Metaheuristic Approach for Solving Scheduling and Financial Derivative Problems

A thesis submitted

for the degree of Doctor of Philosophy

by

Nareyus I Lawrance Amaldass

Supervisor

Dr. Cormac A Lucas

Department of Mathematics, College of Engineering, Design and Physical Sciences

BRUNEL UNIVERSITY

September 2019

Certificate

It is certified that the work contained in this thesis entitled "Metaheuristic Approach for Solving Scheduling and Financial Derivative Problems" by "Nareyus I Lawrance Amaldass" has been carried out under my supervision and that it has not been submitted elsewhere for a degree.

Dr. Cormac A Lucas

Senior Lecturer

Department of Mathematics, College of Engineering, Design and Physical Sciences

Brunel University

September 2019

Abstract

The objective of this thesis is to implement metaheuristic approaches to solve different types of combinatorial problems. The thesis is focused on neighborhood heuristic optimisation techniques such as Variable Neighborhood Search (VNS) and Ant Colony Optimisation (ACO) algorithms. The thesis will focus on two diverse combinatorial problems. A job shop scheduling problem, and a financial derivative matching problem. The first is a NP-hard 2-stage assembly problem, where we will be focussing on the first stage. It consists of sequencing a set of jobs having multiple components to be processed. Each job has to be worked on independently on a specific machine. We consider these jobs to form a vector of tasks. Our objective is to schedule jobs on the particular machines in order to minimise the completion time before the second stage starts. In this thesis, we have constructed a new hybrid metaheuristic approach to solve this unique job shop scheduling problem.

The second problem has arisen in the financial sector, where the financial regulators collects transaction data across regulated assets classes. Our focus is to identify any unhedged derivative, Contract for Difference (CFD), with its corresponding underlying asset that has been reported to the corresponding component authorities. The underlying asset and CFD transaction contain different variables, like volume and price. Therefore, we are looking for a combination of underlying asset variables that may hedge the equivalent CFD variables. Our aim is to identify unhedged or unmatched CFD's with their corresponding underlying asset. This problem closely relates to the goal programming problem with variable parameters. We have developed two new local search methods and embedded the newly constructed local search methods with basic VNS, to attain a new modified variant of the VNS algorithm. We then used these newly constructed VNS variants to solve this financial matching problem.

In tackling the Vector Job Scheduling problem, we developed a new hybrid optimisation heuristic algorithm by combining VNS and ACO. We then compared the results of this

hybrid algorithm with VNS and ACO on their own. We found that the hybrid algorithm performance is better than the other two independent heuristic algorithms. In tackling the financial derivative problem, our objective is to match the CFD trades with their corresponding underlying equity trades. Our goal is to identify the mismatched CFD trades while optimising the search process. We have developed two new local search techniques and we have implemented a VNS algorithm with the newly developed local search techniques to attain better solutions.

Keywords: Scheduling, Variable Neighborhood Search, Any Colony Optimisation, Vector Job Scheduling, Contract for Difference, Equity

Acknowledgements

I sincerely express my gratitude to my supervisor, Dr. Cormac Lucas, for his enormous support, encouragement and the incredible knowledge that he has provided throughout my research. I am fortunate to have Dr. Cormac, for providing me with his invaluable time, and especially in agreeing to meet me during late evenings. His motivation, patience and guidance has helped me so much in completion of my Ph.D.

Further, I would like to thank Dr. Nenad Mladenovic, for his valuable knowledge, suggestions, his time and tremendous support during my Ph.D. His reviews and helpful comments have been a key indicator for the success of my Ph.D.

My sincere thanks to Dr. Paresh Date and Dr. Diana Roman, for their comments, suggestions and feedback during the annual review progress. Their reviews have shaped the progress of my Ph.D completion. Also, I would like to thank the administrative staff for their help and support.

Finally, I thank my parents and sister for their great support. They are the most important people and I dedicate this thesis to them. Also, special thank to my uncle Vincent Jayakumar Thambuswamy, for his phenomenal help and motivation during my education. Above all, I thank Almighty God for giving me the courage and wisdom to undertake this research, and strength to help me through its completion.

Publications

1. Amaldass, N.I.L., Lucas, C., and Mladenovic, N., "A heuristic hybrid framework for vector job scheduling", *Yugoslav Journal of Operations Research*,27(2017) 31-45.

2. Amaldass, N.I.L., Lucas, C., and Mladenovic, N., "Variable Neighborhood Search for financial derivative problem", *Yugoslav Journal of Operations Research*, 29(3)(2019).

Contents

List of Figures

List of Tables

Abbreviations

VNS	**V**ariable **N**eighbourhood **S**earch
ACO	**A**nt **C**olony **O**ptimization
CFD	**C**ontract **f**or **D**ifference
LS	**L**ocal **S**earch
SA	**S**imulated **A**nnealing
TS	**T**abu **S**earch
VND	**V**ariable **N**eighbourhood **D**escent
RVNS	**R**educed **V**ariable **N**eighbourhood **S**earch
BVNS	**B**asic **V**ariable **N**eighbourhood **S**earch
GVNS	**G**eneral **V**ariable **N**eighbourhood **S**earch
SVNS	**S**kewed **V**ariable **N**eighbourhood **S**earch
VNDS	**V**ariable **N**eighbourhood **D**ecomposition **S**earch
AS	**A**nt **S**ystem
EAS	**E**litist **A**nt **S**ystem
RBAS	**R**ank **B**ased **A**nt **S**ystem
MMAS	**M**ax **M**in **A**nt **S**ystem
BWAS	**B**est **W**orst **A**nt **S**ystem
ACS	**A**nt **C**olony **S**ystem
HCF	**H**yper **C**ube **F**ramework
VJS	**V**ector **J**ob **S**cheduling

Symbols

$N(s)$	neighborhood solution
s	initial solution
$\dot{s}$	new solution
$\ddot{s}$	improved new solution
k	number of ants
J	set of jobs
M	set of machines
t_{ij}	time spend on machine i by job j
C_j	time when job j is completed
$\lambda(i,j)$	pheromone deposit on the edge (i,j)
ρ	pheromone evaporation parameter
α	level of pheromone scent deposited by the current ants
β	heuristic information determined by the ants
$\eta(y)$	heuristic information stored at node y
N	set of all nodes
V	visited nodes
U	unvisited nodes
V_t	volume of trades
P_t	price of trades
V_c	volume of CFD
P_c	price of CFD
$\dot{V}_c$	over volume of CFD
$\ddot{V}_c$	under volume of CFD

$\dot{P}_c$	over price of CFD
$\ddot{P}_c$	under price of CFD

Chapter 1

Introduction

1.1 Introduction

Optimisation problems play a vital role in our day-to-day life. For instance, planning a travel route to reach a destination in the least possible time constitutes solving an optimisation problem. These informal problems are solved by individuals every day. Organisations, on the other hand, have a more formal optimisation problem, which is solved using mathematical modelling with precise constraints and dependent variables, so that their objective can be either maximised or minimised. Different approaches are used to solve optimisation problems, depending on the area, such as scheduling, timetabling, pricing, routing, logistics, supply chain management, financial planning, etc.

In complex optimisation problems, finding the global optimum is quite a daunting process. There may be many reasons, but one of the key issues is the size of the search space, which constitutes the computation time and cost. Heuristic methods can help to resolve these complex problems, leading to efficient computational time and cost. They can be categorised into various approaches, such as local search, global search, neighborhood search, biological, or nature-inspired methods. A heuristic method is successful if it can perform an efficient and swift search of the solution space.

In this thesis, we focus on two key heuristic approaches that are considered to be efficient and successful:

- Ant Colony Optimisation (ACO), [39], a biological metaheuristic method inspired by the characteristic of real ants. The exploration of the search space is well utilised by the construction of artificial ants that have the ability to produce good solutions.

