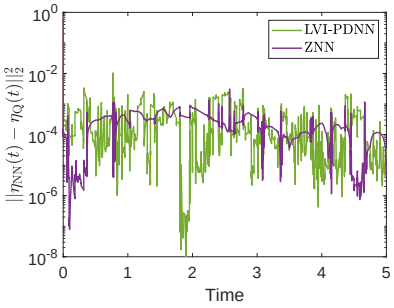
(e) Portfolios convergence under APERT setup.



(f) Portfolios variance % under APERT setup.



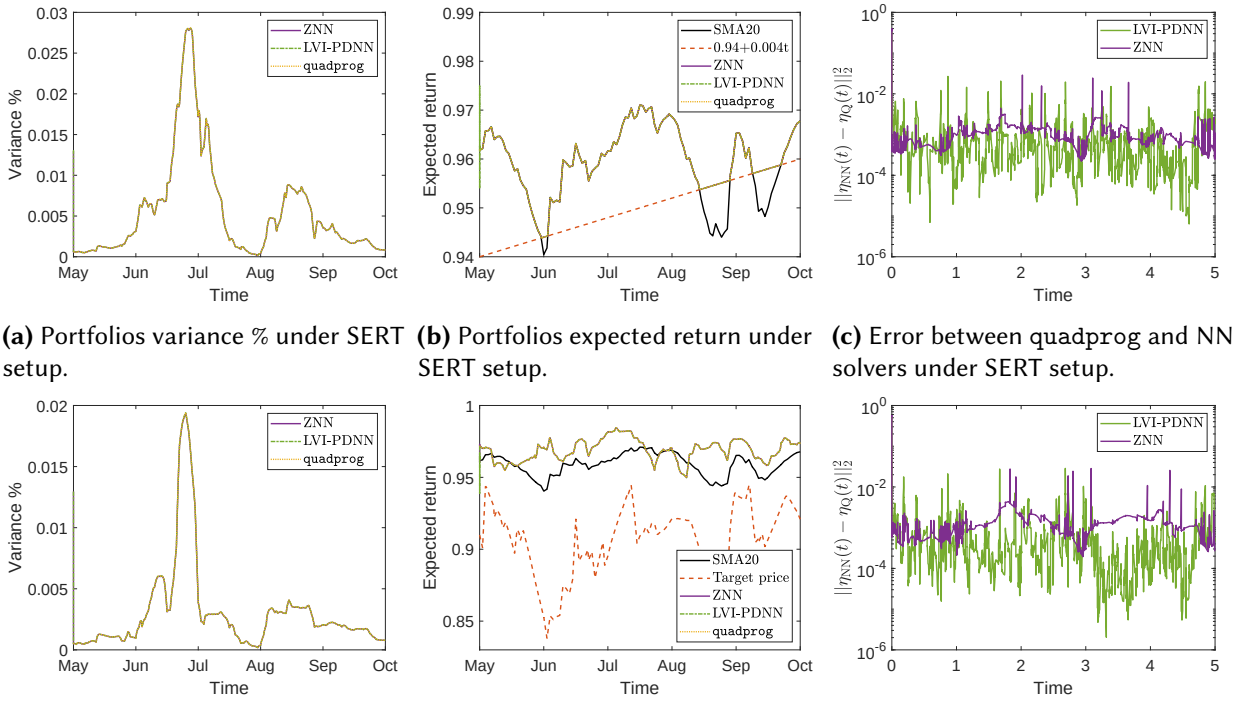
(g) Portfolios expected return under APERT setup.



(h) Error between quadprog and NN solvers under APERT setup.

Figure 5.6: Portfolios convergence, variance %, expected return, and the error between quadprog and NN solvers in case 1.

More precisely, Figs. 5.6a and 5.6e show the optimal mean-variance portfolios $\eta(t)$ generated by the ZNN, LVI-PDNN and quadprog. We can observe, therein, that the portfolios are identical. The variance % of the portfolios $\eta(t)$ for a specific target expected return are shown in Figs. 5.6b and 5.7a and the variance % of the portfolios $\eta(t)$ for all expected returns are shown in Figs. 5.6f and 5.7d. The expected return of the portfolios $\eta(t)$, which is $\eta(t)r(\omega t)$, compared with the outcome of quadprog and the simple moving average $SMA20$ of $X(t)$, which is $\text{mean}(r(\omega t))$, for a specific target expected return are shown in Figs. 5.6c and 5.7b and the expected return of the portfolios $\eta(t)$ for all expected returns are shown in Figs. 5.6g and 5.7e. Also, the Figs. 5.6c and 5.7b show the function $0.94 + 0.004t$. Comparing the Figs. 5.6b, 5.7a, 5.6f and 5.7d to Figs. 5.6c, 5.7b, 5.6g and 5.7e, respectively,



(a) Portfolios variance % under SERT setup.

(b) Portfolios expected return under SERT setup.

(c) Error between quadprog and NN solvers under SERT setup.

(d) Portfolios variance % under APERT setup.

(e) Portfolios expected return under APERT setup.

(f) Error between quadprog and NN solvers under APERT setup.

Figure 5.7: Portfolios variance %, expected return, and the error between quadprog and NN solvers in case 2.

we can observe that the variance of $\eta(t)$ is rising only in the case where it needs to keep its expected return at r_p. Furthermore, considering the ω parameter, which is very helpful in the case where we want to combine different time periods with a different number of observations in each one of them, our approach is more realistic. Another major finding is that, in all the tested cases, the variance of the portfolios for a specific target expected return are significantly higher than the variance of the portfolios for all expected returns. Figs. 5.6d, 5.7c, 5.6h and 5.7f depict the error between quadprog and NN solvers, produced during the ZNN and LVI-PDNN convergence. It is worth mentioning that $\eta_{NN}(t)$ and $\eta_Q(t)$ in these figures correspond to the portfolio produced by the NN solvers (i.e., ZNN and LVI-DPNN) and the quadprog, respectively. Therein, the noise is expected because we are dealing with time-series and, considering the design parameter's γ low value, the error value is excellent. Note that, as the value of parameter γ increases, the performance of the ZNN and LVI-PDNN models improves and overall error value reduces even more. However, since the error between quadprog and LVI-DPNN is lower than the error between quadprog and ZNN, we can assume that the LVI-DPNN solver performs better than the ZNN. Overall, the portfolio cases presented in numerical experiments of this section show that the ZNN and LVI-PDNN worked excellently in solving TV-MVPS problem.

For the purpose of monitoring the performance between the employed custom MATLAB functions (namely `sfots` & `sfotss`, `linots` & `linotss`, `pchinots` & `pchinotss`, `splinots` & `splinotss`) and the MATLAB functions of MathWorks, `ts2func` and `interp1`, we present Tab. 5.2. This table shows the average execution time of ZNN, LVI-PDNN and quadprog on each portfolio case, by using all the aforementioned MATLAB functions for step function of time, linear, P.C.Hermite and C.Spline interpolation. The general conclusion arising from Tab. 5.2 is that the step function of time series is the least efficient method and that the linear interpolation is the most

efficient. In addition, from Tab. 5.2, we conclude that the custom MATLAB functions, which manipulate matrices and structures time-series, are the best alternatives in terms of computation time responses, while they produce the same results. The efficacy and computational efficiency of the ZNN and LVI-PDNN solvers are proven based on this and the analyses presented in this section's numerical experiments.

Table 5.2: The ZNN, LVI-PDNN and quadprog time consumption for solving the TV-MVPS problem.

Interpolation Function	Case 1: 4 Stocks Portfolio					
	SERT Setup			APERT Setup		
	ZNN	LVI-PDNN	quadprog	ZNN	LVI-PDNN	quadprog
sfots & sfotss	1.7 s	2.4 s	4.6 s	1.8 s	1.5 s	4.1 s
ts2func	2.5 s	3 s	4.6 s	2.1 s	2.4 s	4.4 s
linots & linotss	0.5 s	0.5 s	1.4 s	0.6 s	0.6 s	1.4 s
interp1 (linear)	3.6 s	1.8 s	2 s	2.1 s	1.6 s	1.5 s
pchinots & pchinotss	1.3 s	0.6 s	1.6 s	1 s	0.5 s	1.3 s
interp1 (P.C.Hermite)	7 s	3.3 s	2.5 s	4.9 s	3.3 s	2.1 s
splinots & splinotss	0.8 s	0.5 s	1.5 s	0.4 s	0.4 s	1.1 s
interp1 (C.Spline)	5.2 s	2.4 s	2.7 s	2.8 s	2.4 s	1.9 s
	Case 2: 20 Stocks Portfolio					
	SERT Setup			APERT Setup		
	ZNN	LVI-PDNN	quadprog	ZNN	LVI-PDNN	quadprog
sfots & sfotss	8 s	4.7 s	9 s	6 s	3.2 s	7.7 s
linots & linotss	4.9 s	3 s	7 s	2.8 s	2.2 s	4.3 s
pchinots & pchinotss	10 s	5.8 s	8 s	5.4 s	3.3 s	4.8 s
splinots & splinotss	6 s	3.6 s	7.4 s	2.7 s	2 s	4.3 s

5.2.3 Numerical Experiments on the TV-BLPO Problem

As a financial optimization model, the data input of the TV-BLPO problem are time-series. That is, our data input is in discrete-time. As a result, some data preparation and processing should be done before the NN solvers address the TV-BLPO problem. Since investor's views in the TV-BLPO problem are regarded as a forecasting problem, the MAWTSNN model of subsection 3.4.1 is used to create them. Following the MAWTSNN model's creation of the investor's views matrix z, the market space M and the investor's views z are normalized. Then, the discrete-time investor's views vector $Z(t)$, the discrete-time equilibrium excess returns $p_{bl}(t)$ and the discrete-time covariance matrix of the equilibrium returns $C_{bl}(t)$ should be constructed based on the analysis presented in subsection 4.3.1. Without sacrificing generality, the covariance matrix $C_{bl}(t)$ is multiplied by the number 100, resulting in the portfolio's variance being expressed in percent. The first derivatives of the discrete-time equilibrium excess returns and the discrete-time covariance matrix of the equilibrium returns, i.e. $\dot{p}_{bl}(t)$ and $\dot{C}_{bl}(t)$, are then constructed. In Alg. 20, the aforementioned procedure is presented through MATLAB routines in further detail.

It is worth noting that because we are attempting to compute the online solution of a continuous time-varying problem, the data input must be transformed from discrete-time to continuous-time. Converting time-series to continuous-time functions is clearly apparent in this paper. As a result, Alg. 20 makes use of the custom MATLAB

functions `linots` and `linotss` for linear interpolation of arrays and matrices, respectively. More precisely, `linots` is employed to convert the discrete-time arrays $p_{bl}(t)$, $\dot{p}_{bl}(t)$, $p_g(t)$ and $\dot{p}_g(t)$ into interpolated continuous-time functions, and `linotss` is employed to convert the discrete-time matrices $C_{bl}(t)$ and $\dot{C}_{bl}(t)$ into interpolated continuous-time functions. However, several other custom interpolation functions for popular interpolation methods are proposed in [155, 156], where their main advantage over MathWorks commercial functions is that they can be handled more efficiently by the `ode` MATLAB solvers, lowering computational costs. That is, the ZNN and LVI-PDNN generate faster results in the case where time-series are input data.

Algorithm 20 Algorithm for the TV-BLPO problem's data preparation.

Input: The market space $M \in \mathbb{R}^{m\times n}$, which comprises of m prices of n time-series, the number of delays $\theta \leq m-1$, $\theta \in \mathbb{N}$, the matrix $z \in \mathbb{R}^{(m-\theta)\times k}$, that includes the investor's views on $k \leq n$ assets, and the parameter τ.

procedure Data_Prep(M, z, k, τ)
  Set $a = M(\theta+1:\text{end},:)$, $b = a(:,1:k)$ and $c = (z-b)./b$
  Set p =zeros$(m-\theta, n)$ and $C\{m-\theta, 1\} = \{\}$
  Set P =zeros(k, n) and $P(1:(k+1):\text{end}) = 1$
  for $j = 1 : m-\theta$ **do**
    Set p =max$(M(j:\theta+j-1,:))$, $p = p./\max(p)$ and C =cov(p)
    Set u =quadprog$(2C, [\,], [\,], [\,]$,ones$(1,n)$,1,zeros$(1,n)$,ones$(1,n))$
    Set ζ =mean$(p,'all')/($std(mean$(p,2))$sqrt$(u'Cu))$
    Set $r = \zeta Cu$, $Q = \tau C$ and $Z = c(j,:)$
    Set Υ =diag(diag$((1-\tau)PCP')$)
    Set $C_{bl}\{j,1\} = 100($pinv$(P'(\Upsilon \backslash P)+$pinv$(Q)))$
    Set $p_{bl}(j,:)$ =pinv$(P'(\Upsilon \backslash P)+$pinv$(Q))($pinv$(Q)r + P'(\Upsilon \backslash Z'))$
  end for
  for $j = 1$:length$(p_{bl}) - 1$ **do**
    Set $\dot{C}_{bl}\{j,1\} = C_{bl}\{j+1,1\} - C_{bl}\{j,1\}$
    Set $\dot{p}_{bl}(j,:) = p_{bl}(j+1,:) - p_{bl}(j,:)$
  end for
  Set $C_{bl} = @(t)$linotss(C_{bl}, t) and $p_{bl} = @(t)$linots$(p_{bl}, t)'$
  Set $\dot{C}_{bl} = @(t)$linotss$(\dot{C}_{bl}, t)$ and $\dot{p}_{bl} = @(t)$linots$(\dot{p}_{bl}, t)'$
end procedure

Output: The continuous-time function C_{bl}, that contain the covariance matrix of the equilibrium returns of each time t, and its first derivative $\dot{C}_{bl}$, the continuous-time function p_{bl}, that contains the equilibrium excess returns of each t, and its first derivative $\dot{p}_{bl}$.

In the experiments of this subsection, the financial time-series used have been retrieved from https://finance.yahoo.com. The `ode15s` MATLAB solver has been used on (4.62) and (4.63) to produce the online solution of the TV-BLPO problem under the parameter $\gamma = 1e+6$. The parameters in the ZNN solver of (4.62) have been set to $p = 1e-8$ and $s = 3e+4$. Finally, the solutions produced by the ZNN and LVI-PDNN solvers are compared to the supposed theoretical solutions generated by the `quadprog` MATLAB function. It is worth mentioning that the `quadprog` MATLAB function can only solve static QP problems and not continuous TVQP problems. As a result, multiple QP problems have been adjusted and then solved by the `quadprog` to reproduce a part of the the solutions produced by the ZNN and LVI-PDNN solvers on the continuous TVQP problem.

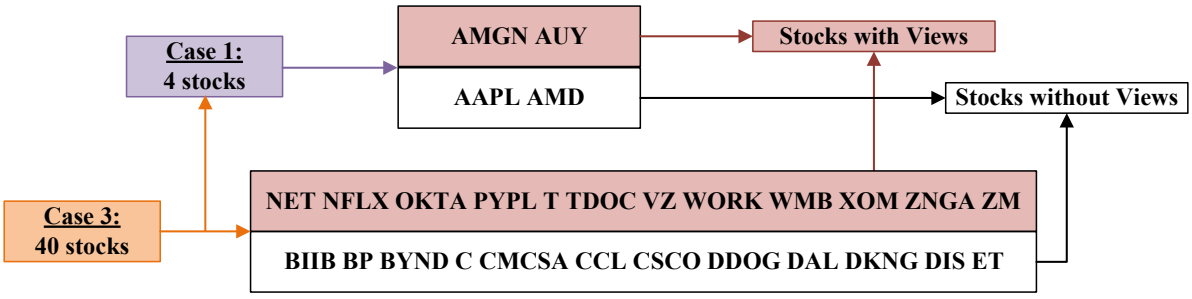
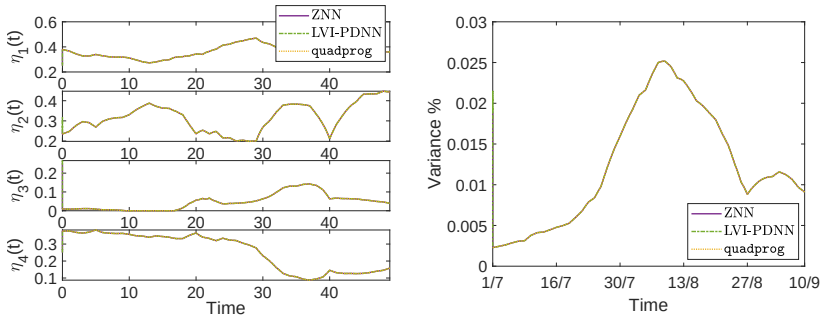


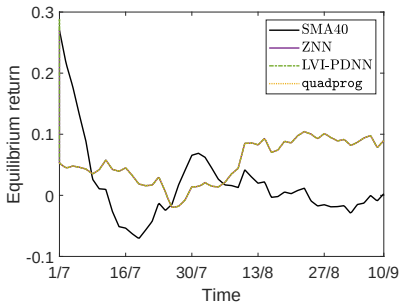
Figure 5.8: The portfolio cases stocks that have been utilized in the TV-BLPO problem experiments.

The experiments cover two different portfolio configuration cases. The portfolio cases are presented in Fig. 5.8, with the market space containing some of the most active stocks in the US market. In the j-th case, $j = 1,2$, we assume the market space $M = [\mu_1, \mu_2, \ldots, \mu_s]$, where M includes the daily close prices of the s stocks shown in Fig. 5.8, from 5/5/2020 to 10/9/2020. That is, $M \in \mathbb{R}^{90\times s}$. Furthermore, the matrix z in Alg. 20 contains the forecasting prices produced by the MAWTSNN for the time-period 1/7/2020 to 10/9/2020 under the specifications used in [69]. Thus, setting the delays number $\theta = 40$, the matrix $z = [z_1, z_2, \ldots, z_{s/2}] \in \mathbb{R}^{50\times(s/2)}$ contains the investor's views in the TV-BLPO problem. Employing Alg. 20 and setting the parameter $\tau = 0.9$, we construct the continuous-time functions $C_{bl}(t)$, $\dot{C}_{bl}(t)$, $p_{bl}(t)$ and $\dot{p}_{bl}(t)$. Then, given an initial portfolio $\eta(0)$ =ones$(s,1)/s$, we find the TV-BLPO solution for the time-period 1/7/2020 to 10/9/2020, with $t_{\text{span}} = [0, 49]$ and target price $p_g(t)$ =min$(p_{bl}(t))$. The results are presented in Figs. 5.9-5.10.