- Variable Neighborhood Search (VNS), [90], a relatively new solution approach, which searches for a near or global optimum by constantly exploring local neighborhoods.

As part of the thesis, we have:

- Combined these two approaches to construct a new hybrid heuristic approach.
- Constructed two new variants of the VNS algorithms.

We have implemented our newly constructed approaches for two types of problems:

- A Vector Job Scheduling problem, in which the objective is to minimise the processing time of the job.
- A Financial Derivative Matching problem, where our aim is to match the corresponding underlying assets to identify the unmatched assets with their respective Contract for Difference (CFD).

Our results shows that our newly constructed hybrid and variants of VNS approaches have produced efficient solutions and could be used in other realted applications.

1.2 Problem Description

In this thesis, we present two unique problems:

- A Vector Job Scheduling problem, which is a variant of the job shop scheduling problem. While this is a two stage problem, we consider only the first stage - to schedule the jobs on their respective machines. Each job has multiple components, which have to be processed on a particular machine. Our objective is to schedule these jobs on their respective machines in a way that minimises their completion time. We have implemented our newly constructed hybrid approach, which combines a biologically inspired algorithm, ACO with VNS, to solve this. Further details about the problem, our approach, and its implementation, is provided in chapter 3.
- A Financial Derivative Matching problem that has arisen in the current financial regulatory framework. The objective of the problem is to find the unhedged underlying derivatives by matching the derivatives with their corresponding underlying

asset. These problems are similar to the goal programming [139]. We have constructed a new local search algorithm that has been embedded in a VNS, which led to the development of a new VNS variant. Our new constructed VNS variant has produced good solutions. Further details about the problem, our approach, and its implementation, is provided in chapter 4.

In both problems, VNS investigates the neighborhood of a feasible local minimum in search of the global minimum, where neighboring solutions are obtained by shaking. Local Search (LS) then searches the solution set in order to obtain the best solution.

1.3 Motivation

1.3.1 Vector Job Scheduling

The Vector Job Scheduling problem is a distinct type of scheduling problem with distinct characteristics. It is faced by various industries, for example, in manufacturing products can be manufactured by only a particular machine, while assembling of these products constitutes a new end product (i.e., computers, automobile vehicles, etc.). Scheduling these products in an optimal computational time will ensure time and cost reduction, and customer satisfaction.

1.3.2 Financial Derivative Matching

The financial Derivative Matching problem has arisen due to increasingly stringent financial regulations in the European Union. Since the recent financial crisis, regulation has become a vital component of the financial industry. Organisations working in the finance sector have an obligation to report all their traded transactions to their corresponding component authority. Financial regulation authorities collect and analyse huge volumes of data to identify suspicious market behaviour. One of the most challenging areas is the matching of financial derivatives with their underlying asset, which regulators are particularly keen on identifying. Resolving this problem helps regulators to identify any suspicious behaviour, and to understand the market behaviour. This in return assists with construction of new policies to support and promote economic growth.

1.4 Contribution

1.4.1 Vector Job Scheduling

The vector job scheduling is an NP (Non-deterministic Polynomial-time) hard optimisation problem. The search for a local optimum is time consuming, hence the problem is quite challenging. Since a job consists of multiple components being processed by its unique machine, scheduling these jobs by its respective component(s)/machine(s) and identifying the best solution is a difficult process.

1.4.2 Financial Derivative Matching

This is a goal programming problem. The quantity of trades is large, and the trades that are used to match one derivative cannot be used to match another derivative. The combination of various trades with its matching derivative trades increases the search space, which creates further challenges to the problem by making it difficult to attain the best solution.

1.4.3 Application of the Heuristics Method

Solving these unique problems through non-linear programming solvers is, generally, not ideal and takes a long computational time. Most solvers are not geared towards identifying the global optima. The problem can be solved by a mixture of search heuristics with local exhaustive (exact) searches of the local minima, or their approximations. This thesis follows this line of research.

1.4.4 Benefits and Application

The results and findings extracted from this research are beneficial to academia and industry. We have developed three new metaheuristic approaches. These approaches can be implemented in solving other problems and may be able to provide solutions to problems in a larger context too. Similarly, application in an industrial setting can greatly reduce computational time; thus, reducing the operational cost, while also being able to detect any discrepancy, especially in the financial markets.

Our newly constructed hybrid and VNS variants can be applied to other problems with high industrial relevance such as vehicle routing, manufacturing, financial surveillance, macroeconomics, and transportation. Thus, this research may contribute towards the mainstream applications of economic and market-oriented strategies.

1.5 Thesis Outline

In this thesis, we discuss two types of combinatorial optimisation problems - a Vector Job Scheduling that has been reported in Chen et al [28], and a Financial Matching problem that has been currently faced in the financial regulator sector.

A major aspect of this thesis focuses on the metaheuristic approaches. Chapter 2 provides the literature review, the first sections in this chapter provides details about optimisation technique focussing on metahueristics. We have used VNS as our base metaheuristic algorithm, which was introduced by Mladenovic et al. [90], and we also used ACO, which was introduced by Dorigo [37], which is discussed in detail in these sections. While, the later sections in this chapter, discusses about various scheduling problems an details about vector job scheduling and financial derivative problems and their literature reviews are described in this sections.

Chapter 3 illustrates our newly constructed hybrid metaheuristic algorithm, VNS-ACO, which is a combination of VNS and ACO that has been implemented to solve our Vector Job Scheduling problem. The mathematical model of our Vector Job Scheduling, as well as computational results with conclusions, are also discussed in this chapter. Further results are available in Appendix A and B.

The Financial Derivative Matching Problem with its mathematical model is presented in chapter 4. It further discusses VNS implementation with our newly constructed local search algorithms while exploring solutions to matching problems. Results and a conclusion are also provided in this chapter.

Finally, Chapter 5 presents our thesis, conclusions, and future work in this area of study.

Chapter 2

Literature Review

2.1 Metaheuristic

2.1.1 Introduction

The word "metaheuristic" is derived from two Greek words 'meta' meaning "beyond in the upper level" and heuristic meaning "to find". Osman et al. [100], formally defined metaheuristis as an iterative generation process, which guides a subordinate heuristic by intelligently combining different concepts for exploring and exploiting the search space. Learning strategies are used to structure information, in order to efficiently find near-optimal solutions. In his book on heuristic search, Salhi [119], presented the basic steps of the most popular heuristics, and stressed their hidden difficulties as well as opportunities. It provides a comprehensive understanding of heuristic search, the applications of which are now widely used in a variety of industries, including engineering, finance, sport, management and medicine. Blum and Roli [21], discuss the fundamental properties of metaheuristics. Some of the properties are:

- Metaheuristics are strategies that "guide" the search process.
- The goal is to explore the search space efficiently in order to find (near-)optimal solutions.
- Metaheuristic techniques range from simple local search procedures to complex learning processes.

- Metaheuristic algorithms are approximate and usually non-deterministic. They have integrated mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of metaheuristics permit an abstract level description.
- Metaheuristics are not problem specific.
- Metaheuristics use domain specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- Advanced metaheuristics use search experience (embodied in some form of memory) to guide the search.

Metaheuristics can be classified into nature inspired and non-nature inspired, or population based single start and multiple start. In some metaheurisitic search approaches, the objective functions are altered during the search process in order to escape from local minima. This type of approach is known as a dynamic objective function, and in the event that the objective function is fixed, it is deemed a static objective function. A metaheuristic is successful if it balances the intensification and diversification of the search within the neighborhoods of the search space [150].

2.1.2 Local Search

Local search methods are classical methods used by metaheuristics. They are considered as a basic principle for optimisation strategies, according to Johnson et al. [70]. The local search method will try to discover the local optimum by starting with the initial solution and improving gradually at each iteration. The search processes that are characterised by a trajectory in a search space are called trajectory methods. The characteristic of trajectory provides the behaviour and effectiveness of the algorithm. Bar-Yam [15], sees the search process of trajectory methods as the evolution in (discrete) time of a discrete dynamical system. Their algorithm starts with the initial solution and describes a trajectory in the state space. We will discuss some of the local search based heuristic methods.

2.1.3 Basic Local Search

The basic local search is also known as iterative improvement. It performs minor changes inside the neighborhood to attain a better solution. At each iteration, if the current solution is better than the previous one, the changes are retained, if not, the changes are

modified to get a better solution. This process is continued until no further improvements can be made. Below is the basic local search pseudo code with solution s.

Algorithm 1: Local Search

```
1 Generate a random initial solution s;
2 repeat
3 |   s ← ImproveN(s);
4 until no improvement is possible;
```

$N(s)$ is the neighborhood for solution s. The function Improve $N(s)$ is the improvement function to attain the new solution. This function can be the best improvement in an isolated instance or based on several instances of improvements, until no further improvement is possible. The method explores the neighborhood $N(s)$ and returns one of the solutions with the lowest objective function value, then stops at a local minima. Therefore, their performance strongly depends on the definition of N. The performance of iterative improvement procedures on combinatorial problems is usually quite inefficient due to time constraints.

2.1.4 Simulated Annealing

Simulated Annealing (SA) is one of the oldest metaheuristic algorithms, introduced by Kirkpatrick et al. [73] and Cerny [24], as a search algorithm for combinatorial optimisation problems. The basic idea is to allow hill climbing moves resulting in a worse solution than the current one, in order to escape local optima, and find the global optimum. The pseudo

code of the algorithm is given below [58]:

Algorithm 2: SA

1 $T \leftarrow T_0$;
2 $s \leftarrow starting - state$;
3 $E \leftarrow C(s)$;
4 **while** *not stopping-criteria* **do**
5 $\quad \dot{s} \leftarrow generate(s)$ with probability $G_{s\dot{s}}$;
6 $\quad \dot{E} \leftarrow C(\dot{s})$;
7 $\quad \Lambda \leftarrow \dot{E} - E$;
8 $\quad$ if $(\Lambda \leq 0) V (random() < e^{-\Lambda/T})$;
9 $\quad s \leftarrow \dot{s}$;
10 $\quad E \leftarrow \dot{E}$;
11 $\quad T \leftarrow reduce - temperature(T)$;
12 **end**

Line 1 sets the initial temperature to T_0. Lines 2 and 3 set the current state s and its cost E. The loop at lines 4-12 generates a trial state $\dot{s}$, evaluates the change in cost Λ. It selects the next current state and reduces the temperature until the stopping criteria is met. Line 8 shows how stimulated annealing accepts a trial state. The first term $(\Lambda \leq 0)$, express greed. It always accepts lower cost trail state. The random function returns a uniformly distributed random value between 0 and 1. The second term of line 8, $(random() < e^{-/T})$, expresses the likelihood of accepting a costlier trail state. When the stopping criteria is met, simulated annealing returns current state s as its outcome.

At high temperature, the second term of line 8, lets the algorithm to explore the entire state space. It accepts almost all cost increase as the temperature decreases, it explores big valleys, then smaller sub valleys to reach the outcome. This allows it to escape local minima.

Simulated annealing has a useful property. At a fixed temperature, it equilibrates, i.e, it approaches a stationary probability distribution. Temperature changes are usually chosen to keep transient distributions close to equilibrium. Simulated annealing equilibrium is the Boltzmann distribution, a probability distribution dependent solely on the cost function.

SA has been used in may applications. Osman [99], has provided an oscillation balance of cooling schedule. In Aarts et al. [2], have used a function of the control parameter to the cooling schedule to analyse the variance and expectation of the cost to solve a traveling salesman's problem. VanLaarhoven et al. [144], used SA to solve a job shop

scheduling problem and shown the algorithm asymptotically converges in probability to a globally minimal solution. Goffe et al. [56], used SA for econometric problems and found the global optimum for four different econometric problems. Currently, SA is used as a component in a metaheuristic rather than a stand alone search algorithm [21].

2.1.5 Tabu Search

Tabu Search (TS) was introduced by Glover [55], and is one of the most commonly used metaheuristics for solving various combinatorial optimisation problems. It is an effective algorithm to solve difficult problems. It uses a strategy of avoiding certain moves to prevent cycling, while providing the ability to escape from local optima. The pseudo code of the algorithm is given below:

Algorithm 3: TS

```
1 Generate Initial Solution s;
2 Initialize Tabu lists (TL_1, TL_2, ...., TL_r);
3 ṡ = s;
4 repeat
5 |   Find the best admissible solution s_1 ;
6 |   if f(s_1) > f(ṡ) then ṡ = s_1;
7 |   else  s = s_1  update Tabu list TL ;
8 until stopping criteria;
```

TS uses a short term memory, called the tabu list, that keeps track of the most recently visited solutions and forbids any moves towards them, it helps to escape local minima. The tabu list prevents cyclic moves and also prevents the moves from being reversed. Hence, the neighborhood of the current solution is restricted to the solution that does not belong to the tabu list. These solutions are classified as allowed sets.

At each iteration, the best solution from the allowed set is chosen as a new solution. This solution is added to the tabu list by removing the oldest solution that already exists in the list. The FIFO (First In First Out) technique is used to eliminate the old solution in the tabu list. The algorithm is terminated if the allowed sets are empty, or the termination condition is met.

The length of the tabu list is called tabu tenure that controls the memory of the search process. A large tabu tenure will force the search on large regions. Contrary, a small tabu tenure will enforce the search on small regions. TS can be made robust by varying

the tabu tenure. Talliard [136], varied the tabu tenure periodically at random intervals to attain the best results. Battiti and Tecchiolli [14], presented a dynamic tabu tenure.

There is a possibility that tabu might lose some unvisited good solutions due to its powerful nature. To overcome this issue, an aspiration criteria is used. The basic idea of the aspiration criteria is to allow a move, even if it gives a solution with a better objective value than the current best known one.

More recently, TS has been modified with other metaheuristic approaches to solve combinatorial problems. Teh and Rangaiah [140], have developed a new TS called enhanced continuous TS to solve many problems. Chelouah and Siarry [26], have concluded that there is some similarity between TS and GA and shown TS converges faster than GA in their application. Nowicki and Smutnicki [97], have implemented a TS technique with a specific neighborhood that employs a critical path method to solve the job shop scheduling problem. It finds a shorter makespan than other approximation approaches in shorter time. Drezner et al. [45], have efficiently applied a tabu search variable selection model within finance problems to predict corporate bankruptcy.

So far, we have discussed some basic metaheuristic approaches used to solve combinatorial optimization problems. With the idea of metaheuristic researchers have developed various effective metaheurisitc approaches that can be implemented in solving complex combinatorial optimisation problems. Two relatively new metaheuristic approaches are Variable Neighborhood Search (VNS) and Ant Colony Optimisation (ACO).

The VNS searches for a near global optimum, starting from several initial solutions, and changes the size or structure of the neighborhood of the current local optimum whenever its search stagnates. The ACO uses the characteristics of real ants to construct the solution guided by pheromone trails and heuristic information. The idea leads to the importance of the shortest path. We will discuss VNS and ACO in detail in the next sections.