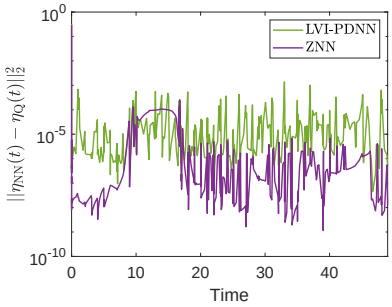


(a) Portfolios weights.

(b) Portfolios risk.

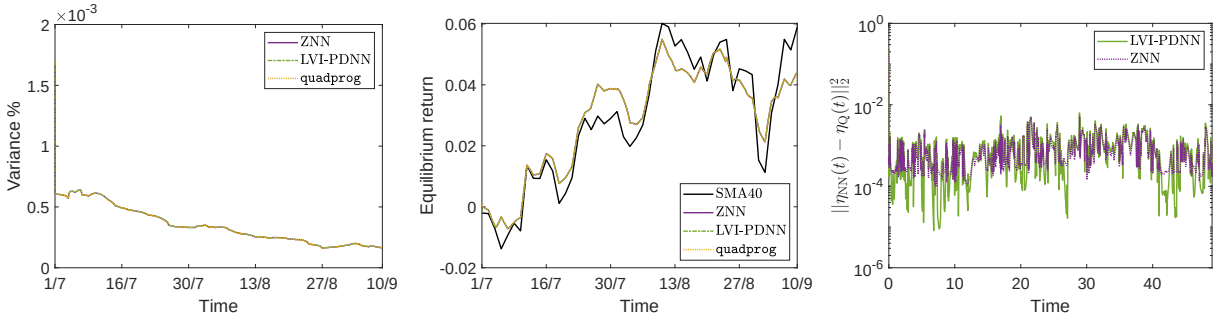


(c) Portfolios equilibrium returns.



(d) Error between quadprog and NN solvers.

Figure 5.9: Portfolios weights, risk and equilibrium returns, and the error between quadprog and NN solvers in portfolio case 1.



(a) Portfolios risk. **(b)** Portfolios equilibrium returns. **(c)** Error between quadprog and NN solvers.

Figure 5.10: Portfolios risk and equilibrium returns, and the error between quadprog and NN solvers in portfolio case 2.

Portfolio case 1 involves a 4-stock portfolio with 2 stocks containing investor views, whereas portfolio case 2 involves a 40-stock portfolio with 20 stocks including investor views. The portfolio's variance % is depicted in Figs. 5.9b and 5.10a for cases 1 and 2, respectively, and the equilibrium return is depicted in Figs. 5.9c and 5.10b. Therein, we observe that the outcomes produced by the ZNN, LVI-DPNN and quadprog are identical. This can be verified from the error between the quadprog and the and NN solvers in Figs. 5.9d and 5.10c. It is worth mentioning that $\eta_{NN}(t)$ and $\eta_Q(t)$ in the y-label of Figs. 5.9d and 5.10c correspond to the portfolio produced by the NN solvers (i.e., ZNN and LVI-DPNN) and the quadprog, respectively. On the one hand, since the error between quadprog and ZNN is lower than the error between quadprog and LVI-DPNN, we can assume that the ZNN solver performs better than the LVI-DPNN in the portfolio case 1. On the other hand, since the error between quadprog and LVI-DPNN is lower than the error between quadprog and ZNN, we can assume that the LVI-DPNN solver performs better than the ZNN in the portfolio cases 2.

Overall, in line with the Figs. 5.9-5.10 depicted results, the NN solvers performed excellent in solving the TV-BLPO problem under two different portfolio configuration cases. Furthermore, in line with the Figs. 5.9d and 5.10c depicted results, the ZNN solver performs better than the LVI-DPNN when the portfolio has small dimensions, while the LVI-DPNN solver performs better than the ZNN when the portfolio has large dimensions. It is worth mentioning that, as the value of the parameter γ rises, the performance of the ZNN and LVI-PDNN solvers improves and with greater precision approaches the expected theoretical solution. In addition, the whole development and implementation of the TV-BLPO problem on this subsection may be found on GitHub at the next link:

https://github.com/SDMourtas/CTBLPO.

5.3 Portfolio Management through Metaheuristics

Numerical experiments on the MPMCPITC, TV-MCPITC, TV-MVPSTC-CC, BMPS, and TV-TPNC problems are presented in this section, along with a thorough discussion of their implementation.

5.3.1 Numerical Experiments on the MPMCPITC Problem

In the experiments of this subsection, the financial time-series used have been retrieved from https://finance.yahoo.com. Note that for all the experiments, the results have been produced from BAS, BA, FA

and GA, where we have set

- $\delta_i^+ = \delta_i^- = 0.1$, $\zeta_i^+ = 0.06$ and $\zeta_i^- = 0.04$ in (4.76);
- $R = 1e5$ in Alg. 3;
- $d_0 = 2$, $\delta_0 = 2$ and $K_{max} = 1200$ in BAS;
- population size 100, max iterations 1000, parameter alpha 0.97, parameter gamma 0.1, initial loudness 1 and initial pulse rate 1 in BA;
- population size 100, randomness reduction factor 0.97, absorption coefficient 0.01; attractiveness constant 1, randomness strength 1 and maximum number of iterations 1000 in FA,
- GA MATLAB function in its default settings.

The experiments cover two different portfolio configuration cases. The portfolio cases are presented in Fig. 5.11, with the market space containing some of the most active stocks in the US market. In the j-th case, $j = 1,2$, we assume the market space $X = [x_1, x_2, \ldots, x_s]$, where X includes the daily close prices of the s stocks shown in Fig. 5.8. On the one hand, the portfolio case 1 refers to the period 7/5/2020 - 31/7/2020, which has 60 observations, i.e. $X \in \mathbb{R}^{60\times 8}$, while by setting $\beta = 10$, we have a 7 period model and, hence, $X(t) \in \mathbb{R}^{10\times 8}$ with $t \in [1,6]$. Moreover, given a portfolio $\theta = [7,2,1,1,2,4,0,1]^{\mathrm{T}}$, we set the floor $\phi = [530,570,550,570,580,600]^{\mathrm{T}}$ and the parameters $\xi = 10$, $\lambda = 5$ in (4.88). On the other hand, the portfolio case 2 refers to the period 10/3/2020 - 30/7/2020, which has 100 observations, i.e. $X \in \mathbb{R}^{100\times 16}$, while by setting $\beta = 10$, we have a 11 period model and, hence, $X(t) \in \mathbb{R}^{10\times 16}$ with $t \in [1,10]$. In addition, given a portfolio $\theta = \text{ones}(1,16)$, we set the floor $\phi = [840 : 40 : 1200]^{\mathrm{T}}$ and the parameters $\xi = 10$, $\lambda = 5$ in (4.88). The results are presented in Fig. 5.12.

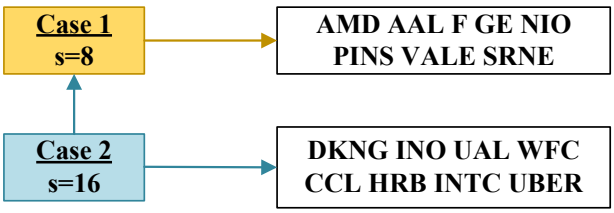
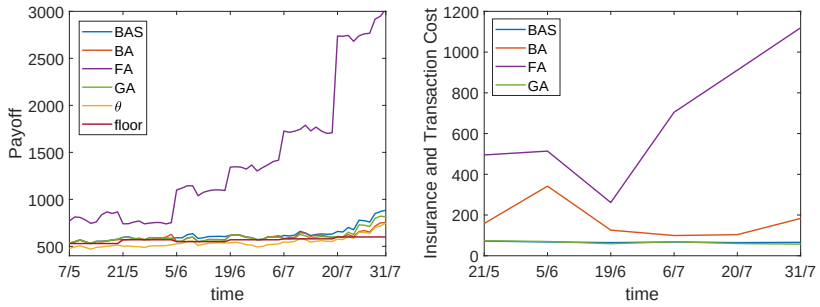
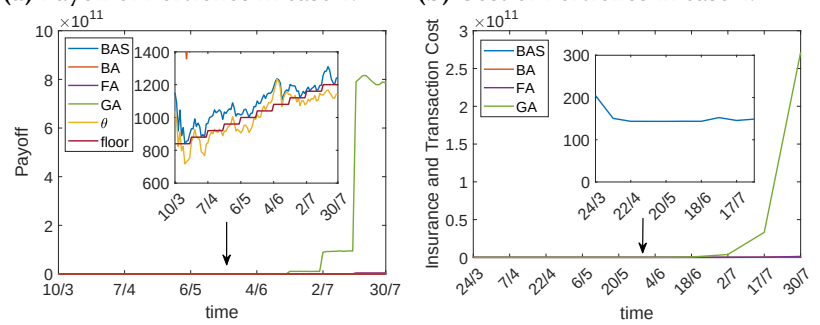


Figure 5.11: The portfolio cases stocks that have been utilized in the MPMCPITC problem experiments.

The MPMCPI problem minimizes (4.77) and, at the same time, keeps the portfolio's payoff above a floor price. Based on that, in Figs. 5.12a and 5.12c, we observe that the outcomes of BAS, BA, FA and GA keep the portfolio's payoff, which is the outcome of $G_1(\eta(t))$, above the floor price in both portfolio cases. Furthermore, Figs. 5.12b and 5.12d show the value of (4.77) in portfolio cases 1 and 2, respectively. More precisely, for the portfolio case 1, we note in Figs. 5.12a and 5.12b that the outcomes of BAS and GA are reasonably similar and are also better than the outcome of BA. Moreover, the outcome of FA is the worst. For the portfolio case 2, we observe in Figs. 5.12c and 5.12d that the outcome of BAS is much better than the outcomes of BA, FA and GA. Note that when the portfolios costs are high in Figs. 5.12b and 5.12d, the portfolios payoff in Figs. 5.12a and 5.12c, are high too.



(a) Payoff of Portfolios in case 1. **(b)** Cost of Portfolios in case 1.



(c) Payoff of Portfolios in case 2. **(d)** Cost of Portfolios in case 2.

Figure 5.12: Portfolios payoff and costs in cases 1 and 2.

However, given that minimizing the portfolios costs is the main task of the MPMCPI problem, the higher the costs in Figs. 5.12b and 5.12d, the worse the method 's solution is.

The time usage of BAS, BA, FA and GA, in portfolio cases 1 and 2, is presented in Tab. 5.3, which includes their average execution time. Therein, we note that BAS is always close to 20 times faster than BA, when BA is always close to 45 times faster than FA. Furthermore, the time consumptions of FA and GA are always close. Overall, BAS produces the same or better results than BA, FA and GA in both portfolio cases and, at the same time, BAS is far less time-consuming. Based on that, we can conclude that in the MPMCPI problem, BAS is more efficient than BA, FA and GA.

Table 5.3: The BAS, BA, FA and GA time consumption for solving the MPMCPITC problem.

Portfolio	BAS	BA	FA	GA
Case 1	0.2 s	3.5 s	160 s	150 s
Case 2	0.2 s	6 s	276 s	232 s

5.3.2 Numerical Experiments on the TV-MCPITC Problem

The data inputs in the TV-MCPITC optimization model are time-series. A time-series is a sequence of time-indexed data points, which means that the data input in the TV-MCPITC is initially discrete. Because we are trying to find the online solution to a time-varying optimization problem, we must convert those data inputs from discrete to corresponding continuous-time form. We achieve that successfully by transforming arrays and matrices of time-series to continuous-time functions. That is, the interpolation functions are employed on $X(t)$ and $p(t)$. The procedure for construction of the insurance prices vector is presented in the following Alg. 21.

Algorithm 21 Algorithm for the data preparation of the portfolio's insurance prices.

Input: The marketed space $X = [x_1, x_2, \ldots, x_n]$ which is a matrix of n time series as column vectors of m prices; the number of time periods $\tau \leq m-1$ $\tau \in \mathbb{N}$; the cost rates θ associated with the risk of assets, and the fixed costs ζ of Eq. (4.93).

1: Set $[m, n]$ =size(X) and p =zeros($m-\tau, n$)
2: **for** $i = 1 : m-\tau$ **do**
3: Set $Y = X(i : \tau + i - 1, :)$ and $p(i, :) = \zeta + \theta \text{var}(Y / \max(Y))$
4: **end for**

Output: The matrix p composed of the insurance prices for a number of time periods of each portfolio's time-series.

The following are the methods used to compare TVBAS performance. The MATLAB function `fmincon` is an NLP solver which finds the minimum of a nonlinear function under nonlinear equality and inequality constraints. This function is a gradient-based method designed to operate on problems where both objective and constraint functions are continuous as well as their first derivatives. The MATLAB function GA finds the local minimum of a function under linear equality and inequality constraints by using genetic algorithm (GA). GA is a meta-heuristic algorithm inspired by the process of natural selection. The MATLAB function PSO (i.e. `particleswarm`) is a derivative-free global optimum solver which finds the local minimum of a function. PSO is inspired by the surprisingly organized behaviour of large groups of simple animals, such as flocks of birds, schools of fish, or swarms of locusts. The MATLAB functions GA, PSO and TVBAS are perfect candidates for comparison because all of them include nature-inspired meta-heuristic optimization algorithms. We also compare obtained results with the MATLAB function `fmincon` because it is assumed to find the optimum solution to an NLP problem that the functions GA, PSO and TVBAS must match.

To solve the TV-MCPITC problem, we first present a complementary algorithm which constructs the variables of the TV-NLP problem described in subsection 4.5.3 combined with Alg. 18.

Algorithm 22 Algorithm for the TV-NLP problem of subsection 4.5.3.

Input: The interpolated marketed space and insurance prices f_X and f_p, respectively; the time t and the vector $noep$, which contains the number of observations in each period; the floor ϕ, the previous portfolio $\eta_{-1} = \eta(t-1)$, and the previous insurance and transaction costs y_{-1}.

1: **function** $[\eta^-, \eta^+, A, b, p]$ = problem($noep, t, f_p, f_X, \phi, \eta_{-1}, y_{-1}$)
2: Set ω =omega($noep, t$), floor = $\phi(\omega t)$, $p = f_p(\omega t)$ and $A = -f_X(\omega t)$
3: Set payoff = $A\eta_{-1} + y_{-1}$, b = min(payoff, −floor), η^- =zeros(length(p), 1) and η^+ = payoff./A'
4: **end function**

Output: The η^-, η^+, A, b, p for the time t.

The main algorithm for solving the TV-MCPITC problem is the following algorithm 23 which also includes the TVBAS MATLAB function.

Algorithm 23 The TVBAS Algorithmic Procedure for Solving the TV-MCPITC Problem.

Input: The $data$, which is a time-series as a vector of n prices; the moving average's number of time periods $\tau \leq m-1$, $\tau \in \mathbb{N}$; the time interval $tspan = [t_{start}, t_{end}]$, where $t_{start}, t_{end} \in [0,\text{length}(data)-1]$; the parameter γ which divides the $tspan$ in γ equal steps, and the vector $noep$, which contains the number of observations in each period. Furthermore, a given portfolio η_p, the penalty parameter R, the fixed costs β^-, β^+ and the costs rates α^-, α^+. Lastly, the requirements of Alg. 21.

1: Construct p from Alg. 21, as well as f_p and f_X from Alg. 27, 28 or 29.
2: **function** $[x,y]$ = TVBAS(η_p,γ,$tspan$,$noep$,f_p,f_X,ϕ,R, d,δ,β^-, β^+,α^-, α^+)
3: Set $t = tspan(1) : \frac{1}{\gamma} : tspan(2)$, n =length(η_p) and tot =length(t)
4: Set x =zeros(n, tot), y =zeros($1, tot$), and $[\eta^-, \eta^+, A, b, p]$ =problem($noep, t(1), f_p, f_X, \phi, \eta_p, 0$)
5: Set $[x(:,1), y(1)]$ =basfunc($\eta_p, p, A, b, R, \eta^-, \eta^+, \beta^-, \beta^+, \alpha^-, \alpha^+, 0.9, 0.8, 1e-6, 1200$)
6: **for** $i = 2 : tot$ **do**
7: Set $[\eta^-, \eta^+, A, b, p]$ =problem($noep, t(i), f_p, f_X, \phi, x(:,i-1), y(i-1)$)
8: Set $[x(:,i), y(i)]$ =basfunc($x(:,i-1), p, A, b, R, \eta^-, \eta^+, \beta^-, \beta^+, \alpha^-, \alpha^+, 0.9, 0.8, 1e-6, 1200$)
9: **end for**
10: **end function**

Output: The online solution of the TV-MCPITC Problem.

In the following experiments, we have set $\beta_i^+ = \beta_i^- = 0.1$, $\alpha_i^+ = 0.06$ and $\alpha_i^- = 0.04$ in the function $\kappa_i(t)$, while the financial time-series used have been retrieved from https://finance.yahoo.com. The experiments cover two different portfolio configuration cases. The portfolio cases are presented in Fig. 5.13, with the market space containing some of the most active stocks in the US market. In the j-th case, $j = 1, 2$, we assume the market space $X = [x_1, x_2, \ldots, x_s]$, where X includes the daily close prices of the s stocks shown in Fig. 5.13.

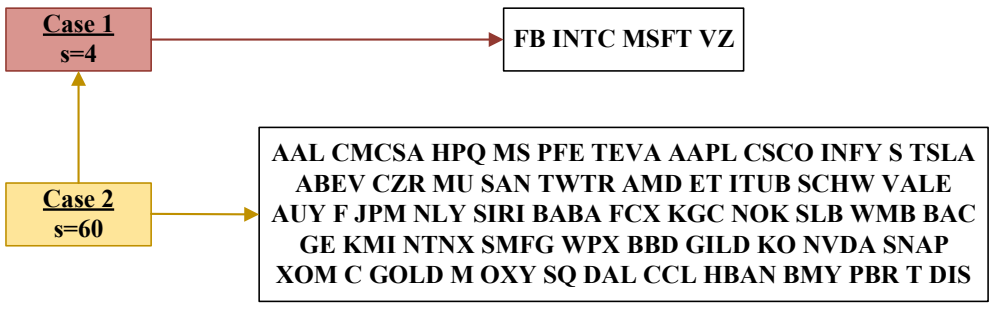
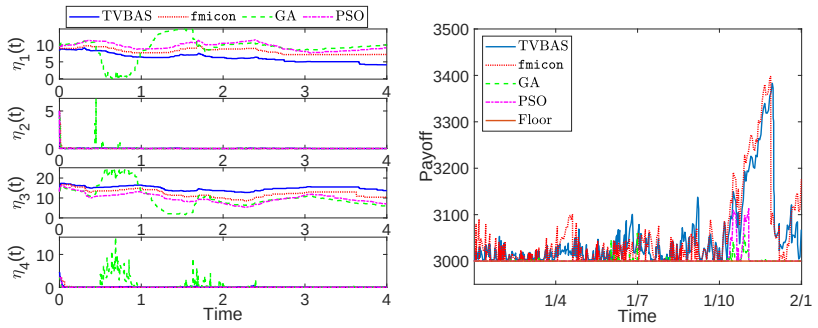


Figure 5.13: The portfolio cases stocks that have been utilized in the TV-MCPITC problem experiments.