2.1.6 Variable Neighborhood Search

Variable Neighborhood Search (VNS) was first introduced by Mladenovic and Hansen [90]. VNS is a metaheuristic that exhibits systematic change in the neighborhood during the search process. The initial solution is changed each time during the local search until a local optimum is reached. VNS is based on three major principles:

- A local optimal solution of one neighborhood structure is not necessary for that of another neighborhood structure;
- A global optimal solution is a local minimum with respect to all neighborhood structures;
- Local optimal solutions with respect to different neighborhoods are relatively close to each other.

Let us assume there are k neighbors structures N_k, $k = 1, ..., k_{max}$. The process starts with the initial solution, we obtain the next solution, from the neighborhood $N(s)$. Performing local changes in the neighborhood, we can obtain a best solution $\dot{s}$ from $N(s)$ Mladenovic and Hansen, [90], described VNS that performs several local searches with different neighborhoods until a local optimum is obtained. Below is the general working pseudo code for VNS:

Algorithm 4: VNS

1 Initialization: Select the set of neighborhood structures N_k for $k = 1, ..., k_{max}$ that will be used in the search;
2 Generate a random initial solution s;
3 Set $k = 1$;
4 Repeat the following steps until $k = k_{max}$;
5 Shaking: generate a point $\dot{s}$ randomly from $N_k(s)$;
6 Local Search: implement Local search method to obtain local optimum $\ddot{s}$ from $\dot{s}$;
7 **if** $\ddot{s}$ *is better than* $\dot{s}$ **then** set $s = \ddot{s}$ and $k = 1$;
8 **else** $k = k + 1$;
9 stop ;

In the above pseudo code:

- Line 1: Selects the neighborhood structure set N_k.
- Line 2: The Algorithm starts by generating a random initial solution s.
- Line 3: Initialising the value $k = 1$.
- Line 5: The algorithm executes the shaking steps by generating a new solution $\dot{s}$. Various neighborhood structures could be used in the shaking step to generate this new solution.

- Line 6: The new solution, $\dot{s}$, from the shaking step is improved by using local search. This local search step can use one or more neighborhood structures in order to improve the solution, $\ddot{s}$.
- Line 7: This step represents move or not, in case the new solution $\ddot{s}$ is better than the old solution s, then $\ddot{s}$ will become the new solution, s. Otherwise the old solution will remain as s. If the stopping criteria are not met, the algorithm repeats from the shaking step line 5 until the stopping condition is met.

VNS is a simple and effective metaheuristic approach to solve difficult optimisation problems. The idea of using more than one neighborhood in the search process has gained interest among various researchers and has been used in a variety of applications. Depending on the complexity of the problem and adaptability nature, VNS has led to several variants of VNS. In the following sections we will discuss some of the most often used VNS variants that have distinctive characteristics.

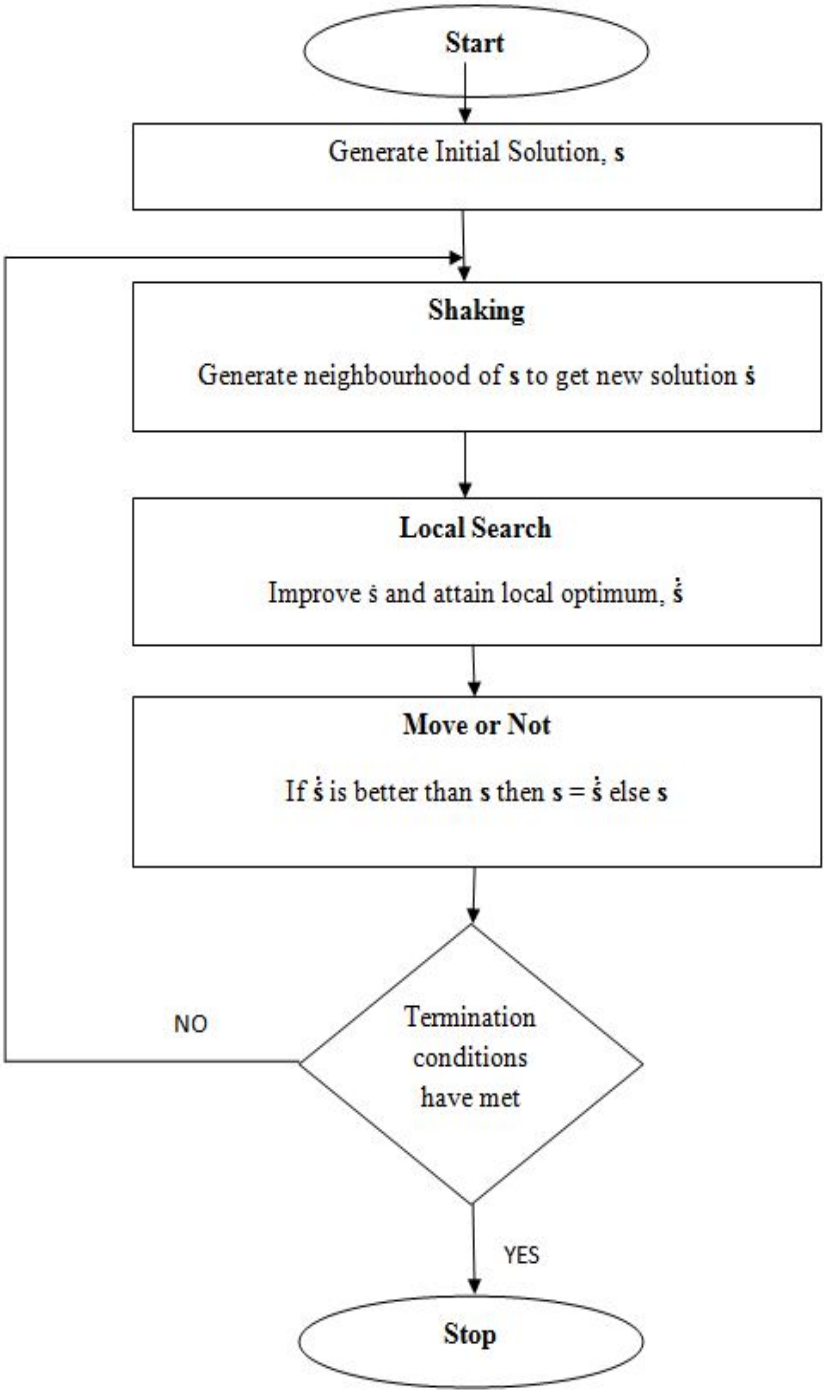

FIGURE 2.1: VNS Flowchart

2.1.6.1 Variable Neighborhood Descent

Variable neighborhood descent (VND) is a variant of VNS that explores the complete neighborhood and makes changes in a deterministic manner. Due to this process, VND results in large computation time. A frequent implementation consists of ranking moves by order of complexity of their application. This is often the same as by size of their neighborhood $N_i(s)$, and returning to the first one each time a direction of descent is found, and a step made in that direction. Alteratively, all moves may be applied in sequence as long as descent is made for some neighborhood in the series. The pseudo code is given below:

Algorithm 5: VND

1 Initialization: Select the set of neighborhood structures N_k,for $k = 1, ..., k_{max}$, that will be used in the descent;
2 Find an initial solution s (or apply the rules to a given s);
3 Repeat the following sequence until no improvement is obtained:;
4 (l)Set $k = 1$;
5 (2) Repeat the following steps until $k = k_{max}$;
6 (a) Exploration of neighborhood: Find the best neighbor $\dot{s}$ of s ($\dot{s}\epsilon N_k(s)$);
7 (b) Move or not: If the solution $\dot{s}$ thus obtained is better than s, set $s = \dot{s}$ and $k = 1$, otherwise, set $k=k + l$;

The computational time in VND is very high, and for that reason, it is used in larger-size combinatorial problems where the application uses more computational time. Rong and Kendall [114], investigated VND for the delay-constrained least cost (DCLC) multicast routing problem, and showed that VND outperforms other existing algorithms. The neighborhood structures they designed in the VND approaches are based on the idea of path replacement in trees. Liang and Wu [81], have implemented VND in the Redundancy Allocation Problem (RAP) and showed that VND overcame the limitation, and offered a practical way to solve large instances of the relaxed RAP where different components can be used in parallel.

2.1.6.2 Reduced Variable Neighborhood Search

Reduced Variable Neighborhood Search (RVNS) is another variant of VNS. It is mostly based on the third principal of VNS, a global optimum is the best solution across all neighborhoods. Hence, in a specific neighborhood a solution is randomly selected. This

random selection constitutes a stochastic search, and it does not use a local search to improve the solution. Below is the pseudo code for RVNS:

Algorithm 6: RVNS

1 Select the set of neighborhood structures N_k for $k = 1, ..., k_{max}$ that will be used in the search;
2 Find an initial solution s;
3 Choose a stopping condition;
4 Repeat the following sequence until the stopping condition is met;
5 (1) Set $k = 1$;
6 (2) Repeat the following steps until $k = k_{max}$;
7 (a) Shaking: Generate a point $\dot{s}$ at random from the k^{th} neighborhood of $s(\dot{s}N_k(s))$;
8 (b) Move or not: If this point is better than the incumbent, move there $s = \dot{s}$, and continue the search with $k = 1$; otherwise, set $k = k + 1$;

A set of neighborhoods $N_1(s), N_2(s), , .., N_{kmax}(s)$ is considered around the current point s (which may be or not a local optimum). Usually, these neighborhoods are nested, i.e. each one contains the previous. Then a point is chosen at random in the first neighborhood. If its value is better than that of the incumbent (i.e. $f(\dot{s}) < f(s)$), the search is reentered there ($s = \dot{s}$). Otherwise, one proceeds to the next neighborhood. After all neighborhoods have been considered, one begins again with the first, until a stopping condition is satisfied (usually it will be maximum computing time since the last improvement, or maximum number of iterations). Due to the nestedness property, the size of successive neighborhoods is increasing. Therefore, one will explore more thoroughly close neighborhoods of s than further ones, but nevertheless search within these, when no further improvements are observed within the first, smaller ones.

The RVNS can be applied to large instances where the elimination of local search may improve the computational time. Crainic et al. [34], provide additional performance analysis of a parallel implementation of the RVNS. Hansen et al. [62], have described the speed of RVNS. They have compared the performance of RVNS with other faster heuristics. Maric et al. [87], have implemented a hybrid version of RVNS to solve a bi-level incapacitated location problem with clients. They have shown that the RVNS hybrid performs better than swarm optimisation and simulated annealing.

2.1.6.3 Basic Variable Neighborhood Search

The basic variable neighborhood search (BVNS) is a variant of VNS. So far, we have seen VND, where the computational time is high as it searches every neighborhood and RVNS, that randomly chooses the solution which may not provide a good quality solution. BVNS is a hybrid of VND and RVNS. Thus, the BVNS uses a process to find the next optimal solution from the most fitting neighborhood structure, then the solution is further refined and improved by using a local search technique. This improved solution is the current solution from the neighborhood in the iteration. The process provides a good solution and save computational time without analysing the full neighborhood structure. The pseudo code for the BVNS is given below:

Algorithm 7: BVNS

1 Initialization: Select the set of neighborhood structures N_k, for $k = 1, ..., k_{max}$ that will be used in the search; Find an initial solution s;
2 choose a stopping condition;
3 Repeat the following sequence until the stopping condition is met:;
4 (1)Set $k = 1$;
5 (2) Repeat the following steps until $k = k_{max}$;
6 (a) Shaking: Generate a point $\dot{s}$ at random from the k^{th} neighborhood of $s(\dot{s}\epsilon N_k(s))$;
7 (b) Local search: Apply some local search method with $\dot{s}$ as initial solution; denote with $\ddot{s}$, the so obtained local optimum;
8 (c) Move or not. If the local optimum $\ddot{s}$ is better than the incumbent s, move there ($s = \ddot{s}$), and continue the search with $k = 1$; otherwise, set $k = k + 1$;

A series of neighborhood structures, which define neighborhoods around any point $s\epsilon\ddot{s}$ of the solution space, are first selected. Then the local search is used and leads to a local optimum s. A point $\dot{s}$ is selected at random within the first neighborhood $N_1(s)$ of s and a descent from $\dot{s}$ is done with the local search routine. This leads to a new local minimum $\ddot{s}$. At this point, three outcomes are possible:

- $\ddot{s} = s$, i.e., one is again at the bottom of the same valley. In this case, the procedure is iterated using the next neighborhood $N_k(s)$, $k > 2$.
- $\ddot{s} \neq s$ but $f(\ddot{s}) > f(s)$,i.e., another local optimum has been found, which is not better than the previous best solution (or incumbent). In this case, too the procedure is iterated using the next neighborhood.

- $\ddot{s} \neq s$ and $f(\ddot{s}) < f(s)$, and another local optimum, better than the incumbent has been found. In this case, the search is reentered around $\ddot{s}$ and begins again with the first neighborhood.

Should the last neighborhood is reached without a solution better than the incumbent being found, the search begins again at the first neighborhood $N_1(s)$ until a stopping condition, e.g., a maximum time or maximum number of iterations, or maximum number of iterations since the last improvement, is satisfied.

The BVNS is a commonly used VNS variant in many combinatorial optimisation problems by combining the deterministic and stochastic way of changing the neighborhood, and performing local search to improve the solutions. Sevkli and Aydin, [124], have proposed BVNS for job shop scheduling, compared the results and shown BVNS performs better than other published work in quality solutions. Amaldass et al. [7], have proposed a hybrid algorithm that combines VNS and ACO for the vector job scheduling problem. They have showed that the hybrid algorithm outperforms standalone VNS and ACO in solution quality.

2.1.6.4 General Variable Neighborhood Search

The general variable neighborhood search (GVNS) is a hybrid of VND and RVNS with BVNS. Initially, it explores RVNS to get the solution, then applies local search to get the

improved solution. Here, the local search of BVNS is replaced by a VND procedure:

Algorithm 8: GVNS

1 Initialization: Select the set of neighborhood structures N_k, for $k = 1, ..., k_{max}$ that will be used in the shaking phase, and the set of neighborhood structures N_l, for $l = 1, ..., l_{max}$, that will be used in the local search; find an initial solution s and improve it by using RVNS; choose a stopping condition;
2 Repeat the following sequence until the stopping condition is met:; (1)Set $k = 1$;
3 (2) Repeat the following steps until $k = k_{max}$;
4 (a) Shaking: Generate a point $\dot{s}$ at random from the k^{th} neighborhood $N_k(s)$ of s;
5 (b) Local search by VND ;
6 (b1) Set $l = 1$;
7 (b2) Repeat the following steps until $l = l_{max}$;
8 Exploration of neighborhood: Find the best neighbor $\ddot{s}$ of $\dot{s}$ in $N_l(\dot{s})$;
9 Move or not: If $f(\ddot{s}) < f(\dot{s})$ st $\dot{s} = \ddot{s}$ and $l = 1$
10 otherwise set $l = l + 1$;
11 (c) Move or not: If this local optimum is better than the incumbent, move there ($s = \ddot{s}$), and continue the search with $k = 1$; otherwise, set $k = k + 1$;

In the above GVNS pseudo code, we obtain a feasible solution by RVNS. In step 4, we perform the shaking procedure to attain the current best solution by the random process. In the next step we improve this current best solution by implementing VND as our local search.

GVNS has been applied to various large-sized problems. Mladenovic et al. [92], have implemented GVNS for the travelling salesman's problem, and provided the upper bounds in more than half of the existing benchmark instances. Andreatta and Ribeiro [8], have listed many applications where they have used GVNS.