On the one hand, the portfolio case 1 refers to a quarterly portfolio. For a period of one year from 18/10/2018 to 2/1/2020, we use the first 50 prices of the observations to calculate the insurance prices p of Alg. 21, where we also set $\tau = 50$, $\zeta = 5$ and $\theta = 3e3$. As a result, we find the TV-MCPITC problem's optimal solution for the period 2/1/2019 to 2/1/2020, which comprises of 253 observations. Note that this time-period can be divided

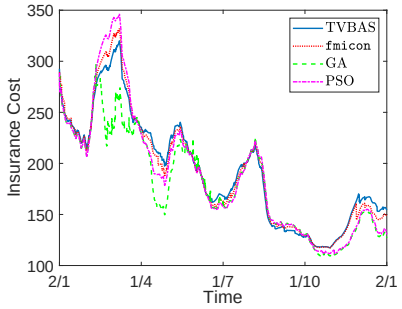
into quarters, i.e. $Q1, Q2, Q3$ and $Q4$, while each quarter comprises of three months. In particular, $Q1$ of 2019 has 61 observations, $Q2$ has 63, $Q3$ has 64 and $Q4$ along with the one observation from the year 2020 has 65 observations. So, we have $noep = [61, 63, 64, 65]$ at $tspan = [0\ 4]$. Also, we use the linear data interpolation in order to convert p and X into functions of time $f_p(t)$ and $f_X(t)$, respectively, through the custom MATLAB function `linots`. Given a portfolio $\eta_p = [2, 5, 7, 3]^{\mathrm{T}}$, we set $\gamma = 1000$, $R = 1e5$ and the constant floor $\phi = 3000$. The results are presented in Fig. 5.14.

On the other hand, the portfolio case 2 refers to the period 5/6/2019 to 3/2/2020. We use the first 40 prices from the aforementioned time series to calculate the insurance prices p of Alg. 21, where we also set $\tau = 40$, $\zeta = 2$ and $\theta = 1e3$. As a consequence, we find the TV-MCPITC problem's optimal solution for the period 1/8/2019 to 3/2/2020 with 128 observations. In particular, August comprises 22 observations, September and November comprise 20 observations each, October 21 and December with January have 22 observations together. So, $noep = [22, 20, 23, 20, 21, 22]$ at $tspan = [0\ 6]$. Also, we use cubic spline data interpolation in order to convert p and X into time-varying functions $f_p(t)$ and $f_X(t)$, respectively, through the custom MATLAB function `pchinots`. Given a portfolio η_p =ones$(s, 1)$, we set $\gamma = 1000$, $R = 1e8$ and a floor $\phi = @(t)\phi = @(t)2800 + 200t$. The results are as shown in Fig. 5.15

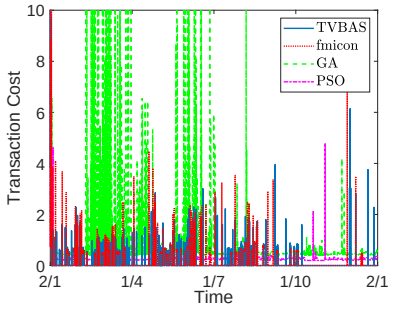


(a) Convergence of portfolios.

(b) Payoff of portfolios.



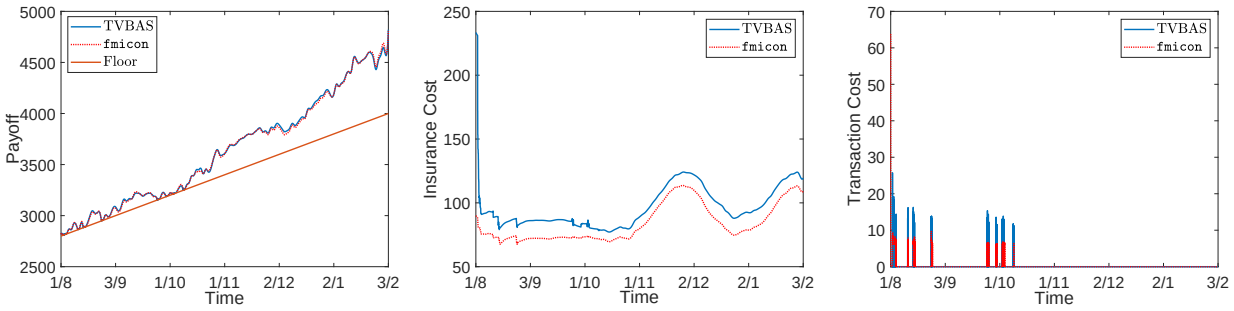
(c) Insurance Costs of portfolios.



(d) Transaction costs of portfolios.

Figure 5.14: The convergence, the payoff, the costs of portfolios in case 1.

The results that are depicted in Fig. 5.14a show that the TVBAS solves the TV-MCPITC problem and produces its online solution $\eta(t)$. The solution of the TVBAS is similar to the solution of the MATLAB functions `fmincon`, GA and PSO. The payoff of the portfolio $\eta(t)$ is shown in Figs. 5.14c and 5.15b, while its insurance costs are shown in Figs. 5.14b and 5.15a. The transaction costs are shown in Figs. 5.14d and 5.15c. In portfolio case 1, it is observable that the TVBAS solution requires on average the least transaction costs compared to the solutions generated by `fmincon`, GA and PSO, while their payoff and insurance costs are close. In portfolio case 1, it is observable that the TVBAS solution is not the optimal but it is very close to the solution generated by `fmincon`.



(a) Payoff of portfolios. **(b)** Insurance Costs of portfolios. **(c)** Transaction costs of portfolios.

Figure 5.15: The payoff and the costs of portfolios in case 2.

It is worth noting that we do not compare the GA and PSO in portfolio case 2 because of their slow execution time and the constraints violation in the TV-MCPITC problem which is caused by the portfolio's large size. It is also observable that the transaction costs of the portfolio increase when the insurance costs of the portfolio reduce. The time consumption of these experiments is presented in Tab. 5.4, and shows that the TVBAS is on average much faster compared with the second faster `fmincon` MATLAB function, while the GA is the slowest. Overall, the TVBAS worked excellently in solving the TV-MCPITC problem.

Table 5.4: The TVBAS, GA, PSO and `fmincon` time consumption for solving the TV-MCPITC problem.

Interpolation Function	Case 1			
	4 Stocks Portfolio			
	TVBAS	`fmincon`	GA	PSO
`linots`	55.6 s	81 s	2522 s	94 s
`pchinots`	57.3 s	81.7 s	2569 s	95.2 s
`splinots`	57.2 s	84.3 s	2480 s	92.8 s
	Case 2			
	60 Stocks Portfolio			
	TVBAS	`fmincon`	GA	PSO
`linots`	113.6 s	132.7 s	-	-
`pchinots`	114.1 s	137.1 s	-	-
`splinots`	114.7 s	137.2 s	-	-

5.3.3 Numerical Experiments on the TV-MVPSTC-CC Problem

The numerical experiments of this section comprise of two portfolio cases. The portfolio cases are presented in Fig. 5.16, with the market space containing some of the most active stocks in the US market. More precisely, in the j-th case, $j = 1, 2$, we assume the market space $X = [x_1, x_2, \ldots, x_s]$, where X includes the daily close prices of the s stocks shown in Fig. 5.13, and then we solve online a TV-MVPSTC-CC problem by using the TVPBAS of Alg. 14. Furthermore, some additional information must be given for the following experiments as preliminaries. First, the function $\phi_i(t)$ has the same coefficients for all portfolio cases. Thus, in $\phi_i(t)$, we have set $\beta_i^+ = \beta_i^- = 0.02$, $\alpha_i^+ = 0.06$ and $\alpha_i^- = 0.04$. Second, we have set $\gamma = 10$, $d = 0.2$, $\delta = 0.5$, $tol = 1e{-}6$ and $kmax = 1200$ in Algorithm 11. Third, the price of r_p is a user's choice as long as $r_p(t) \in [\min(r(t)), \max(r(t))] \subseteq \mathbb{R}$. Thus, we have set

$r_p = -\min(r)$ based on our preferences and in order to find the minimum variance possible. Lastly, the procedure for construction of the portfolio expected return and covariance is presented in the following Alg. 24.

Algorithm 24 Algorithm for the portfolio expected return and covariance.

Input: The marketed space $X = [x_1, x_2, \ldots, x_n]$ which is a matrix of n time series as column vectors of m prices, the moving average's number of time periods $\tau \leq m-1$, $\tau \in \mathbb{N}$.

1: Set $[m, n]$ =size(X)
2: Set r =movavg$(X,'\text{method}', \tau)$
3: Set $r = r(\tau + 1 : \text{end}, :)$
4: Set $C\{m - s, 1\} = \{\}$
5: **for** $i = 1 : m - \tau$ **do**
6: Set $C\{i, 1\}$ =cov$(X(i : \tau + i - 1, :))$
7: **end for**

Output: The structure array C comprises of the covariance matrices for each time period of all time-series of the normalized portfolio and the matrix r consists of the expected return for a number of time periods of each time-series of the portfolio.

It is worth mentioning that a complete implementation of the methods proposed in this section can be obtained from GitHub in the following link:

https://github.com/SDMourtas/TV-MVPSTC-CC.

Therein, we developed a MATLAB package inline with Algs. 9-14 for solving the TV-MVPSTC-CC problem. The MATLAB package contains comprehensive installation specifications as well as a detailed implementation of numerical experiments of this section, which uses real-world data. In addition, by feeding with their own data and changing the parameters values in the `Main_TVMVPSTCCC` MATLAB file of the package, everyone can draw conclusions from their own results. Note that the data used in the MATLAB package are taken from the Yahoo finance (https://finance.yahoo.com/) and includes the daily close prices of some of the most active stocks on the market. Finally, the purpose of this package is to solve online the TV-MVPSTC-CC problem by using a TV auto penalty-based Beetle Antennae Search (TVPBAS) algorithm. Several algorithms are currently implemented, based on the available literature and our understanding.

On the one hand, the portfolio case 1 refers to a quarterly portfolio for a period of one year from 18/10/2018 to 2/1/2020. For the portfolio case 1 time-series, the initial 50 observation prices have been used to compute the expected return matrix r, where every r_i is an $EMA50$, and the covariance structure C of Alg. 24. Consequently, we set $\tau = 50$ and $method = exponential$ in Alg. 24. The remaining data is from 2/1/2019 to 2/1/2020 which comprise of 253 observations. More precisely, Q1 of 2019 has 61 observations, Q2 has 63, Q3 has 64 and Q4 together with the first observation of the year 2020 include 65 observations. Note that, every quarter consists of three months and Q1, Q2, Q3 and Q4 imply the 1st, 2nd, 3rd and 4th quarter of a year, respectively. Thus, we have $noep = [61, 63, 64, 65]$ at $tspan = [0\ 4]$. Furthermore, linear data interpolation has been used in order to transform r, C and X into time-dependent functions $f_r(t)$, $f_C(t)$ and $f_X(t)$, respectively, via custom MATLAB

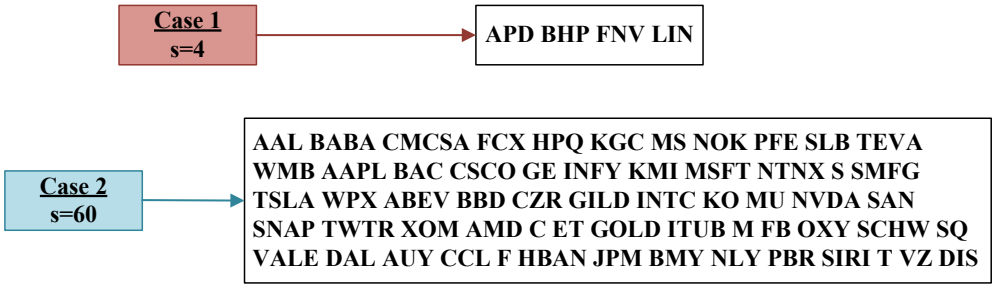
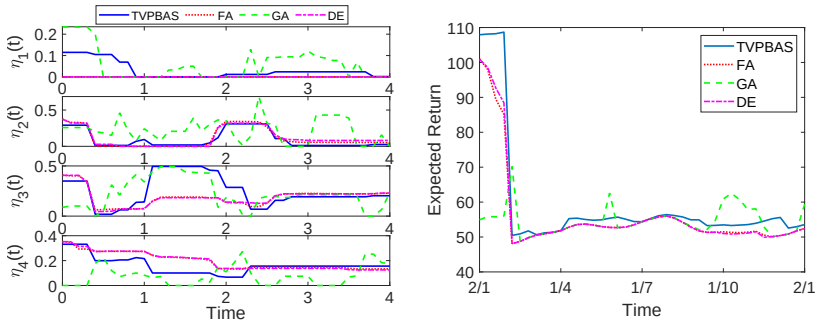


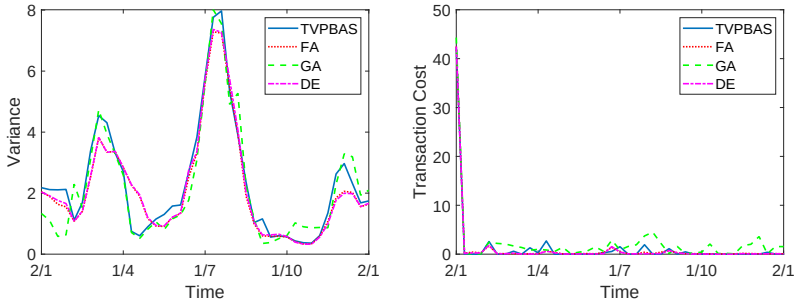
Figure 5.16: The portfolio cases stocks that have been utilized in the TV-MVPSTC-CC problem experiments.

functions `linots` and `linotss`. Considering a portfolio $x_p = [2,3,1,4]^{\mathrm{T}}$, we set cardinality $K = 3$. The results are as shown in Fig. 5.17

On the other hand, the portfolio case 2 refers to a monthly portfolio from 19/6/2019 to 3/2/2020. For the portfolio case 2 time-series, the initial 30 prices have been used to compute the expected return matrix r, where every r_i is an $EMA30$, and the covariance structure C of Alg. 24. Consequently, we set $\tau = 30$ and $method = exponential$ in Alg. 24. The remaining data is from 2/1/2019 to 2/1/2020 which comprise of 253 observations. More precisely, Aug. has 22 observations, Sept. and Nov. include 20 observations each, Oct. has 21 observations and Dec. together with the first observation of Jan. include 22 observations. Thus, we have $noep = [22,20,23,20,21,22]$ at $tspan = [0\ 6]$. Furthermore, linear data interpolation has been used in order to transform r, C and X into time-dependent functions $f_r(t)$, $f_C(t)$ and $f_X(t)$, respectively, via custom MATLAB functions `linots` and `linotss`. For the portfolio x_p =ones$(s,1)/4$, we use $K = 16$. The results are as shown in Fig. 5.18

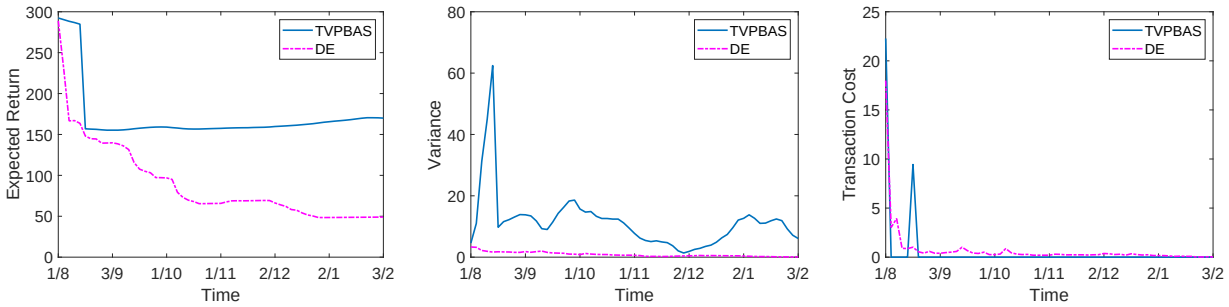


(a) Convergence of portfolios. **(b)** Expected return of portfolios.



(c) Variance of portfolios. **(d)** Transaction costs of portfolios.

Figure 5.17: The convergence, the expected return, the variance and the transaction costs of the portfolios in case 1.



(a) Expected return of portfolios. **(b)** Variance of portfolios. **(c)** Transaction costs of portfolios.

Figure 5.18: The expected return, the variance and the transaction costs of portfolios in case 2.

The findings that are depicted in Fig. 5.17a show that TVPBAS solves the TV-MVPSTC-CC problem and generates its online solution, $\eta(t)$, while it is observable that TVPBAS solution is close to the solution of FA, GA and DE. In Figs. 5.17b, 5.17c, 5.17d and 5.18a, 5.18b, 5.18c, we can see the expected return, the variance and the transaction costs of portfolio cases 1 and 2, respectively. Therein, we observe that the TVPBAS solution is always close to the solutions of the other methods. Also, we remark that when the variance of the portfolio is reduced, the expected return of the portfolio reduces too. Note that in portfolio case 2, we do not contrast the GA because of its sluggish time performance that is caused by the portfolio's big size, as well as we do not compare the FA because of its low efficacy that is caused by the portfolio's big size. The time consumptions of numerical experiments are displayed in Tab. 5.5. Therein, we observe that TVPBAS is more than 10 times faster than the second fastest FA in portfolio case 1, while DE is the slowest of them all. Moreover, TVPBAS is on average more than 30 times faster than DE in portfolio case 2. In general, TVPBAS has worked extremely well to solve the TV-MVPSTC-CC problem.

Table 5.5: The TVPBAS, FA, GA and DE time consumption for solving the TV-MVPSTC-CC problem.