2.1.6.5 Skewed Variable Neighborhood Search

The Skewed Variable Neighborhood Search (SVNS) is motivated by the topology of the search space and is a modified version of BVNS. The basic idea is to explore larger neighborhoods to escape from local optimum in order to move towards a global optimum. This characteristic has the advantage of permitting the solution to move to a worse solution

than the previous one. But this process could be extremely time consuming. The pseudo code of SVNS is given below:

Algorithm 9: SVNS

Initialization: Select the set of neighborhood structures N_k, for $k = 1, ..., k_{max}$ used in the search;
find an initial solution s and its value $f(s)$;
set $s_{opt} = s$, $f_{opt} = f(s)$;
choose a stopping condition and a parameter value α;
Repeat the following until the stopping condition is met:;
(1) Set $k = 1$;
(2) Repeat the following steps until $k = k_{max}$;
(a) Shaking: Generate a point $\dot{s}$ at random from the k^{th} neighborhood of s;
(b) Local search: Apply some local search method with $\dot{s}$ as initial solution; denote with $\ddot{s}$ the so obtained local optimum;
(c) Improvement or not: If $f(\ddot{s}) < f_{opt}$; set $f_{opt} = f(s)$ and $s_{opt} = \ddot{s}$;
(d) Move or not: If $f(\ddot{s})$ - $\alpha\rho(s, \ddot{s}) < f(s)$;
set $s = \ddot{s}$ and $k = 1$; otherwise set $k = k + 1$;

The relaxed rule for recentering uses an evaluation function linear in the distance from the incumbent: i.e., $f(\ddot{s})$ is replaced by $f(\ddot{s})$ - $\alpha\rho(s, \ddot{s})$, where $\rho(s, \ddot{s})$ is the distance from s to $\ddot{s}$, and α a parameter. A metric for the distance between solutions is usually easy to find, e.g. the Hamming distance when solutions are described by Boolean vectors, or the Euclidean distance in the continuous case.

Mladenovic and Hansen [90], have demonstrated SVNS for the weighted maximum satisfiability of logic problem. They have shown SVNS has performed better than tabu search for large and medium size problems.

2.1.6.6 Variable Neighborhood Decomposition Search

The Variable Neighborhood decomposition search (VNDS) was introduced by Mladenovic et al. [62]. It is a two level VNS that resolves the problem based on decomposition. The VNDS is an extension of BVNS. VNDS divides the search space into subspaces, and a local search method is used to analyze these subspaces. At each level, the local optimum is varied if there is an improvement in the solution. The VNDS exhibits the most reliable and effective local search tool that uses the idea of reducing the search process into subsets

of the whole space, and then it analyses more efficiently, in less computational time, to obtain good solutions, unlike a simple VNS. Below is the pseudo code for VNDS:

Algorithm 10: VNDS

1 Initialization. Select the set of neighborhood structures N_k, for $k = 1, ..., k_{max}$ used in the search;
2 Find an initial solution s choose a stopping condition;
3 Repeat the following until the stopping condition is met:;
4 (1) Set $k = 1$;
5 (2) Repeat the following steps until $k = k_{max}$;
6 (a) Shaking: Generate a point $\dot{s}$ at random from the k^{th} neighborhood of $s(\dot{s}\epsilon N_k(s))$ in other words, let y be a set of k solution attributes present in $\dot{s}$ but not in $(y = \dot{s}/s)$;
7 (b) Local search: Find a local optimum in the space of y either by inspection or by some heuristic; denote the best solution found with $\acute{y}$ and with $\ddot{s}$ the corresponding solution in the whole space $s(\ddot{s} = (\dot{s}/y)U\acute{y}$;
8 (c) Move or not: If the solution thus obtained is better than the incumbent, move there $(s < -\ddot{s})$,and continue the search with $N_1(k = 1)$; otherwise, set $k = k + 1$;

VNDS has been used in a number of applications. Lazic et al. [78], have implemented VNDS in $0 - 1$ mixed integer programs and shown its performance is better than other measurable approaches, and further used a special VNDS variant for the same problem and improved the lower and upper bounds, thus, reducing the optimality gaps. Their approach yields the best average optimality gap and running time for binary multi-dimensional knapsack benchmark instances.

2.1.6.7 Primal-Dual VNS

Generally, in a heuristic approach, the solution obtained may not be an optimal solution or near optimal, since the optimal solution is unknown. However, there are certain techniques that are able to attain the optimal solution. For instance, if the lower bounds of the objective function is known, then we may be able to obtain the optimal solution. Hence, mathematical programming is applied on primal variables to relax the integrality constraints. If the problem has a large instance, then the solver may not be able to find the best solution. To overcome this issue, we can solve a dual relaxed problem with the primal one. Primal-dual VNS has been successfully implemented on large problems to find the exact solution and guaranteed bound.

Hansen et al. [60], have implemented the primal dual VNS for the simple plant location problem. They have reduced the dual by exploiting the complementary slackness conditions. They have further implemented the primal dual VNS in solving p-median clustering problems and shown that the primal-dual VNS was able to tackle large datasets directly without the need for data reduction or sampling as employed in certain popular methods. Their computation results show that the primal-dual VNS outperforms other local search methods.

2.1.6.8 Summary and Conclusion

In this section, we have reviewed several variants of VNS that can be used to solve combinatorial optimisation problems. They explore distant neighborhoods in search of a global optimum. VNS uses a simple technique that requires few parameters. VNS can be hybrid with other heuristic approaches. Amaldass et al. [7], constructed a hybrid VNS with ACO for vector job scheduling and showed that the hybrid approach had better solutions than applying either VNS or ACO only. Kandavanam et al.[72], hybridise VNS with a genetic algorithm for multi-class network communication planning problem in order to satisfy service quality. They applied the hybrid heuristic to maximise the residual bandwidth of all links in the network and met the requirements of the expected quality of service. Palomo-Martinez et al. [102], implemented general VNS and a reactive Greedy Randomize Adaptive Search Procedure (GRASP), hybrid algorithm for an orienteering problem with mandatory visits and exclusionary constraints. Their computational results showed the efficiency of their hybrid when using large data instances. Irawan et al. [67], implemented hybrid metaheuristic technique by hybrid with VNS and Exact Methods for application to large unconditional and conditional vertex p-centre problems. Mladenovic et al. [91], implemented a new variant variable neighborhood search, called two level general variable neighborhood search (GVNS), for solving traveling salesman problems. They used GNVS as a local search and showed that it outperformed the tabu search heuristics. Seda [121], constructed a mathematical model for flow and job shop problems, where he proposed different methods for small and large problem instances.

Sevkli and Aydin [123], have proposed VNS for a job shop scheduling problem with makespan criterion. They have compared their results with the best known results in the literature, and concluded that VNS provides better quality solutions. Zhang at el. [153], have proposed a hybrid algorithm, combining VNS with a genetic algorithm, for flexible job shop scheduling problems. They have used VNS to improve the quality of the individual in the GA by strengthening the local search ability. Liao and Cheng [82], proposed

a new hybrid metaheuristic, which uses tabu search within VNS to minimise the total weighted earliness and tardiness with a restrictive common due date in a single machine environment. They examined 280 benchmark problem instances and showed that their algorithm produced a better quality solution and faster computational time.

- VND: Determines the change in neighborhoods, more likely to reach a global minimum if any neighborhood structures are used.
- RVNS: Useful for very large instances, for which local search is cost.
- BVNS: Deterministic and stochastic changes of neighborhoods, and systematic change in neighborhoods.
- GVNS: VND is used as local search within the BVNS, and very effective and useful for low level hybridisation.
- SVNS: Useful for clustering, local optima.
- VNDS: A two level VNS (decomposition at the first level) and useful for the last instance.

The above table summarises the main characteristic of various VNS variants we have discussed in this section. Depending on the specific optimisation problem, a VNS variant can be implemented. The optimal solution, or near optimal, can be attained by clever selection of neighborhood structures and the local search strategy. The local search strategy could vary from various local search techniques. Likewise, other heuristic approaches could be used as local search, which may lead to hybridisation.

In the next section, we will give a detail description of ACO and its corresponding variants.

2.1.7 The Ant Colony Optimization Algorithm

The development of ACO was inspired by the behaviour of real ants. Ants communicate with each other by a chemical substance produced by them, called pheromone. Ants explore the space in search for food, during this exploration they lay pheromone trails on the ground that mark the path from their nest to the foods, and vice versa. Ants smell pheromones that are laid by other ants. Ants follows the path that has the stronger pheromone concentration. Thus, the ants are able to find their way back to the nest or to the food source

The idea of ACO is to make the ants walk on the paths and construct a feasible solution. Each ant has a memory that records the movements on the paths. The construction of the solution is based on existing pheromone trails and heuristic information depending on the optimisation problem. Once the ants construct a feasible solution, pheromone is added to the path represented by each ant. Thus, solutions are stored for the pheromone trails and considered for the construction phase of the next iteration. To eliminate any bad path, a small amount of pheromone is deducted as an evaporation factor.

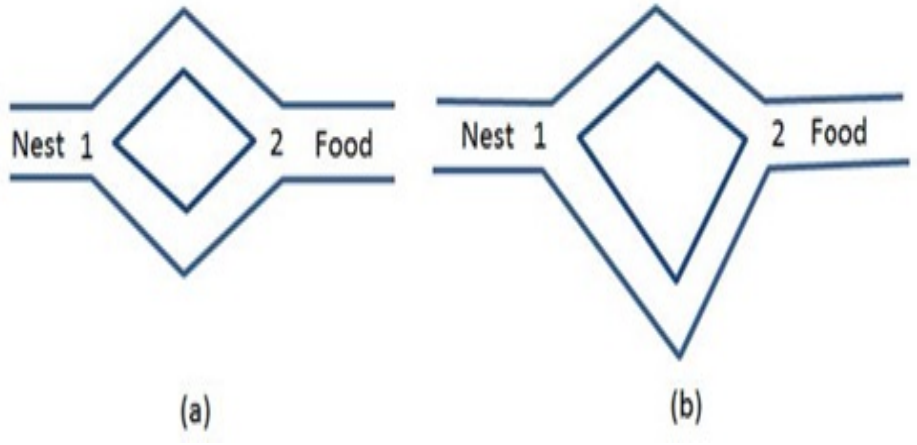

FIGURE 2.2: Ant Nest Graph

Deneubourg et al. [36], developed a double-bridge experiment that demonstrated the foraging of a colony of ants. In these experiments, the nest and the food sources are connected via two different path. Further, the behaviour is examined by varying the ratio between the lengths of the two paths as shown in the above Figure (2.2).

Following the idea of real ants, Dorigo et al. [42], constructed some key components in the design of artificial ants. One of them is to store the path of the ants while they construct a solution. The pheromone amount deposited on the path plays a vital role on the quality of the solution. The higher the strength of the pheromone deposit, the higher chances that the path will be followed. The pheromone evaporation is added to improve the search to escape from local optimum. The movement of the artificial ants depends both on some heuristic information and the pheromone trails.

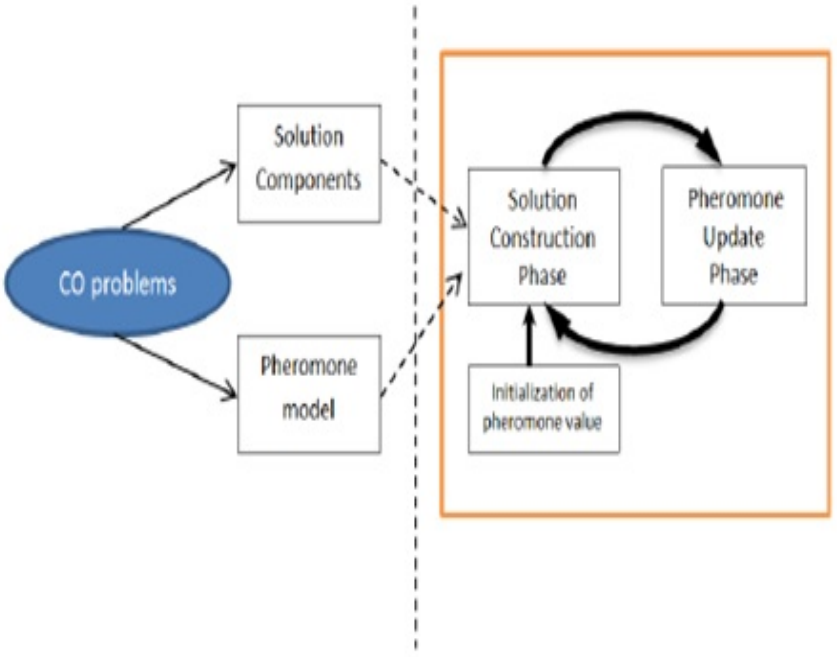

FIGURE 2.3: Basic Ant Construction

Figure (2.3) shows the basic principle of ACO [20], consists of two steps: a solution component that has to be derived to construct the solution, and the pheromone model that consists of pheromone values associated with the solution construction and are used to parameterise the probabilistic model. Dorigo and Stutzle [44], have constructed the ACO metaheuristc in three major phases, solution construction, pheromone updates, and

daemon action as an optional phase. Below is the basic pseudo code for ACO:

Algorithm 11: ACO

```
1  Initialize all edges to pheromone level λ_0 = 1;
2  Place each ant k on a randomly chosen node j;
3  for each iteration t do
4      while each ant k has not completed its set of nodes J do
5          for each ant k do
6              Move ant k to the next node j by the probability function P_k;
7          end
8          for each ant k with a complete set of nodes J do
9              Evaporate pheromone ρ;
10             Apply pheromone updates;
11             if ant k time is the shortest then
12                 global pheromone update;
13                 Update global solution;
14             end
15         end
16     end
17 end
```

The solution construction phase is a probabilistic construction that is associated with pheromone trail, heuristic information, and the constraints of the specific optimisation problems. The pheromone update phase improves the search space to gain a better quality solution. An evaporation function is used to decrease the pheromone in order to attain a good solution.

2.1.7.1 Characteristics of ACO

The ACO metaheuristic can be represented in a graph $G = (V, E)$, where V is a set of nodes and E is a set of edges connecting these nodes. The path in G corresponds to the set of solutions s. Pheromone trails are associated with either the nodes or the edges, although the latter is more common.

The Graph G represents an ant starting at node V and moving along the edges E, and storing the visited nodes within its path in its memory in a sequential order. The ants explore the graph G from each node in search of the solution using the pheromone trails,

which contain the heuristics and problem constraints. Once the tour is completed by visiting all the nodes, the ants calculate the best solution by tracing its best path depending on the pheromone it deposited.