Portfolio	Interpolation	TVPBAS	FA	GA	DE
Case 1	Linear/C.Spline	1.7 s	23.3 s	65.7 s	67.7 s
Case 2	Linear/C.Spline	3.7 s	-	-	120.2 s

5.3.4 Numerical Experiments on the BMPS Problem

For the experiments in this subsection, some additional information must be given. First, we have set the maximum iterations to 1000 in VSBAS, BBAS, BBA, BGA and VPSO. In this way, we can monitor and compare the efficiency of the algorithms, which is based on the time consumption and the optimal solution. Furthermore, the population size has been set to 30, 50 and 30 for BBA, BGA and VPSO, respectively, and the Random Selection method has been used in BGA. The rest of the algorithms parameters have been set to their default state. Second, since all the algorithms are directly applicable only to unconstrained optimization, we use the penalty function method of [9] that will keep their solutions in the feasible district. Penalty functions operate in a series of steps, modifying a set of penalty parameters each time and beginning a new one with the previous one. During the construction of any step, the following penalty function is minimized:

$$Q(x, P) = f(x) - O(P, p(x)), \tag{5.1}$$

where $f(x)$ is the objective function and $O(P, p(x))$ is the penalty term, with P being a set of penalty parameters and $p(x)$ being the inequality constraint function. This method has the advantage of being able to satisfy both convex and nonconvex constraints. The penalty method employed in this approach embodies the bracket operator $\langle\cdot\rangle$ with $\langle z\rangle = 0$, if z is positive, otherwise $\langle z\rangle = z$. This operator is primarily used to manipulate inequality constraints. Inhere, we use the penalty function method as proposed in [155]. That is, the task in all algorithms is to minimize (5.1) instead of (4.145), where the penalty parameter P value has been set to $1e5$. Third, the parameters in (4.144) have been set to $\theta^- = \theta^+ = 0.03$, $\zeta^- = 0.05$ and $\zeta^+ = 0.07$. Last, all the data used in this section are the daily close prices of the stocks included in Fig. 5.19. Note that Fig. 5.19 contains the stocks' ticker symbols.

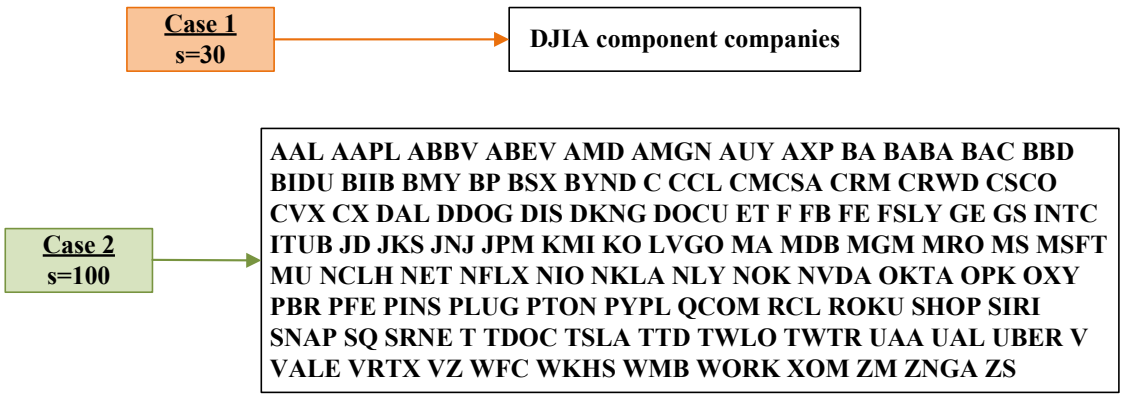


Figure 5.19: The portfolio cases stocks that have been utilized in the BMPS problem experiments.

The full procedure for solving a BMPS problem employing VSBAS can be summarized in the following Alg. 25, according to subsection 4.7.2.

Algorithm 25 Algorithm for solving a BMPS problem employing VSBAS.

Input: The data $X(t) \in \mathbb{R}^n$, the maximum time number m, the number of delays $\beta \in \mathbb{N}$, the cardinality limits $\mu^-, \mu^+ \in \mathbb{N}$, an initial portfolio $\eta(\beta) \in \mathbb{R}^n$, the fixed rates prices θ^- and θ^+, the cost rates ζ^- and ζ^+, the penalty parameter P.

1: Set $i = \beta$ and construct $C(t)$ and $r(t)$ for $t = \beta$ to $m-1$
2: **while** $i < m$ **do**
3: Set $O(P, p(\eta(i))) = P(\text{sum}((r^{\mathrm{T}}(i)\cdot\eta(i) + R(i) < 0) \odot (r^{\mathrm{T}}(i)\cdot\eta(i) + R(i))^{\odot 2})) + P^2(\text{sum}(\eta(i) > 0) < \mu^-) + P^2(\text{sum}(\eta(i) > 0) > \mu^+)$
4: Set $f(\eta(i))$ ←(4.145) and as objective function $Q(\eta(i), P)$ ←(2.4)
5: Employ Alg. 16 on $Q(\eta(i), P)$ to find the optimal $\eta(i+1)$.
6: Set $i = i + 1$
7: **end while**

Output: The optimal solution $\eta(t)$ for $t = \beta + 1$ to m.

The complete development and implementation of the computational methods proposed in this paper can be

obtained from GitHub. More precisely, in the following link:

https://github.com/SDMourtas/BMPS,

we developed a MATLAB package inline with the process described in this section for solving the BMPS problem. The MATLAB package contains comprehensive installation specifications as well as a detailed implementation of the applications described in this section, which use real-world data. In addition, by feeding with their own data and changing the parameters values in the main MATLAB functions of the package, everyone can draw conclusions from their own results. Note that the data used in the MATLAB package are taken from the Yahoo finance (https://finance.yahoo.com/) and includes the daily close prices of some of the most active stocks on the market.

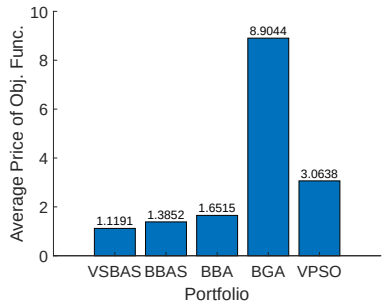
The experiments in this subsection cover two different portfolio configuration cases. In the first case, the market space contains the stocks of the DJIA component companies, while in the second case, the market space contains some of the most active stocks in the US market, see also Fig. 5.19.

In the i-th case, $i = 1, 2$, we consider the market space to be $X_i = [x_1, x_2, \ldots, x_s]$, where X_i contains the daily close prices of the s stocks of DJIA component companies ($i = 1,\ s = 30$) from 3/12/2019 to 4/1/2021, totaling 274 observations, and s stocks located in Fig. 5.19 ($i = 2,\ s = 100$), from 4/12/2019 to 1/6/2020, totaling 123 observations. Therefore, $X_1 \in \mathbb{R}^{274\times 30}$, where $X_1(t) \in \mathbb{R}^{30}$ at time $t = 1, 2, \ldots, 274$ and $X_2 \in \mathbb{R}^{123\times 100}$, where $X_2(t) \in \mathbb{R}^{100}$ at time $t = 1, 2, \ldots, 123$. We create the vector $r(t) = [r_1(t), r_2(t), \ldots, r_s(t)] \in \mathbb{R}^s$ by taking the number of delays, $\beta = 20$ in case 1 and $\beta = 40$ in case 2. Then, $r(t)$ contains the normalized expected returns on the s in number assets at time $t = 21, 32, \ldots, 274$ (case 1) or at time $t = 41, 42, \ldots, 123$ (case 2). In addition, we create the covariance matrix of the normalized market $C(t) \in \mathbb{R}^{s\times s}$ at time $t = 21, 32, \ldots, 274$ (case 1) or at time $t = 41, 42, \ldots, 123$ (case 2). Given an initial portfolio $\eta(\beta) = [\xi^-, \xi^+] \in \mathbb{R}^s$, where $\xi^- = \mathbf{1} \in \mathbb{R}^{s/2}$ and $\xi^+ = \mathbf{0} \in \mathbb{R}^{s/2}$, we find the optimal BMPS portfolio, with cardinality

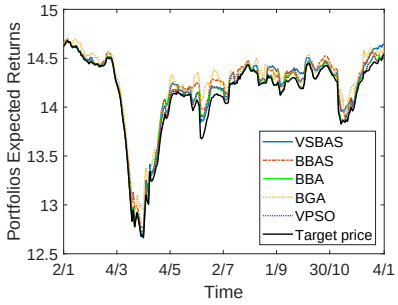
$$\mu^- = \begin{cases} 12, \text{case } 1 \\ 16, \text{case } 2 \end{cases} \quad \text{and } \mu^+ = \begin{cases} 40, \text{case } 1 \\ 45, \text{case } 2 \end{cases}$$

and target price $R = \mu^+ \cdot \text{mean}(r(t))$ at time $t = 21, 32, \ldots, 274$ (case 1, period 2/1/2020 to 4/1/2021) or at time $t = 41, 42, \ldots, 123$ (case 2, period 3/2/2020 to 1/6/2020).

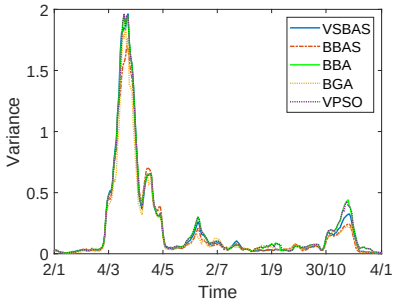
The results are as depicted in Fig. 5.20 for portfolio cases 1 and 2. More precisely, the first case deals with a 30 stocks portfolio, where these stocks correspond to the DJIA component companies. The results of this case are depicted in Figs. 5.20a-5.20d, where we observe that the solutions of VSBAS, BBAS, BBA and VPSO are close. Moreover, the solution of VSBAS is very close to BBAS, while the solution of BGA is far away from them. Fig. 5.20a shows the average price of (4.145), which is the average value of variance plus the transaction costs of the optimal portfolio $\eta(t)$ for the period 2/1/2020 to 4/1/2021. VSBAS generates the lowest average price, while BBAS generates the second lowest average price, which is very close to the lowest average price. In addition, BBA and VPSO generate the third and fourth lowest average prices, respectively, while BGA generates the highest average price. Fig. 5.20b shows the portfolio variance, where we observe that the variances produced by VSBAS, BBAS, BBA, BGA and VPSO are almost identical. Fig. 5.20c shows the portfolio transaction costs, where we observe that the lowest portfolio transaction costs are found in VSBAS, the highest in BBAS, and the most transactions are found in VPSO. Fig. 5.20d shows the portfolio expected returns along with the target price R. Therein, all



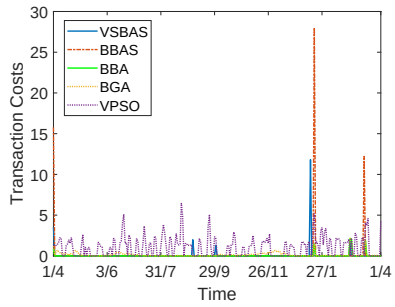
(a) Average of (4.145) in portfolio case 1.



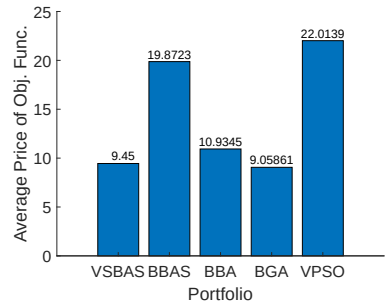
(b) Expected return of portfolios in case 1.



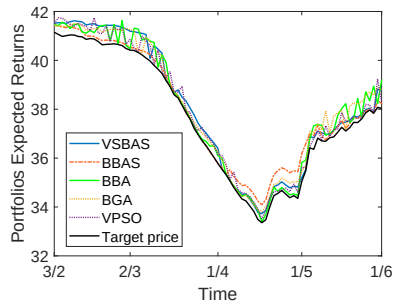
(c) Variance of portfolios in case 1.



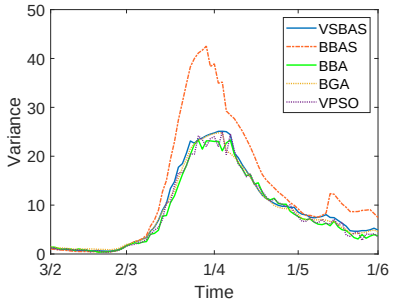
(d) Transaction costs of portfolios in case 1.



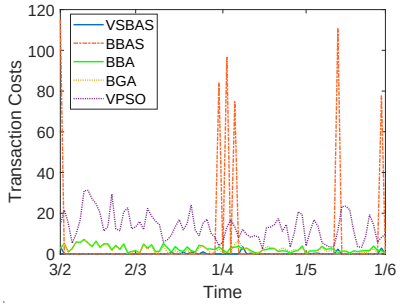
(e) Average of (4.145) in portfolio case 2.



(f) Expected return of portfolios in case 2.



(g) Variance of portfolios in case 2.



(h) Transaction costs of portfolios in case 2.

Figure 5.20: The average of (4.145), the expected return, the variance and the transaction costs of portfolios in cases 1 and 2.

the expected returns are above the target price R, the expected returns of VSBAS, BBAS, BBA, BGA and VPSO portfolios are almost identical. The time consumption of this experiment is depicted on Tab. 5.6 and indicates that BBAS is less than two time faster than VSBAS, while it is on average more than 10 times faster compared to VPSO and BBA, and almost 69 times faster compared to the slowest BGA.

The results of the second case, which deals with a 100 stocks portfolio, are depicted in Figs. 5.20e-5.20h. Therein, we observe that the solutions of VSBAS, BBA and BGA are very close, while the solutions of BBAS and VPSO are far away from them. Fig. 5.20e shows the average price of (4.145), which is the average value of variance plus the transaction costs of the optimal portfolio $\eta(t)$ for the period 3/2/2020 to 1/6/2020. In there, BGA produces the minimum average price, while the average prices produced by VSBAS and BBA are very close to the minimum average price. In addition, VPSO produces the maximum average price. Fig. 5.20f shows the portfolio variance, where we observe that the variances produced by VSBAS, BBA, BGA and VPSO are close, while the variance

produced by BBAS is the highest. Fig. 5.20g shows the portfolio transaction costs, where we observe that VSBAS portfolio transaction costs are the lowest, while the transaction costs of BBAS portfolio are the highest. Fig. 5.20h shows the portfolio expected returns along with the target price R. Therein, all the expected returns are above the target price R and the expected returns of VSBAS, BBAS, BBA, BGA and VPSO are close. The time consumption of this experiment is depicted on Tab. 5.6 and indicates that BBAS is on average almost 1.5 time faster than VSBAS, while it is on average more than 17 times faster compared to VPSO and BBA, and almost 44 times faster compared to the slowest BGA.

Table 5.6: The VSBAS, BBAS, BBA, BGA and VPSO time consumption for solving the BMPS problem.

Portfolio	VSBAS	BBAS	BBA	BGA	VPSO
Case 1	5.4 s	3 s	38.3 s	206 s	36.8 s
Case 2	3.6 s	2.3 s	39.7 s	102.1 s	39.4 s

Based on the aforementioned analysis, we can conclude that BBAS is the most efficient algorithm for an input of max size 40. This occurs because the BBAS problem of trapping local minima is more prevalent in problems with large dimensions. In addition, for an input of size 40 and above, VSBAS is the most efficient algorithm. This is because the VSBAS incorporates a V-shaped transfer function into the BBAS process, which solves the BBAS problem of trapping in local minima. Note that all algorithms can produced better outcome by rising their iterations but at the same time their time consumption is increasing proportional too. Thus, the general conclusion is that VSBAS worked excellently and efficiently in solving the BMPS problem, especially in large portfolio dimension.

5.3.5 Numerical Experiments on the TV-TPNC Problem

This subsection compares and contrasts SIBAS' performance with those of state-of-the-art meta-heuristics algorithms such as PSO of MATLAB, DE of [13] and SMA of [208] in solving the TV-TPNC problem of (4.157)-(4.160). The daily close prices of the stocks shown in Fig. 5.21 are the real-world data employed. This figure contains stocks' ticker symbols divided into two portfolio's cases. This section also contains information regarding the data and code availability. Moreover, in all experiments along with all the nature inspired algorithms used inhere, the penalty parameter has been set to $R = 1e5$, and the maximum iterations to $1e3$. The SIBAS parameters have been set to $d = 0.2$ and $\delta = 0.5$, the PSO used with its default settings and the population size of SMA and DE have been set to 30 and 50, respectively. The variance (risk) number has been set to $\alpha = 1e-3$, and the delays number has been set to $\beta = 40$, while the parameters in (4.151) have been set to $\zeta^- = 2$, $\zeta^+ = 4$ and $\theta^- = \theta^+ = 1$. It is worth mentioning that the abbreviations SR, TC and CC refer to Sharpe Ratio, transaction costs and cardinality constraint, respectively.

In the i-th portfolio case, $i = 1, 2$, we assume the market dataset to be $M \in \mathbb{R}^{123\times s}$. Note that $i = 1$ has $s = 40$ and $K = 20$, while $i = 2$ has $s = 80$ and $K = 40$. Based on this and the number of delays, we construct the market $X(t) = [x_1, x_2, \ldots, x_s] \in \mathbb{R}^s$ for $t = 1$ to 83. That is, $X(t)$ contains 83 daily prices of s in number stocks that correspond to the time-period 3/2/2020-1/6/2020. Because of the cardinality number, the optimal portfolio $\eta_{\text{opt}}(t)$ holds exactly K in number stocks, at least one of which is risk-free. The findings for solving the TV-TPNC

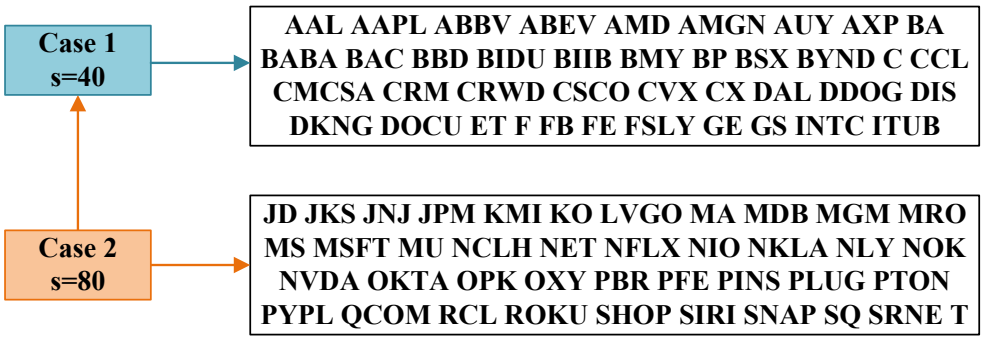
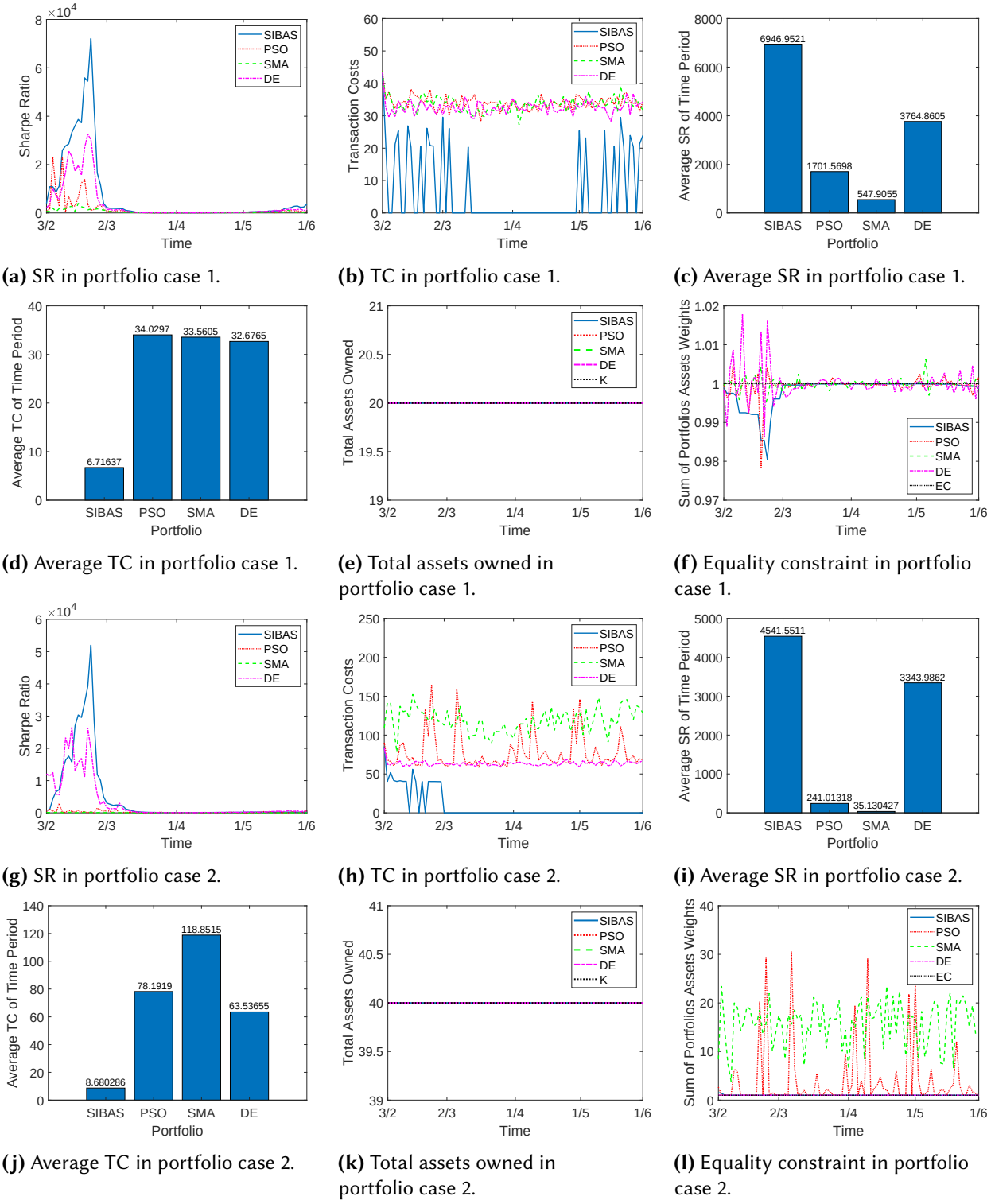


Figure 5.21: The portfolio cases stocks that have been utilized in the TV-TPNC problem experiments.

problem with initial portfolio $\eta_{\text{in}} = \mathbf{1}/s \in \mathbb{R}^s$ are presented in Figs. 5.22a-5.22f for the portfolio case 1, and in Figs. 5.22g-5.22l for the portfolio case 2.

On the one hand, Figs. 5.22a, 5.22g depict the SR of the portfolios under a market containing 40 and 80 stocks, respectively. Therein, it can be observed that the SR produced by the optimal portfolio of SIBAS is always higher than the optimal portfolios produced by PSO, SMA, and DE. Figs. 5.22b, 5.22h show the TC, where it is observable that SIBAS optimal portfolios have always lower TC compared to PSO, SMA, and DE. The TC of the PSO, SMA, and DE optimal portfolios are similar in portfolio case 1, however they are not in portfolio case 2. Figs. 5.22c, 5.22i show the average SR during the time period for the portfolio cases 1 and 2, respectively. According to these figures, SIBAS optimal portfolios produce the highest SR during the specific time period in both portfolio cases, while DE optimal portfolios produce the second highest SR and SMA optimal portfolios produce the lowest SR. Figs. 5.22d, 5.22j show the average TC during the time period for the portfolio cases 1 and 2, respectively. Based on these figures, SIBAS optimal portfolios produce the lowest TC during the specific time period in both portfolio cases, while DE optimal portfolios produce the second lowest TC and SMA optimal portfolios produce the highest TC. Figs. 5.22e, 5.22k show the total assets owned from the optimal portfolios produced by SIBAS, PSO, SMA, and DE during the time period along with the cardinality number K for the portfolio cases 1 and 2, respectively. Therein, it is observable that all portfolios always owns K in number assets and, hence, the CC is satisfied in both portfolio cases. Figs. 5.22f, 5.22l show the sum of the optimal portfolios assets weights, which is the left part of (4.158), produced by SIBAS, PSO, SMA, and DE during the time period for the portfolio cases 1 and 2, respectively, along with the equality constraint (EC) number of (4.158), which is equal to 1. Therein, it is observable that the outcome of the SIBAS optimal portfolios always have the least noise and are closest to 1 in both portfolio cases. That is, SIBAS produces the best outcome in both portfolio cases, while SMA produces the second best outcome and DE the worst in portfolio case 1, and DE produces the second best outcome and SMA the worst in portfolio case 2.

Figs. 5.23a, 5.23b present the SIBAS, PSO, SMA, and DE convergence in the portfolio cases 1 and 2, respectively, for $t = 1$, while the corresponding time consumption of SIBAS, PSO, SMA, and D at each iteration is presented in Figs. 5.23c, 5.23d, respectively. That is, the value of (4.157) at each iteration of the SIBAS, PSO, SMA, and DE when the time-period for solving the TV-TPNC problem is 3/2/2020 is depicted in Figs. 5.23a, 5.23b. In Figs. 5.23a, 5.23b, we observe that SIBAS has the best convergence in both portfolio cases, whereas SMA has the worst, with PSO



(a) SR in portfolio case 1.

(b) TC in portfolio case 1.

(c) Average SR in portfolio case 1.

(d) Average TC in portfolio case 1.

(e) Total assets owned in portfolio case 1.

(f) Equality constraint in portfolio case 1.

(g) SR in portfolio case 2.

(h) TC in portfolio case 2.

(i) Average SR in portfolio case 2.

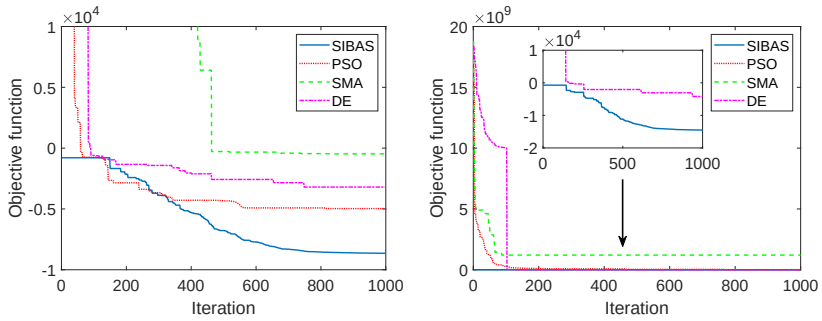
(j) Average TC in portfolio case 2.

(k) Total assets owned in portfolio case 2.

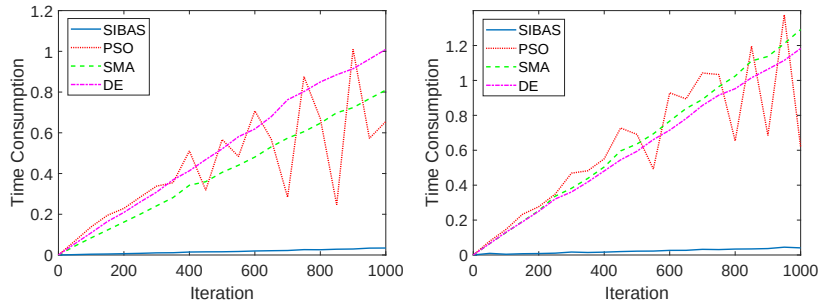
(l) Equality constraint in portfolio case 2.

Figure 5.22: The SR and TC, the average SR and TC of time-period, the total assets owned and the equality constraint of the two portfolio cases.

having the second best convergence in portfolio case 1 and DE having the second best convergence in portfolio case 2. In Figs. 5.23c, 5.23d, we observe that SIBAS has the lowest time consumption in both portfolio cases, whereas DE has the highest time consumption in portfolio case 1 and SMA has the highest time consumption in



(a) Convergence of portfolios. **(b)** Payoff of portfolios.



(c) Insurance Costs of portfolios. **(d)** Transaction costs of portfolios.

Figure 5.23: The SIBAS, PSO, SMA, and DE convergence and time consumption in the two portfolio cases for $t = 1$.

portfolio case 2. Furthermore, the time consumption of PSO is non-linear and quite noisy in both portfolio cases. However, PSO has the second lowest time consumption in both portfolio cases at iteration 1000. SIBAS has the lowest computational complexity, as shown in Figs. 5.23a and 5.23b, since it converges faster to the optimum solution in less iterations than PSO, SMA, and DE. Also, SIBAS has the lowest time complexity, as shown in Figs. 5.23c and 5.23d, since it takes less time to complete an iteration than PSO, SMA, and DE. As a result, SIBAS outperforms PSO, SMA, and DE when it comes to solving the TV-TPNC problem.

The average time consumption demanded from SIBAS, PSO, SMA, and DE to generate the optimal solutions for the TV-TPNC problem in both portfolio cases, on the other hand, is contained in Tab. 5.7. It is evident from this that SIBAS is always the quickest algorithm. More particularly, SIBAS is about 5 times faster in portfolio case 1 than the second fastest PSO, and more than 10 times faster in portfolio case 2. Moreover, SIBAS is about 28 times faster than the third faster DE in both portfolio cases, while it is more than 40 times faster in portfolio case 1 than the slowest SMA, and about 60 times faster in portfolio case 2.

The above analysis leads to the conclusion that SIBAS performed admirably and effectively in resolving the TV-TPNC problem. According to Figs. 5.22, 5.23 and Tab. 5.7 results, the SIBA produces more effective optimal portfolios than the PSO, SMA, and DE, whereas SMA produces the least effective ones. When contrasted to PSO, SMA, and DE, the average time consumption of SIBAS is the shortest, whereas as the market dimension grows, its accuracy falls less than that of PSO, SMA, and DE. This implies that market size has a significant impact on SIBAS, PSO, SMA, and DE performance.

The whole design and implementation of the computational approaches suggested in this paper may be seen on GitHub:

https://github.com/SDMourtas/TV-TPNC

Table 5.7: The SIBAS, PSO, SMA and DE average time consumption for solving the TV-TPNC problem.

Portfolio	SIBAS	PSO	SMA	DE
Case 1	4.2 s	19.8 s	173.8 s	118.9 s
Case 2	5.3 s	56.4 s	320.5 s	144.9 s

There, we created a MATLAB repository for solving the TV-TPNC problem inline with Alg. 17. The MATLAB repository includes thorough installation instructions along with a meticulous implementation of the real-world data applications mentioned in this subsection. Furthermore, anyone may draw conclusions from their own findings by providing the repository's main MATLAB function with their own data and adjusting the parameter values. Notice that the MATLAB repository's data comes from Yahoo Finance (`https://finance.yahoo.com/`) and contains some of the market's most active stocks daily close prices.

6
Conclusion

6.1 Summary of Main Contributions

With the technical contributions of the current doctoral dissertation presented, we may now reflect on the milestones of the entire effort in the direction of time-varying financial modeling and the application of intelligent algorithms. Portfolio management is one of the most important subjects in finance, as seen by the engagement with prior art. Because of the rapid advancement of computer science, as well as modern information technology and financial development, it is clear that more advanced computational methods for portfolio management can be developed. In the current dissertation we have verified the above observation, both in terms of portfolio modeling and computational methods, especially for the case of time-dependent variability.

Starting from popular portfolio selection problems, we developed and presented several time-varying (TV) financial optimization problems. Particularly, based on the following financial optimization problems:

- the minimum-cost portfolio insurance (MCPI) problem;
- the Markowitz's mean-variance portfolio selection (MVPS) problem;
- the Black-Litterman portfolio optimization (BLPO) problem;
- the tangency portfolio (TP) problem,

we created their TV versions and by adding nonlinear constraints (NC), which refer to transaction costs (TC) and cardinality constraints (CC), the following TV financial optimization problems are presented in this dissertation:

- the TV MCPI (TV-MCPI) problem [150, 151];
- the TV MVPS (TV-MVPS) problem [152, 153];
- the TV BLPO (TV-BLPO) problem [69];
- the multi-period (MP) MCPI under TC (MPMCPITC) problem [154];

- the TV MCPI under TC (TV-MCPITC) problem [155];
- the TV MVPS under TC and CC (TV-MVPSTC-CC) problem [156];
- the binary Markowitz-based portfolio selection (BMPS) problem [157];
- the TV TP under NC (TV-TPNC) problem [158].

These financial problems can be classified into two categories. The one category includes TV linear programming (LP) problems and TV quadratic programming (QP) problems, while the other category includes TV nonlinear programming (NLP) problems and TV integer linear programming (ILP) problems. As a result, the TV-MCPI problem is defined and studied as a TV LP problem, while the TV-MVPS and the TV-BLPO problems are defined and studied as TV QP problems. In addition, the TV-MCPITC, TV-MVPSTC-CC, MPMCPITC and TV-TPNC problems are defined and studied as TV NLP problems, while the BMPS problem is defined and studied as a TV ILP problem. Based on this classification, intelligent online optimization algorithms, which include modern neural network (NN) techniques and state-of-the-art metaheuristics, are employed to solve the aforementioned TV financial optimization problems using real-world datasets. More specifically, two continuous-time (CT) NN solvers, the zeroing NN (ZNN) and the linear-variational-inequality primal-dual NN (LVI-PDNN), are used to approach the TV LP/QP financial optimization problems, whereas metaheuristics, such as the beetle antennae search algorithm (BAS) and its variations, the TVBAS, TVPBAS, VSBAS, and SIBAS, are used to approach the TV NLP/ILP financial optimization problems. It is worth noting that the BAS variants handle issues like large dimensionality, local minima trapping, and equality constraint accuracy.

Several numerical experiments in various portfolio setups with real-world datasets have shown the efficiency and accuracy of the intelligent algorithms presented in this dissertation. Apart from the fact that continuous NN solvers like ZNN and LVI-PDNN have never been used in finance, all of the proposed metaheuristic algorithms, which are based on BAS, were compared to a number of state-of-the-art metaheuristic algorithms. All of the findings indicate that the proposed intelligent algorithms are a suitable alternative to conventional and state-of-the-art approaches.

6.2 Study Limitations and Future Work

The online solution of a time-varying financial problem is a great technical analysis tool as well as an important financial analysis tool. Similar technique could also be applied to other financial problems, such as asset allocation, risk management, option pricing, model calibration, etc., which at first requires a proper modification of them as realistic time-varying models, and then the definition of a suitable time-varying method for their solution.

However, the maximum portfolio dimension in the case of NN solvers, in contrast to BAS-based metaheuristic algorithms, was a study limitation, as these NN solvers cannot handle problems with very large dimensions well. On the one hand, the LP/QP ZNN solver based on the KKT conditions (ZNNKKTC) has better accuracy than the LP/QP ZNN solver based on the penalty function method (ZNNPFM), which use the Lagrange multiplier method. It is worth mentioning that the KKT technique to nonlinear programming generalizes the Lagrange multiplier method, which only supports equality constraints, by including inequality constraints. On the other hand, the state variable of ZNNPFM has a lower dimension than the state variable of ZNNKKTC, implying less time consumption and computational costs. According to numerical experiments in this dissertation's portfolio management problems, the LVI-PDNN solver outperforms the ZNNKKTC and ZNNPFM solvers in terms of

accuracy and time consumption. As a result, the LVI-PDNN solver is the most efficient of the three LP/QP NN solvers.

As a further research direction, we believe that all the computational approaches presented in this dissertation could be used as a basis for defining and solving a wide range of other financial optimization problems under appropriate modifications and considerations. Furthermore, we believe that the ZNN approach could be used as a basis for defining a new model that can also deal with noise perturbation which is inevitable in practical application scenarios, in the same line as in [142]. Note that the proposed ZNN models are under the linear activation function. As a result, it is possible to investigate the influence of nonlinear activation on the convergence speed and behavior of the proposed ZNN designs. In addition, the integration of fuzzy parameters in relevant upgrades of the ZNN design and BAS algorithm is an interesting topic of research. Last, a combination of BAS and ZNN, such as that described in [209, 210], could overcome the limitations of the presented LP/QP NN solvers and approach TV financial optimization problems in a gradient-free manner, contributing to the computational efficiency of the neural network.

A

Interpolation Algorithms

Algorithm 26 Algorithm for the step function of time.

Input: The marketed space $X = [x_1, x_2, \ldots, x_n]$, which is a matrix of n time series as column vectors of m prices, and the moving average's number of time periods $\tau \leq m - 1$, $\tau \in \mathbb{N}$.

1: Based on the normalized portfolio, create the structure array C comprises of the covariance matrices of and the matrix r consists of the expected return.
2: **function** g = sfots(data,t)
3: Set T as the floor price of t
4: **return** $g = data(T + 1, :)$
5: **end function**
6: Set $f_r = @(t)\text{sfots}(r, t)$
7: **function** h = sfotss(data,t)
8: Set T as the floor price of t
9: **return** $h = data\{T + 1\}$
10: **end function**
11: Set $f_C = @(t)\text{sfotss}(C, t)$

Output: The conversion of the covariance matrix and the expected return of n time series into time-varying step functions, $f_C(t)$ and $f_r(t)$, respectively.

Algorithm 27 Algorithm for the linear interpolation.

Input: The marketed space $X = [x_1, x_2, \ldots, x_n]$, which is a matrix of n time series as column vectors of m prices, and the moving average's number of time periods $\tau \leq m-1$, $\tau \in \mathbb{N}$.

Based on the normalized portfolio, create the structure array C comprises of the covariance matrices of and the matrix r consists of the expected return.
function g = linots(data,t)
  Set T as the floor price of t
  if $t = T$ **then**
    return g =data$(t+1,:)$
  else
    return g =data$(T+1,:)$ + (data$(T+2,:)$−data$(T+1,:)$)$(t-T)$
  end if
end function
Set $f_r = @(t)$linots$(r,t)'$ and $f_X = @(t)$linots$(X(\tau+1:\text{end},:),t)'$
function h = linotss(data,t)
  Set T as the floor price of t
  if $t = T$ **then**
    return h =data$\{t+1\}$
  else
    return h =data$\{T+1\}$ + (data$\{T+2\}$−data$\{T+1\}$)$(t-T)$
  end if
end function
Set $f_C = @(t)$linotss(C,t)

Output: The linear interpolation of the covariance matrix, the expected return and the close prices of n time series into time-variant functions, $f_C(t)$, $f_r(t)$ and $f_X(t)$, respectively.

Algorithm 28 Algorithm for the cubic Spline interpolation

Input: The marketed space $X = [x_1, x_2, \ldots, x_n]$, which is a matrix of n time series as column vectors of m prices, and the moving average's number of time periods $\tau \leq m-1$, $\tau \in \mathbb{N}$.

Based on the normalized portfolio, create the structure array C comprises of the covariance matrices of and the matrix r consists of the expected return.
function g = sp(data)
  Set data=data$'$, $[m,n]$ =size(data) and del =data$(:,2:\text{end})$−data$(:,1:\text{end}-1)$
  Set a =zeros(m,n) and $a(:,2:n-1) = 3(del(:,1:n-2) + del(:,2:n-1))$
  Set $a(:,1) = (5del(:,1) + del(:,2))/2$ and $a(:,n) = (del(:,n-2) + 5del(:,n-1))/2$
  Set b =ones$(n-2,1)$ and c =spdiags$([2,1,0;b,4b,b;0,1,2],[-1,0,1],n,n)$
  return $d = (a/c)'$
end function
function g = splinots(data,d,t)
  Set T as the floor price of t
  if $t >$size$(\text{data},1)-1$ **then**
    g =data(end,:)
  else if $t = T$ **then**
    g =data$(t+1,:)$

else
    Set del =data$(T+2,:)-$data$(T+1,:)$, $dzz = del - d(T+1,:)$ and $dzx = d(T+2,:) - del$
    Set $g = (dzx - dzz)(t-T)^3 + (2dzz - dzx)(t-T)^2 + d(T+1,:)(t-T)+$data$(T+1,:)$
end if
return g
end function
Set d =sp(r) and $f_r = @(t)$splinots$(r, d, t)'$
Set dd =sp$(X(\tau+1:\text{end},:))$ and $f_X = @(t)$splinots$(X(\tau+1:\text{end},:), dd, t)'$
function g = sps(data)
    Set n =size(data, 1) and m =length(data{1})
    for $i = 1 : m$ **do**
        Set w =cell2mat(cellfun(@$(x)(x(i,:))$,data,'UniformOutput',false))
        Set data= $w(1:n, 1:m)'$ and del =data$(:, 2:\text{end})-$data$(:, 1:\text{end}-1)$
        Set a =zeros(m, n) and $a(:, 2:n-1) = 3(del(:, 1:n-2) + del(:, 2:n-1))$
        Set $a(:,1) = (5del(:,1) + del(:,2))/2$ and $a(:,n) = (del(:,n-2) + 5del(:,n-1))/2$
        Set b =ones$(n-2, 1)$, c =spdiags$([2,1,0; b, 4b, b; 0,1,2], [-1,0,1], n, n)$ and $d = (a/c)'$
        for $j = 1 : n$ **do**
            Set data$\{j,1\}(i, 1:m) = d(j,:)$
        end for
    end for
    return data
end function
function h = splinotss(data,ds,t)
    Set T as the floor price of t
    if $t >$length(data)-1 **then**
        h =data{end}
    else if $t = T$ **then**
        h =data$\{t+1\}$
    else
        Set del =data$\{T+2\}-$data$\{T+1\}$, $dzz = del - ds\{T+1\}$ and $dzx = ds\{T+2\} - del$
        $h = (dzx - dzz)(t-T)^3 + (2dzz - dzx)(t-T)^2 + ds\{T+1\}(t-T)+$data$\{T+1\}$
    end if
    return h
end function
Set ds =sps(C) and $f_C = @(t)$splinotss(C, ds, t)

Output: The cubic spline interpolation of the covariance matrix and the expected return and the close prices of n time series into time-variant functions, $f_C(t)$, $f_r(t)$ and $f_X(t)$, respectively.

Algorithm 29 Algorithm for the piecewise cubic Hermite interpolation

Input: The marketed space $X = [x_1, x_2, \ldots, x_n]$, which is a matrix of n time series as column vectors of m prices, and the moving average's number of time periods $\tau \leq m-1$, $\tau \in \mathbb{N}$.

Based on the normalized portfolio, create the structure array C comprises of the covariance matrices of and the matrix r consists of the expected return.
function g = pchinots(data,t)
    Set T as the floor price of t.
    if $t = T$ **then**
        return $g = data(t+1,:)$
    else
        Set $u = size(data,2)$
        if $t >$length$(data) - 2$ **then**
            Set $del = data(T+1:T+2,:) - data(T:T+1,:)$ and $d =$zeros$(3,m)$
        else if $t < 1$ **then**
            Set $del = data(T+2:T+3,:) - data(T+1:T+2,:)$ and $d =$zeros$(2,m)$
        else
            Set $del = data(T+1:T+3,:) - data(T:T+2,:)$ and $d =$zeros$(3,m)$
        end if
        $[k1, k2] = find(\text{sign}(del(1:end-1,:)).\text{sign}(del(end-1:end,:)) > 0)$
        for $i = 1:k1$ **do**
            $d(k1(i)+1, k2(i)) = 2(\min(\|del(k1(i), k2(i))\|, \|del(k1(i)+1, k2(i))\|)$
$.\max(\|del(k1(i), k2(i))\|, \|del(k1(i)+1, k2(i))\|)./(del(k1(i), k2(i)) + del(k1(i)+1, k2(i)))$
        end for
        if $t < 1$ **then**
            Set $d(1,:) = (3del(1,:) - del(2,:))/2$
            for $i = 1:u$ **do**
                if sign$(d(1,i)) \neq$sign$(del(1,i))$ **then**
                    Set $d(1,i) = 0$
                else if sign$(del(1,i)) \neq$sign$(del(2,i))$ **and** $\|d(1,i)\| > \|3del(1,i)\|$ **then**
                    Set $d(1,i) = 3del(1,i)$
                end if
            end for
        end if
        if $t >$length$(data) - 2$ **then**
            Set $d(3,:) = (3del(2,:) - del(1,:))/2$
            for $i = 1:u$ **do**
                if sign$(d(3,i)) \neq$sign$(del(2,i))$ **then**
                    Set $d(3,i) = 0$
                else if sign$(del(2,i)) \neq$sign$(del(1,i))$ **and** $\|d(3,i)\| > \|3del(2,i)\|$ **then**
                    Set $d(3,i) = 3del(2,i)$
                end if
            end for
        end if
        Set $dzx = del(2,:) - d(end-1,:)$ and $dxz = d(end,:) - del(2,:)$

return $g = (dxz - dzx)(t-T)^3 + (2dzx - dxz)(t-T)^2 + d(end-1,:)(t-T) + data(T+1,:)$
end if
end function
Set $f_r = @(t)\text{pchinots}(X_r, t)$
function h = pchinotss(data,t)
Set T as the floor price of t
if $t = T$ **then**
return $h = data\{t+1\}$
else
Set u =length($data\{T\}$)
if $t >$length($data\{t\}$) -2 **then**
Set $del1 = data\{T+1\} - data\{T\}$, $del2 = data\{T+2\} - data\{T+1\}$ and d =zeros($3u, u$)
else if $t < 1$ **then**
Set $del2 = data\{T+2\} - data\{T+1\}$, $del3 = data\{T+3\} - data\{T+2\}$, d =zeros($2u, u$)
else
Set $del1 = data\{T+1\} - data\{T\}$ and $del2 = data\{T+2\} - data\{T+1\}$
Set $del3 = data\{T+3\} - data\{T+2\}$ and d =zeros($3u, u$)
end if
$[k1, k2] = \text{find}(\text{sign}(del(1:end-u,:)).\text{sign}(del(u+1:end,:)) > 0)$
for $i = 1:k1$ **do**
$d(k1(i) + u, k2(i)) = 2(\min(\|del(k1(i), k2(i))\|, \|del(k1(i)+u, k2(i))\|)$
$.\max(\|del(k1(i), k2(i))\|, \|del(k1(i)+u, k2(i))\|)./(del(k1(i), k2(i)) + del(k1(i)+u, k2(i)))$
end for
Set $v = del2$
if $t < 1$ **then**
Set $v = del1$
Set $d(1:u,:) = (3del1 - del2)/2$
for $i = 1:u$ **do**
for $j = 1:u$ **do**
if sign($d(i,j)$) $\neq$sign($del1(i,j)$) **then**
Set $d(i,j) = 0$
else if sign($del1(i,j)$) $\neq$sign($del2(i,j)$) **and** $\|d(i,j)\| > \|3del1(i,j)\|$ **then**
Set $d(i,j) = 3del1(i,j)$
end if
end for
end for
end if
if $t >$length($data$) -2 **then**
Set $d(2u+1:3u,:) = (3del2 - del1)/2$
for $i = 1:u$ **do**
for $j = 1:u$ **do**
if sign($d(2u+i,j)$) $\neq$sign($del2(i,j)$) **then**
Set $d(2u+i,j) = 0$
else if sign($del2(i,j)$) $\neq$sign($del1(i,j)$)**and** $\|d(2u+i,j)\| > \|3del2(i,j)\|$ **then**

Set $d(2u+i,j) = 3del2(i,j)$
end if
end for
end for
end if
Set $w = d(end-2u+1:end-u,:)$, $dzx = v - w$ and $dxz = d(end-u+1:end,:) - v$
return $h = (dxz - dzx)(t-T)^3 + (2dzx - dxz)(t-T)^2 + w(t-T) + data\{T+1\}$
end if
end function
Set $f_C = @(t)\text{pchinotss}(C,t)$

Output: The piecewise cubic Hermite interpolation of the covariance matrix and the expected return of n time series into time-varying functions, $f_C(t)$ and $f_r(t)$, respectively.

Bibliography

[1] A. Ghorbel and A. Trabelsi, "Energy portfolio risk management using time-varying extreme value copula methods," *Economic Modelling*, vol. 38, pp. 470–485, 2014.

[2] Z. Dai, "A Closer Look at the Minimum-Variance Portfolio Optimization Model," *Mathematical Problems in Engineering*, vol. 2019, pp. 1–8, Aug. 2019.

[3] V. N. Katsikis and S. D. Mourtas, "A heuristic process on the existence of positive bases with applications to minimum-cost portfolio insurance in C[a, b]," *Applied Mathematics and Computation*, vol. 349, pp. 221–244, 2019.

[4] V. N. Katsikis and S. D. Mourtas, "Optimal Portfolio Insurance under Nonlinear Transaction Costs," *Journal of Modeling and Optimization*, vol. 12, no. 2, pp. 117–124, 2020.

[5] V. N. Katsikis and S. D. Mourtas, "ORPIT: A Matlab Toolbox for Option Replication and Portfolio Insurance in Incomplete Markets," *Computational Economics*, vol. 56, pp. 711–721, Oct. 2019.

[6] K. Ye, P. Parpas, and B. Rustem, "Robust portfolio optimization: a conic programming approach," *Comp. Opt. and Appl.*, vol. 52, no. 2, pp. 463–481, 2012.

[7] H. Konno, K. Akishino, and R. Yamamoto, "Optimization of a long-short portfolio under nonconvex transaction cost," *Comp. Opt. and Appl.*, vol. 32, no. 1-2, pp. 115–132, 2005.

[8] W. Ogryczak and T. Sliwinski, "On solving the dual for portfolio selection by optimizing Conditional Value at Risk," *Comp. Opt. and Appl.*, vol. 50, no. 3, pp. 591–595, 2011.

[9] K. Deb, *Optimization for Engineering Design: Algorithms and Examples*. PHI, second ed., July 2013.

[10] E. Mezura-Montes and C. A. C. Coello, "Constraint-handling in nature-inspired numerical optimization: Past, present and future," *Swarm and evolutionary computation*, vol. 1, no. 4, pp. 173–194, 2011.

[11] D. Powell and M. M. Skolnick, "Using genetic algorithms in engineering design optimization with non-linear constraints," in *Proceedings of the fifth international conference on genetic algorithms* (S. Forrest, ed.), (University of Illinois at Urbana-Champaign. San Mateo, CA, USA), Morgan Kaufmann, 1993.

[12] S. Koziel and Z. Michalewicz, "Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization.," *Evolutionary computation*, vol. 7, pp. 19–44, 1999.

[13] X. S. Yang, *Nature-inspired optimization algorithms*. Elsevier insights, Amsterdam: Elsevier, 1st edition ed., 2014. Literaturangaben.

[14] K. L. Du and M. N. S. Swamy, *Search and Optimization by Metaheuristics*. Birkhäuser Basel, 2016.

[15] S. Koziel and X. S. Yang, eds., *Computational Optimization, Methods and Algorithms*, vol. 356 of *Studies in Computational Intelligence*. Springer-Verlag Berlin Heidelberg, 2011.

[16] X. S. Yang, ed., *Nature-Inspired Algorithms and Applied Optimization*, vol. 744 of *Studies in Computational Intelligence*. Springer International Publishing, 2018.

[17] E. Süli and D. F. Mayers, *An Introduction to Numerical Analysis*. Cambridge University Press, 2003.

[18] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 International Conference on Neural Networks*, (Perth, WA, Australia), 1995.

[19] R. V. Belavkin, "Optimal measures and Markov transition kernels," *Journal of Global Optimization*, vol. 55, no. 2, pp. 387–416, 2013.

[20] J. Holland, *Adaptation in Natural and Artificial Systems*. Bradford Books, reprint ed., 1992.

[21] R. V. Belavkin, "On evolution of an information dynamic system and its generating operator," *Optimization Letters*, vol. 6, no. 5, pp. 827–840, 2012.

[22] R. Hooke and T. A. Jeeves, ""Direct search" solution of numerical and statistical problems," *Journal of the Association for Computing Machinery*, vol. 8, pp. 212–229, 1961.

[23] X. S. Yang, S. Deb, and S. Fong, "Accelerated Particle Swarm Optimization and Support Vector Machine for Business Optimization and Applications," in *Networked Digital Technologies. NDT 2011* (S. Fong, ed.), vol. 136 of *Communications in Computer and Information Science*, pp. 53–66, 2011.

[24] R. Storn, "On the usage of differential evolution for function optimization," in *Proceedings of North American Fuzzy Information Processing*, (Berkeley, CA, USA), 1996.

[25] R. Storn and K. Price, "Differential Evolution - A Simple and Efficient Heuristic for global Optimization over Continuous Spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[26] K. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*. Natural Computing Series, Springer-Verlag Berlin Heidelberg, 2005.

[27] X. S. Yang, *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008.

[28] X. S. Yang, "Firefly algorithm, stochastic test functions and design optimisation," *International Journal of Bio-Inspired Computation*, vol. 2, no. 2, pp. 78–84, 2010.

[29] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671–680, 1983.

[30] X. S. Yang and S. Deb, "Cuckoo Search via Lévy flights," in *Proceedings of world congress on nature & biologically inspired computing (NaBIC 2009)*, (Coimbatore, India), pp. 210–214, IEEE, 2009.

[31] I. Pavlyukevich, "Lévy flights, non-local search and simulated annealing," *Journal of Computational Physics*, vol. 226, no. 2, pp. 1830–1844, 2007.

[32] F. Wang, X. S. He, Y. Wang, and S. M. Yang, "Markov Model and Convergence Analysis Based on Cuckoo Search Algorithm," *Computer Engineering*, vol. 38, no. 11, pp. 180–182,185, 2012.

[33] X. S. Yang, *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, vol. 284 of *Studies in Computational Intelligence*, ch. A New Metaheuristic Bat-Inspired Algorithm. Springer, Berlin, Heidelberg, 2010.

[34] V. Patel, A. Tiwari, and A. Patel, "A comprehensive survey on hybridization of artificial bee colony with particle swarm optimization algorithm and ABC applications to data clustering," in *Proceedings of the International Conference on Informatics and Analytics*, ICIA-16, (New York, NY, USA), p. 1–9, Association for Computing Machinery, 8 2016.

[35] M. Sebai, E. Fatnassi, and L. Rejeb, "A honeybee mating optimization algorithm for solving the static bike rebalancing problem," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, (New York, NY, USA), p. 77–78, Association for Computing Machinery, 7 2019.

[36] D. Karaboga and B. Basturk, "On the performance of artificial bee colony (ABC) algorithm," *Applied soft computing*, vol. 8, no. 1, pp. 687–697, 2008.

[37] X. Zhang, Y. Zhang, and Y. Deng, "An improved bio-inspired algorithm for the directed shortest path problem," *Bioinspiration and Biomimetics*, vol. 9, p. 046016, Dec. 2014.

[38] L. Liu, Y. Song, H. Zhang, H. Ma, and A. V. Vasilakos, "Physarum Optimization: A Biology-Inspired Algorithm for the Steiner Tree Problem in Networks," *IEEE Transactions on Computers*, vol. 64, pp. 818–831, 2015.

[39] X. Jiang and S. Li, "BAS: Beetle Antennae Search Algorithm for Optimization Problems," *arXiv preprint*, vol. abs/1710.10724, 2017.

[40] Q. Wu, X. Shen, Y. Jin, Z. Chen, S. Li, A. H. Khan, and D. Chen, "Intelligent Beetle Antennae Search for UAV Sensing and Avoidance of Obstacles," *Sensors*, vol. 19, p. 1758, 2019.

[41] A. T. Khan, X. Cao, S. Li, B. Hu, and V. N. Katsikis, "Quantum beetle antennae search: A novel technique for the constrained portfolio optimization problem," *SCIENCE CHINA Information Sciences*, 2020.

[42] A. H. Khan, X. Cao, S. Li, V. N. Katsikis, and L. Liao, "BAS-ADAM: an ADAM based approach to improve the performance of beetle antennae search optimizer," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 2, pp. 461–471, 2020.

[43] M. A. Medvedeva, V. N. Katsikis, S. D. Mourtas, and T. E. Simos, "Randomized time-varying knapsack problems via binary beetle antennae search algorithm: Emphasis on applications in portfolio insurance," *Math Meth Appl Sci*, vol. 4, no. 2, pp. 2002–2012, 2020.

[44] A. H. Khan, X. Cao, V. N. Katsikis, P. Stanimirovic, I. Brajevic, S. Li, S. Kadry, and Y. Nam, "Optimal Portfolio Management for Engineering Problems Using Nonconvex Cardinality Constraint: A Computing Perspective," *IEEE Access*, pp. 1–1, 2020.

[45] A. E. Eiben and S. K. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 19–31, 2011.

[46] D. L. Poole and A. K. Mackworth, *Artificial Intelligence: Foundations of Computational Agents*. Cambridge UniversityPress, second ed., 2017.

[47] S. Marsland, *Machine Learning: An Algorithmic Perspective*. Chapman and Hall/CRC, second ed., 2014.

[48] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*. Morgan Kaufmann, third ed., 2011.

[49] F. Peng, D. Schuurmans, and S. Wang, "Augmenting Naive Bayes Classifiers with Statistical Language Models," *Information Retrieval*, vol. 7, no. 3-4, pp. 317–345, 2004.

[50] R. Chen and C. Chen, *Artificial intelligence: An introduction for the inquisitive reader*, ch. Support vector machines, pp. 223–229. Chapman and Hall/CRC, 2022.

[51] L. Jiang, Z. Cai, D. Wang, and S. Jiang, "Survey of improving K-nearest-neighbor for classification," in *Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007)*, pp. 679–683, 2007.

[52] D. Maulud and A. Mohsin Abdulazeez, "A review on linear regression comprehensive in machine learning," *Journal of Applied Science and Technology Trends*, vol. 1, pp. 140–147, 2020.

[53] T. Rymarczyk, E. Kozłowski, G. Kłosowski, and K. Niderla, "Logistic regression for machine learning in process tomography," *Sensors*, vol. 19, no. 15, p. 3400, 2019.

[54] T. S. Madhulatha, "An overview on clustering methods," *IOSR Journal of Engineering*, vol. 2, no. 4, pp. 719–725, 2012.

[55] R. Zafarani, M. A. Abbasi, and H. Liu, *Social Media Mining*. Cambridge University Press, 2014.

[56] O. Chapelle, B. Schölkopf, and A. Zien, *Semi-Supervised Learning*. Adaptive Computation and Machine Learning series, The MIT Press, 2006.

[57] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, second ed., 2016.

[58] D. Raví, C. Wong, F. Deligianni, M. Berthelot, J. Andreu-Perez, B. Lo, and G.-Z. Yang, "Deep Learning for Health Informatics," *IEEE Journal of Biomedical and Health Informatics*, vol. 21, pp. 4–21, 2017.

[59] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011* (G. J. Gordon, D. B. Dunson, and M. Dudík, eds.), vol. 15 of *JMLR Proceedings*, pp. 315–323, JMLR.org, 2011.

[60] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, 1998.

[61] A. Krizhevsky, S. Ilya, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, NIPS 2012, 2012.

[62] P. Wlodarczak, *Machine Learning and its Applications*. CRC Press, first ed., 2020.

[63] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 79, no. 8, pp. 2554–2558, 1982.

[64] E. Siegel, *Predictive Analytics: The Power to Predict Who Will Click, Buy, Lie, or Die*. Wiley, first ed., 2013.

[65] A. Fischer and C. Igel, "Bounding the bias of contrastive divergence learning," *Neural computation*, vol. 23, pp. 664–673, 2011.

[66] J. Li, J. Cheng, J. Shi, and F. Huang, "Brief Introduction of Back Propagation (BP) Neural Network Algorithm and its Improvement," in *Advances in Computer Science and Information Engineering*, vol. 169 of *Advances in Intelligent and Soft Computing*, pp. 553–558, Springer, Berlin, Heidelberg, 2012.

[67] T. Han, Y. Lu, S. Zhu, and Y. N. Wu, "Alternating Back-Propagation for Generator Network," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA* (S. P. Singh and S. Markovitch, eds.), pp. 1976–1984, AAAI Press, 2017.

[68] Y. Zhang, D. Chen, and C. Ye, *Deep Neural Networks: WASD Neuronet Models, Algorithms, and Applications*. CRC Press, 2019.

[69] S. Mourtas and V. Katsikis, "Exploiting the Black-Litterman framework through error-correction neural networks," *Neurocomputing*, vol. 498, pp. 43–58, 08 2022.

[70] T. E. Simos, S. D. Mourtas, and V. N. Katsikis, "Time-varying Black–Litterman portfolio optimization using a bio-inspired approach and neuronets," *Applied Soft Computing*, vol. 112, p. 107767, 2021.

[71] Y. Zhang, X. Yu, L. Xiao, W. Li, Z. Fan, and W. Zhang, "Weights and structure determination of articial neuronets," in *Self-Organization: Theories and Methods*, New York, NY, USA: Nova Science, 2013.

[72] Y. Zhang, Y. Yin, D. Guo, X. Yu, and L. Xiao, "Cross-validation based weights and structure determination of Chebyshev-polynomial neural networks for pattern classification," *Pattern Recognit.*, vol. 47, no. 10, pp. 3414–3428, 2014.

[73] Y. Zhang, D. Guo, Z. Luo, K. Zhai, and H. Tan, "CP-activated WASD neuronet approach to Asian population prediction with abundant experimental verification," *Neurocomputing*, vol. 198, no. 198, pp. 48–57, 2016.

[74] T. Zeng, Y. Zhang, Z. Li, B. Qiu, and C. Ye, "Predictions of USA Presidential Parties From 2021 to 2037 Using Historical Data Through Square Wave-Activated WASD Neural Network," *IEEE Access*, vol. 8, pp. 56630–56640, 2020.

[75] Y.-J. Liu, S. Tong, D.-J. Li, and Y. Gao, "Fuzzy adaptive control with state observer for a class of nonlinear discrete-time systems with input constraint," *IEEE Transactions on Fuzzy Systems*, vol. 24, no. 5, pp. 1147–1158, 2015.

[76] H. Wang, W. Liu, J. Qiu, and P. X. Liu, "Adaptive fuzzy decentralized control for a class of strong interconnected nonlinear systems with unmodeled dynamics," *Neurocomputing*, vol. 26, pp. 836–846, Nov. 2018.

[77] P. S. Stanimirović, I. S. Živković, and Y. Wei, "Recurrent neural network approach based on the integral representation of the Drazin inverse," *Neural Computation*, vol. 27, pp. 2107–2131, Oct. 2015.

[78] I. S. Živković, P. S. Stanimirović, and Y. Wei, "Recurrent Neural Network for Computing Outer Inverse," *Neural Computation*, vol. 28, pp. 970–998, May 2016.

[79] X. Luo, M. Zhou, S. Li, Y. Xia, Z. You, Q. Zhu, and H. Leung, "Incorporation of efficient second-order solvers into latent factor models for accurate prediction of missing QoS data," *IEEE Transactions on Cybernetics*, vol. 48, pp. 1216–1228, Apr. 2018.

[80] X. Luo, M. Zhou, S. Li, Y. Xia, Z. You, Q. Zhu, and H. Leung, "An efficient second-order approach to factorize sparse matrices in recommender systems," *IEEE Transactions on Industrial Informatics*, vol. 11, pp. 946–956, Aug. 2015.

[81] L. Jin, Y. Zhang, and B. Qiu, "Neural network-based discrete-time Z-type model of high accuracy in noisy environments for solving dynamic system of linear equations," *Neural Computing and Applications*, vol. 29, pp. 1217–1232, June 2018.

[82] J. Wang and G. Wu, "Recurrent neural networks for synthesizing linear control systems via pole placement," *International Journal of Systems Science*, vol. 26, pp. 2369–2382, May 1995.

[83] Y. Zhang and J. Wang, "Recurrent neural networks for nonlinear output regulation," *Automatica*, vol. 37, pp. 1161–1173, Aug. 2001.

[84] Y. Zhang, K. Chen, and H. Tan, "Performance analysis of gradient neural network exploited for online time-varying matrix inversion," *IEEE Transactions on Automatic Control*, vol. 54, pp. 1940–1945, Aug. 2009.

[85] Y. Zhang, Y. Yang, and G. Ruan, "Performance analysis of gradient neural network exploited for online time-varying quadratic minimization and equality-constrained quadratic programming," *Neurocomputing*, vol. 74, pp. 1710–1719, May 2011.

[86] Y. Zhang, D. Jiang, and J. Wang, "A recurrent neural network for solving Sylvester equation with time-varying coefficients," *IEEE Transactions on Neural Networks*, vol. 13, pp. 1053–1063, Sept. 2002.

[87] Y. Zhang and S. S. Ge, "Design and analysis of a general recurrent neural network model for time-varying matrix inversion," *IEEE Transactions on Neural Networks*, vol. 16, pp. 1477–1490, Nov. 2005.

[88] D. Guo, C. Yi, and Y. Zhang, "Zhang neural network versus gradient-based neural network for time-varying linear matrix equation solving," *Neurocomputing*, vol. 74, pp. 3708–3712, Oct. 2011.

[89] Y. Zhang, L. Xiao, G. Ruan, and Z. Li, "Continuous and discrete time Zhang dynamics for time-varying 4th root finding," *Numerical Algorithms*, vol. 57, p. 35, May 2011.

[90] Y. Zhang, Y. Wang, L. Jin, B. Mu, and H. Zheng, "Different ZFs leading to various ZNN models illustrated via online solution of time-varying underdetermined systems of linear equations with robotic application," in *Advances in Neural Networks – ISNN 2013*, Springer, Jan. 2013.

[91] Y. Zhang, L. Jin, and Z. Ke, "Superior performance of using hyperbolic sine activation functions in ZNN illustrated via time-varying matrix square roots finding," *Computer Science and Information Systems*, vol. 9, no. 4, pp. 1603–1625, 2012.

[92] Y. Zhang, C. Yi, D. Guo, and J. Zheng, "Comparison on Zhang neural dynamics and gradient-based neural dynamics for online solution of nonlinear time-varying equation," *Neural Computing and Applications*, vol. 20, no. 1, pp. 1–7, 2011.

[93] D. Guo and Y. Zhang, "Novel recurrent neural network for time-varying problems solving [research frontier]," *IEEE Computational Intelligence Magazine*, vol. 7, pp. 61–65, Nov. 2012.

[94] L. Xiao and Y. Zhang, "Zhang neural network versus gradient neural network for solving time-varying linear inequalities," *IEEE Transactions on Neural Networks*, vol. 22, pp. 1676–1684, Oct. 2011.

[95] Y. Zhang and Z. Li, "Zhang neural network for online solution of time-varying convex quadratic program subject to time-varying linear-equality constraints," *Physics Letters A*, vol. 373, pp. 1639–1643, Apr. 2009.

[96] Y. Zhang, Z. Li, and K. Li, "Complex-valued zhang neural network for online complex-valued time-varying matrix inversion," *Applied Mathematics and Computation*, vol. 217, pp. 10066–10073, Aug. 2011.

[97] L. Jin, Y. Zhang, and S. Li, "Integration-enhanced Zhang neural network for real-time-varying matrix inversion in the presence of various kinds of noises," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, pp. 2615–2627, Dec. 2016.

[98] L. Jin, Y. Zhang, S. Li, and Y. Zhang, "Modified ZNN for time-varying quadratic programming with inherent tolerance to noises and its application to kinematic redundancy resolution of robot manipulators," *IEEE Transactions on Industrial Electronics*, vol. 63, pp. 6978–6988, Nov. 2016.

[99] M. Liu, B. Liao, L. Ding, and L. Xiao, "Performance analyses of recurrent neural network models exploited for online time-varying nonlinear optimization," *Computer Science and Information Systems*, vol. 13, no. 2, pp. 691–705, 2016.

[100] Y. Yang and Y. Zhang, "Superior robustness of power-sum activation functions in Zhang neural networks for time-varying quadratic programs perturbed with large implementation errors," *Neural Computing and Applications*, vol. 22, no. 1, pp. 175–185, 2013.

[101] S. Li, S. Chen, and B. Liu, "Accelerating a recurrent neural network to finite-time convergence for solving time-varying sylvester equation by using a sign-bi-power activation function," *Neural Processing Letters*, vol. 37, no. 2, pp. 189–205, 2013.

[102] S. Li, Y. Li, and Z. Wang, "A class of finite-time dual neural networks for solving quadratic programming problems and its k-winners-take-all application," *Neural Networks*, vol. 39, pp. 27–39, Mar. 2013.

[103] S. P. Bhat and D. S. Bernstein, "Finite-time stability of continuous autonomous systems," *SIAM Journal on Control and Optimization*, vol. 38, no. 3, pp. 751–766, 2000.

[104] P. Miao, Y. Shen, and X. Xia, "Finite time dual neural networks with a tunable activation function for solving quadratic programming problems and its application," *Neurocomputing*, vol. 143, pp. 80–89, Nov. 2014.

[105] B. Liao and Y. Zhang, "From different ZFs to different ZNN models accelerated via Li activation functions to finite-time convergence for time-varying matrix pseudoinversion," *Neurocomputing*, vol. 133, pp. 512–522, June 2014.

[106] L. Xiao and B. Liao, "A convergence-accelerated Zhang neural network and its solution application to lyapunov equation," *Neurocomputing*, vol. 193, pp. 213–218, June 2016.

[107] L. Xiao, "A nonlinearly-activated neurodynamic model and its finite-time solution to equality-constrained quadratic optimization with nonstationary coefficients," *Applied Soft Computing*, vol. 40, pp. 252–259, Mar. 2016.

[108] L. Xiao, "A finite-time convergent neural dynamics for online solution of time-varying linear complex matrix equation," *Neurocomputing*, vol. 167, pp. 254–259, Nov. 2015.

[109] S. Qiao, X.-Z. Wang, and Y. Wei, "Two finite-time convergent Zhang neural network models for time-varying complex matrix Drazin inverse," *Linear Algebra and its Applications*, vol. 542, pp. 101–117, Apr. 2018.

[110] L. Xiao and Y. Zhang, "From different Zhang functions to various ZNN models accelerated to finite-time convergence for time-varying linear matrix equation," *Neural Processing Letters*, vol. 39, no. 3, pp. 309–326, 2014.

[111] S. Li and Y. Li, "Nonlinearly activated neural network for solving time-varying complex Sylvester equation," *IEEE Transactions on Cybernetics*, vol. 44, pp. 1397–1407, Aug. 2014.

[112] B. Liao and Y. Zhang, "Different complex ZFs leading to different complex ZNN models for time-varying complex generalized inverse matrices," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, pp. 1621–1631, Sept. 2014.

[113] B. Liao, L. Xiao, J. Jin, L. Ding, and M. Liu, "Novel complex-valued neural network for dynamic complex-valued matrix inversion," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 20, no. 1, pp. 132–138, 2016.

[114] L. Xiao, W. Meng, R. Lu, X. Yang, B. Liao, and L. Ding, "A fully complex-valued neural network for rapid solution of complex-valued systems of linear equations," *Advances in Neural Networks – ISNN 2015*, Jan. 2015.

[115] K. Chen and C. Yi, "Robustness analysis of a hybrid of recursive neural dynamics for online matrix inversion," *Applied Mathematics and Computation*, vol. 273, pp. 969–975, Jan. 2016.

[116] Y. Zhang, "Zeroing Dynamics, Gradient Dynamics, and Newton Iterations," 2015. Description based upon print version of record.

[117] Y. Zhang, Y. Zhang, D. Chen, Z. Xiao, and X. Yan, "From Davidenko method to Zhang dynamics for nonlinear equation systems solving," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, pp. 2817–2830, Nov. 2017.

[118] Y. Zhang, B. Cai, M. Liang, and W. Ma, "On the variable step-size of discrete-time Zhang neural network and Newton iteration for constant matrix inversion," in *Proc. Second Int. Symp. Intelligent Information Technology Application*, vol. 1, pp. 34–38, Dec. 2008.

[119] Y. Zhang, W. Ma, and B. Cai, "From Zhang Neural Network to Newton Iteration for Matrix Inversion," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, pp. 1405–1415, July 2009.

[120] M. Mao, J. Li, L. Jin, S. Li, and Y. Zhang, "Enhanced discrete-time Zhang neural network for time-variant matrix inversion in the presence of bias noises," *Neurocomputing*, vol. 207, pp. 220–230, Sept. 2016.

[121] Y. Zhang, L. Jin, D. Guo, Y. Yin, and Y. Chou, "Taylor-type 1-step-ahead numerical differentiation rule for first-order derivative approximation and ZNN discretization," *Journal of Computational and Applied Mathematics*, vol. 273, pp. 29–40, Jan. 2015.

[122] D. Guo and Y. Zhang, "Zhang neural network, Getz–Marsden dynamic system, and discrete-time algorithms for time-varying matrix inversion with application to robots' kinematic control," *Neurocomputing*, vol. 97, pp. 22–32, Nov. 2012.

[123] D. Guo, Z. Nie, and L. Yan, "Novel discrete-time Zhang neural network for time-varying matrix inversion," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, pp. 2301–2310, Aug. 2017.

[124] Y. Zhang and S. S. Ge, "A primal neural network for solving nonlinear equations and inequalities," in *Proc. IEEE Conf. Cybernetics and Intelligent Systems*, vol. 2, pp. 1232–1237, Dec. 2004.

[125] Y. Zhang, "On the LVI-based primal-dual neural network for solving online linear and quadratic programming problems," in *Proc. 2005, American Control Conf*, pp. 1351–1356 vol. 2, June 2005.

[126] Y. Zhang, Y. Wang, D. Chen, C. Peng, and Q. Xie, "Neurodynamic solvers robotic applications and solution nonuniqueness of linear programming," in *Linear Programming: Theory, Algorithms and Applicant*, pp. 27–100, Nova Science Publishers, 2014.

[127] N. Zhong, Q. Huang, S. Yang, F. Ouyang, and Z. Zhang, "A varying-parameter recurrent neural network combined with penalty function for solving constrained multi-criteria optimization scheme for redundant robot manipulators," *IEEE Access*, vol. 9, pp. 50810–50818, 2021.

[128] Z. Zhang, S. Yang, and L. Zheng, "A penalty strategy combined varying-parameter recurrent neural network for solving time-varying multi-type constrained quadratic programming problems.," *IEEE transactions on neural networks and learning systems*, vol. 32, pp. 2993–3004, July 2021.

[129] L. Jin and S. Li, "Nonconvex function activated zeroing neural network models for dynamic quadratic programming subject to equality and inequality constraints," *Neurocomputing*, vol. 267, pp. 107–113, 2017.

[130] J. Li, Y. Shi, and H. Xuan, "Unified model solving nine types of time-varying problems in the frame of zeroing neural network," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, pp. 1896–1905, 2021.

[131] S. Boyd and L. Vandenberghe, *Convex optimization problems.* Cambridge University Press, New York, 2004.

[132] Y. Zhang and W. E. Leithead, "Exploiting Hessian matrix and trust-region algorithm in hyperparameters estimation of Gaussian process," *Applied Mathematics and Computation*, vol. 171, no. 2, pp. 1264–1281, 2005.

[133] K. Lqbal and Y. Pai, "Predicted region of stability for balance recovery: motion at the knee joint can improve termination of forward movement.," *Journal of biomechanics*, vol. 33, pp. 1619–1627, Dec. 2000.

[134] D. Anguita, A. Boni, and S. Ridella, "A digital architecture for support vector machines: theory, algorithm, and FPGA implementation," *IEEE Transactions on Neural Networks*, vol. 14, pp. 993–1009, Sept. 2003.

[135] Y. Zhang, W. Ma, X. Li, H. Tan, and K. Chen, "MATLAB Simulink modeling and simulation of LVI-based primal-dual neural network for solving linear and quadratic programs," *Neurocomputing*, vol. 72, no. 7-9, pp. 1679–1687, 2009.

[136] Y. Zhang, J. Wang, and Y. Xia, "A dual neural network for redundancy resolution of kinematically redundant manipulators subject to joint limits and joint velocity limits," *IEEE Transactions on Neural Networks*, vol. 14, pp. 658–667, May 2003.

[137] P. S. Stanimirović, V. N. Katsikis, and S. Li, "Hybrid GNN-ZNN models for solving linear matrix equations," *Neurocomputing*, vol. 316, pp. 124–134, 2018.

[138] M. D. Petković, P. S. Stanimirović, and V. N. Katsikis, "Modified discrete iterations for computing the inverse and pseudoinverse of the time-varying matrix," *Neurocomputing*, vol. 289, pp. 155–165, 2018.

[139] P. S. Stanimirović, V. N. Katsikis, and S. Li, "Higher-Order ZNN Dynamics," *Neural Processing Letters*, pp. 1–25, 2019.

[140] H. Ma, N. Li, P. S. Stanimirović, and V. N. Katsikis, "Perturbation theory for Moore–Penrose inverse of tensor via Einstein product," *Computational and Applied Mathematics*, vol. 38, no. 3, p. 111, 2019.

[141] P. S. Stanimirović, V. N. Katsikis, Z. Zhang, S. Li, J. Chen, and M. Zhou, "Varying-parameter Zhang neural network for approximating some expressions involving outer inverses," *Optimization Methods and Software*, pp. 1–27, 2019.

[142] P. S. Stanimirović, V. N. Katsikis, and S. Li, "Integration enhanced and noise tolerant ZNN for computing various expressions involving outer inverses," *Neurocomputing*, vol. 329, pp. 129–143, 2019.

[143] Y. Zhang, S. S. Ge, and T. H. Lee, "A unified quadratic-programming-based dynamical system approach to joint torque optimization of physically constrained redundant manipulators," *Part B (Cybernetics) IEEE Transactions on Systems, Man, and Cybernetics*, vol. 34, pp. 2126–2132, Oct. 2004.

[144] C. Tsitouras, "Neural networks with multidimensional transfer functions," *IEEE Transactions on Neural Networks*, vol. 13, pp. 222–228, Jan. 2002.

[145] C. Tsitouras, "Using Neural Networks for the Derivation of Runge–Kutta–Nyström Pairs," in *AIP Conference Proceedings*, vol. 1048, pp. 1034–1036, AIP, 2008.

[146] C. Tsitouras and I. T. Famelis, "Using neural networks for the derivation of Runge–Kutta–Nyström pairs for integration of orbits," *New Astronomy*, vol. 17, no. 4, pp. 469–473, 2012.

[147] A. Alexandridis, I. T. Famelis, and C. Tsitouras, "Long-term time-series prediction using radial basis function neural networks," 2015.

[148] S. S. Ge, Y. Zhang, and T. H. Lee, "An acceleration-based weighting scheme for minimum-effort inverse kinematics of redundant manipulators," in *Proc. IEEE Int. Symp. Intelligent Control*, pp. 275–280, Sept. 2004.

[149] Y. Zhang, F. Wu, Z. Xiao, Z. Li, and B. Cai, "Performance analysis of LVI-based PDNN applied to real-time solution of time-varying quadratic programming," in *2014 International Joint Conference on Neural Networks, IJCNN 2014, Beijing, China, July 6-11, 2014*, pp. 3155–3160, IEEE, 2014.

[150] V. N. Katsikis, S. D. Mourtas, P. S. Stanimirović, S. Li, and X. Cao, "Time-varying minimum-cost portfolio insurance problem via an adaptive fuzzy-power LVI-PDNN," *Applied Mathematics and Computation*, vol. 441, p. 127700, 2023.

[151] V. N. Kovalnogov, R. V. Fedorov, D. A. Generalov, A. V. Chukalin, V. N. Katsikis, S. D. Mourtas, and T. E. Simos, "Portfolio insurance through error-correction neural networks," *Mathematics*, vol. 10, no. 18, p. 3335, 2022.

[152] V. N. Katsikis, S. D. Mourtas, P. S. Stanimirović, S. Li, and X. Cao, "Time-varying mean-variance portfolio selection problem solving via LVI-PDNN," *Computers & Operations Research*, vol. 138, p. 105582, 2022.

[153] S. D. Mourtas and C. Kasimis, "Exploiting mean-variance portfolio optimization problems through zeroing neural networks," *Mathematics*, vol. 10, p. 3079, 08 2022.

[154] V. N. Katsikis and S. D. Mourtas, *Computational Management*, vol. 18 of *Modeling and Optimization in Science and Technologies*, ch. Portfolio Insurance and Intelligent Algorithms, pp. 305–323. Springer, Cham., 2021.

[155] V. N. Katsikis, S. D. Mourtas, P. S. Stanimirović, S. Li, and X. Cao, "Time-varying minimum-cost portfolio insurance under transaction costs problem via beetle antennae search algorithm (BAS)," *Applied Mathematics and Computation*, vol. 385, p. 125453, 2020.

[156] V. N. Katsikis, S. D. Mourtas, P. S. Stanimirović, S. Li, and X. Cao, "Time-varying mean-variance portfolio selection under transaction costs and cardinality constraint problem via beetle antennae search algorithm (BAS)," *SN Operations Research Forum*, vol. 2, no. 18, 2021.

[157] S. D. Mourtas and V. N. Katsikis, "V-shaped BAS: Applications on large portfolios selection problem," *Computational Economics*, vol. 60, pp. 1353–1373, 2021.

[158] V. N. Katsikis and S. D. Mourtas, "Diversification of time-varying tangency portfolio under nonlinear constraints through semi-integer beetle antennae search algorithm," *AppliedMath*, vol. 1, no. 1, pp. 63–73, 2021.

[159] J. Annaert, M. D. Ceuster, and J. Vandenbroucke, "Mind the Floor: Enhance Portfolio Insurance without Borrowing," *The Journal of Investing*, vol. 28, pp. 39–50, June 2019.

[160] K. Matsumoto, "Portfolio Insurance with Liquidity Risk," *Asia-Pacific Financial Markets*, vol. 14, p. 363, May 2008.

[161] C. D. Aliprantis, D. J. Brown, and J. Werner, "Minimum-cost portfolio insurance," *Journal of Economic Dynamics & Control*, vol. 24, pp. 1703–1719, 2000.

[162] V. N. Katsikis, "Computational methods in portfolio insurance," *Applied Mathematics and Computation*, vol. 189, no. 1, pp. 9–22, 2007.

[163] V. N. Katsikis, "Computational methods in lattice-subspaces of C[a,b] with applications in portfolio insurance," *Applied Mathematics and Computation*, vol. 200, pp. 204–219, 2008.

[164] V. N. Katsikis, *Matlab - Modelling, Programming and Simulations*, ch. Computational and mathematical methods in portfolio insurance - A MATLAB-based approach. IntechOpen, Oct. 2010.

[165] S. Wu, L. He, M. Zou, J. Li, and Y. Zhang, "Time-varying linear programming via LVI-PDNN with numerical examples," in *Proc. IEEE 13th Int. Conf. Signal Processing (ICSP)*, pp. 326–331, Nov. 2016.

[166] Y. Zhang and D. Guo, "Linear programming versus quadratic programming in robots' repetitive redundancy resolution: A chattering phenomenon investigation," in *Proc. 4th IEEE Conf. Industrial Electronics and Applications*, pp. 2822–2827, May 2009.

[167] H. Markowitz, "Portfolio selection," *The journal of finance*, vol. 7, no. 1, pp. 77–91, 1952.

[168] I. Kulali, "Portfolio Optimization Analysis with Markowitz Quadratic Mean-Variance Model," *European Journal of Business and Management*, vol. 8, no. 7, pp. 73–79, 2016.

[169] T. R. Bielecki, H. Jin, S. R. Pliska, and X. Y. Zhou, "Continuous-Time Mean–Variance Portfolio Selection with Bankruptcy Prohibition," *Mathematical Finance*, vol. 15, pp. 213–244, Apr. 2005.

[170] G. Cornuejols and R. Tütüncü, *Optimization Methods in Finance*. Cambridge: Cambridge University Press, 2006.

[171] T. Draviam and T. Chellathurai, "Generalized Markowitz mean–variance principles for multi–period portfolio–selection problems," *Proc. R. Soc. Lond. A.*, vol. 458, pp. 2571–2607, 2002.

[172] H. M. Markowitz, "The general mean-variance portfolio selection problem," *Philosophical Transactions of the Royal Society of London. Series A: Physical and Engineering Sciences*, vol. 347, no. 1684, pp. 543–549, 1994.

[173] V. Zakamulin, *Market Timing with Moving Averages: The Anatomy and Performance of Trading Rules*. Springer, 2017.

[174] H. M. Markowitz, *Portfolio Selection: Efficient Diversification of Investments*. Cowles Foundation Monograph: No. 16, Yale University Press, 1959.

[175] Y. Zhang and C. Yi, *Zhang neural networks and neural-dynamic method*. Nova Science Publishers, Inc., 2011.

[176] Z. Zhang and Y. Zhang, "Acceleration-Level Cyclic-Motion Generation of Constrained Redundant Robots Tracking Different Paths," *Part B (Cybernetics) IEEE Transactions on Systems, Man, and Cybernetics*, vol. 42, pp. 1257–1269, Aug. 2012.

[177] Y. Zhang, "Towards Piecewise-Linear Primal Neural Networks for Optimization and Redundant Robotics," in *Proc. Sensing and Control 2006 IEEE Int. Conf. Networking*, pp. 374–379, Apr. 2006.

[178] F. Black and R. B. Litterman, "Global asset allocation with equities, bonds, and currencies," *Fixed Income Research*, vol. 2, pp. 1–44, Oct. 1991.

[179] F. Black and R. Litterman, "Global Portfolio Optimization," *Financial Analysts Journal*, vol. 48, pp. 28–43, 1992.

[180] D. Bertsimas, V. Gupta, and I. C. Paschalidis, "Inverse Optimization: A New Perspective on the Black-Litterman Model," *Operations Research*, vol. 60, pp. 1389–1403, 2012.

[181] X. Jia and J. Gao, "Extensions of Black-Litterman portfolio optimization model with downside risk measure," in *Proc. Chinese Control and Decision Conf. (CCDC)*, pp. 1114–1119, 2016.

[182] V. N. Katsikis and S. D. Mourtas, "Binary beetle antennae search algorithm for tangency portfolio diversification," *Journal of Modeling and Optimization*, vol. 13, no. 1, pp. 44–50, 2021.

[183] J. Walters, "The Black-Litterman model in detail," *Available at SSRN: https://ssrn.com/abstract=1314585*, June 2014.

[184] G. He and R. Litterman, "The intuition behind Black-Litterman model portfolios," *Available at SSRN 334304*, 2002.

[185] T. Idzorek, "A step-by-step guide to the Black-Litterman model: Incorporating user-specified confidence levels," in *Forecasting expected returns in the financial markets*, pp. 17–38, Elsevier, 2007.

[186] A. Meucci, "Fully flexible views: Theory and practice," *Fully Flexible Views: Theory and Practice, Risk*, vol. 21, no. 10, pp. 97–102, 2008.

[187] A. S. D. Silva, W. Lee, and B. Pornrojnangkool, "The Black–Litterman Model for Active Portfolio Management," *The Journal of Portfolio Management*, vol. 35, pp. 61–70, 2009.

[188] W. Cheung, "The Black–Litterman model explained," *Journal of Asset Management*, vol. 11, pp. 229–243, 2010.

[189] R. Hohmann, *Portfolio Insurance Reloaded.* Springer, 2018.

[190] M. Escobar-Anel, A. Lichtenstern, and R. Zagst, "Behavioral portfolio insurance strategies," *Financial Markets and Portfolio Management*, vol. 34, pp. 353–399, 2020.

[191] M. Xu, M. Sherris, and A. W. Shao, "Portfolio insurance strategies for a target annuitization fund," *Available at SSRN 3417818*, 2019.

[192] M. S. Lobo, M. Fazel, and S. Boyd, "Portfolio optimization with linear and fixed transaction costs," *Annals of Operations Research*, vol. 152, pp. 341–365, 2007.

[193] S. Mirjalili, S. M. Mirjalili, and X. Yang, "Binary bat algorithm," *Neural Computing and Applications*, vol. 25, no. 3-4, pp. 663–681, 2014.

[194] S. Mirjalili and A. Lewis, "S-shaped versus v-shaped transfer functions for binary particle swarm optimization," *Swarm Evol. Comput.*, vol. 9, pp. 1–14, 2013.

[195] S. Corsaro and V. D. Simone, "Adaptive l_1-regularization for short-selling control in portfolio selection," *Comp. Opt. and Appl.*, vol. 72, no. 2, pp. 457–478, 2019.

[196] Y. Trichilli, M. B. Abbes, and A. Masmoudi, "Islamic and conventional portfolios optimization under investor sentiment states: Bayesian vs markowitz portfolio analysis," *Research in International Business and Finance*, vol. 51, p. 101071, 2020.

[197] Y. Zhang, J. Jiang, Y. Xiang, Y. Zhu, L. Wan, and X. Xie, "Cloud-assisted privacy-conscious large-scale markowitz portfolio," *Information Sciences*, vol. 527, pp. 548–559, 2020.

[198] J. Branke, B. Scheckenbach, M. Stein, K. Deb, and H. Schmeck, "Portfolio optimization with an envelope-based multi-objective evolutionary algorithm," *European Journal of Operational Research*, vol. 199, pp. 684–693, 2009.

[199] N. A. Canakgoz and J. E. Beasley, "Mixed-integer programming approaches for index tracking and enhanced indexation," *European Journal of Operational Research*, vol. 196, no. 1, pp. 384–399, 2009.

[200] P. Xidonas and G. Mavrotas, "Multiobjective portfolio optimization with non-convex policy constraints: Evidence from the Eurostoxx 50," *European Journal of Finance*, vol. 20, pp. 957–977, 2014.

[201] P. Xidonas and G. Mavrotas, "Comparative issues between linear and non-linear risk measures for non-convex portfolio optimization: Evidence from the s&p 500," *Quantitative Finance*, vol. 14, pp. 1229–1242, 2014.

[202] J. Nobre and R. F. Neves, "Combining principal component analysis, discrete wavelet transform and xgboost to trade in the financial markets," *Expert Syst. Appl.*, vol. 125, pp. 181–194, 2019.

[203] M. A. Akbay, C. B. Kalayci, and O. Polat, "A parallel variable neighborhood search algorithm with quadratic programming for cardinality constrained portfolio optimization," *Knowl.-Based Syst.*, vol. 198, p. 105944, 2020.

[204] E. Rashedi, H. Nezamabadi-Pour, and S. Saryazdi, "Bgsa: binary gravitational search algorithm," *Natural Computing*, vol. 9, no. 3, pp. 727–745, 2010.

[205] J. Tobin, "Liquidity Preference as Behavior Towards Risk," *The Review of Economic Studies*, vol. 25, no. 2, pp. 65–86, 1958.

[206] D. G. Maringer, *Portfolio Management with Heuristic Optimization*, vol. 8 of *Advances in Computational Management Science*. Springer, first ed., 2005.

[207] R. Jansen and R. van Dijk, "Optimal Benchmark Tracking with Small Portfolios," *The Journal of Portfolio Management*, vol. 28, pp. 33–39, 2002.

[208] S. Li, H. Chen, M. Wang, A. A. Heidari, and S. Mirjalili, "Slime mould algorithm: A new method for stochastic optimization," *Future Gener. Comput. Syst.*, vol. 111, pp. 300–323, 2020.

[209] A. T. Khan, X. Cao, Z. Li, and S. Li, "Enhanced beetle antennae search with zeroing neural network for online solution of constrained optimization," *Neurocomputing*, vol. 447, pp. 294–306, 2021.

[210] A. T. Khan, S. Li, and X. Cao, "Human guided cooperative robotic agents in smart home using beetle antennae search," *Science China Information Sciences*, vol. 65, p. 122204, 2022.

Curriculum Vitae

Spyridon D. Mourtas was born in 1987 in Patras, Greece. He received his B.S. degree in Mathematics from the University of Patras, Greece, in 2016 and his M.Sc. degree in Applied Economics and Finance from the National and Kapodistrian University of Athens, Greece, in 2019. During his Ph.D. studies in financial and computational mathematics, with the Department of Economics of the National and Kapodistrian University of Athens, Greece, Spyridon participated in the Erasmus Plus program and transferred to the Department of Computer Science, University of Nis, Serbia, for the elaboration of his research. It is worth mentioning that his Ph.D. studies were supported by the Stavros Tsakirakis scholarship. He has co-authored 40 articles in international peer-reviewed journals, as well as 2 book chapters and 3 conference papers. Additionally, Spyridon has participated at three conferences/webinars and has won one international award, the best paper award, for the paper entitled "Credit Card Attrition Classification Through Neuronets". His main research interests include neural networks, matrix analysis, and intelligent financial optimization.