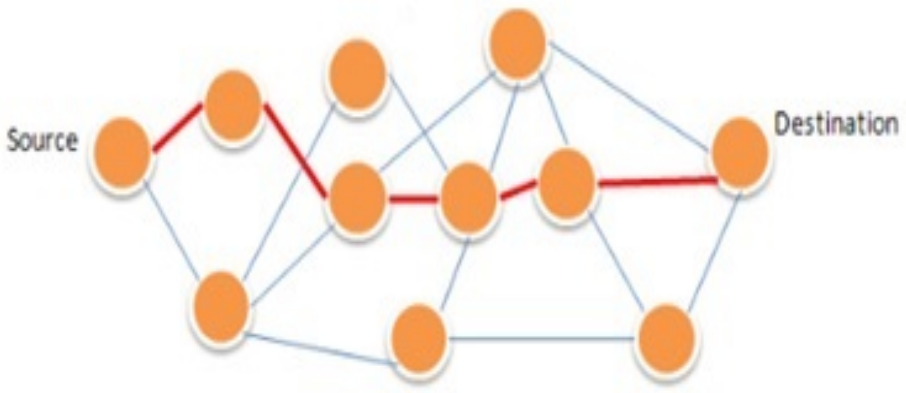

FIGURE 2.4: Ant Pheromone Exploration

Many researchers have used ACO metaheuristics on various combinatorial problems. Some have classified the behavior of ACO as an evolutionary algorithm as the ants communicate through pheromone trails. Some disagree that ACO is an Evolutionary Algorithm (EA) because of the lack of selection and cross over. However, the characteristic of ACO and EA are similar as both are population based and use adaptive memory. Dorigo et al. [42], have developed various ACO variants to improve the performance of the basic ACO framework. In the following sections, we will discuss various variants of ACO [88].

2.1.7.2 The Ant System

Dorigo et al. [42], introduced the ant system (AS) as the first ACO algorithm. The AS consists of an initial phase, where the pheromones are initialised, a construction phase, where the solution is constructed, and finally, a pheromone update phase. In the initial phase, the pheromone trails are set with an equal amount λ, such that $\forall\ (i, j)$, $\lambda_{ij} = \lambda_0 = k/\omega$, where i and j are the nodes, k is the number of ants and ω is some greedy constructive heuristic depending on the optimization problem, e.g., for the TSP the nearest-neighbor heuristic may be used, where a salesperson starts at a random city i and repeatedly visits the nearest unvisited city j, until all cities have been visited.

In the graph $G = (V, E)$, the solution, s may be associated with the set of vertices V with the pheromone trails λ on the edges E. Solutions are constructed at every stage along the path G. The choice of the solution depends on the probabilistic selection at each construction step. The probabilistic step may vary depending upon the ACO algorithm variants.

Considering the TSP example, at each iteration each ant k starts from a randomly chosen city. Each ant k builds a solution from one city to another by using a probabilistic decision rule. At each construction step the city visited is added to the partial solution of the ant. The probability that ant k chooses the next city j depends on the below probabilistic equation:

$$P^k(i,j) = \begin{cases} \frac{[\lambda(i,j)]^\alpha . \eta(j)^\beta}{\sum_{j \epsilon N_i^k} [\lambda(i,j)]^\alpha . \eta(j)^\beta} & \text{if } j \epsilon N_i^k, \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

where λ_{ij} and η_{ij} are the pheromone trails and heuristic information respectively, α and β are the constant parameters of pheromone trail and the heuristic information, respectively, N_i^k is the set of unvisited cities of ant k with i as the current city. The heuristic in the TSP problem is defined as $\eta_{ij} = 1/d_{ij}$, the distance between cities i and j are inversely proportional.

Once a feasible solution is built by the ants, the pheromone trails T^k are updated. We use a constant evaporation rate ρ to eliminate pheromone trails that impact the feasible solution. The pheromone evaporation factor is given by the following equation:

$$\lambda_{i,j} = (1 - \rho)\lambda_{i,j}, \forall (i,j) \tag{2.2}$$

where $0 < \rho$ is the evaporation rate. Once bad pheromone trails are eliminated, the pheromone deposited on the arc by the ants is given by the following equation:

$$\lambda_{i,j} = \lambda_{i,j} + \sum_{k=1}^{\mu} \Delta {\lambda_{i,j}}^k, \forall (i,j) \tag{2.3}$$

$$\Delta\lambda_{i,j}^{k} = \begin{cases} \frac{1}{C^k} & \text{if } (i,j)\epsilon T^k, \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

where $\Delta\lambda_{ij}^{k}$ is the amount of pheromone deposited by ant k between the cities i and j. T^k is the tour constructed by ant k and C^k is the cost. The solution depends on the amount of pheromone deposited by each ant k, the increase in the amount of pheromone deposited by the ant will yield a better ant tour, and thereby, produce better quality solutions.

The solution depends on the ant tour and the amount of pheromone deposited by the ants. This has led researchers to construct various techniques for the pheromone deposit that lead into different ACO algorithm variants. We discuss some of the AS variants in the next sections.

2.1.7.3 Elitist Ant System

The Elitist Ant System (EAS) is a modification of the AS where the best ant deposits an additional pheromone. It was introduced by Dorgio et al. [42]. The initial phase and the solution construction phase are similar to the AS algorithm. However, there are modifications on pheromone trails and pheromone evaporation rate:

$$\lambda_{i,j} = \lambda_{i,j} + \textstyle\sum_{k=1}^{\mu} \Delta\lambda_{i,j}^{k} + e\Delta\lambda_{i,j}^{bs}, \forall(i,j) \tag{2.5}$$

$$\Delta\lambda_{i,j}^{bs} = \begin{cases} \frac{1}{C^{bs}} & \text{if } (i,j)\epsilon T^{bs}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

Where $\Delta\lambda_{ij}^{k}$ is the amount of pheromone deposited by ant k that has been defined in equation (2.4). e is the additional pheromone trail for the best ant. $\Delta\lambda_{i,j}^{bs}$ is the amount of pheromone deposited by the best ant at a given time known as the 'best-so-far' ant. T^{bs} is the tour constructed by the best-so-far ant k and C^{bs} is the cost.

2.1.7.4 Rank-Based Ant System

Bullnheimer et al. [16], proposed an improvement of the EAS, called the Rank-Based Ant System (RAS). In the RAS, each ant deposits an amount of pheromone proportional to its

rank. In addition, the best-so-far ant will always deposit a higher amount of pheromone than the other ants as in EAS.

The initial phase and the solution construction are the same as in the AS algorithm. All the ants are ranked according to the solution they produced after the pheromone evaporation rate. Once ranked, only, the $w-1$ best-ranked ants and the best-so-far ant are allowed to deposit pheromone.

The best-so-far ant receives the highest weight to deposit pheromone, i.e., w, whereas the r^{th} best ant of the iteration has weight $max(0, w-r)$. Formally, the pheromone update in RAS is defined as follows:

$$\lambda_{i,j} = \lambda_{i,j} + \sum_{r=1}^{w-1}(w-r)\Delta\lambda_{i,j}^{r} + w\lambda_{i,j}^{bs}, \forall(i,j) \tag{2.7}$$

$$\Delta\lambda_{i,j}^{r} = \begin{cases} \frac{1}{C^r} & \text{if } (i,j)\epsilon T^r, \\ 0 & \text{otherwise.} \end{cases} \tag{2.8}$$

where $\Delta\lambda_{ij}^{k}$ is the amount of pheromone deposited by ant k and defined in equation (2.4). $\Delta\lambda_{i,j}^{bs}$ is the amount of pheromone deposited by the best so far ant defined by equation (2.6). $\Delta\lambda_{ij}^{r}$ is the amount of pheromone deposited by the rank best ant. T^r is the tour constructed by the rank best ant k and C^r is the cost of the r^{th} ranked best ant.

2.1.7.5 The Max-Min Ant System

The Max-Min Ant System (MMAS) is considered as one of the most efficient ACO variants and was introduced by Stutzle and Hoos [133]. One of the key characteristics of the MMAS is to define the quantity of pheromone trails on each edge between the range $[\lambda_{min}, \lambda_{max}]$. This helps to prevent the search being stagnated. Further the MMAS allows only the best-so-far ant or iteration best-ant to deposit the pheromone. The construction phase and initialization phase are similar to the AS.

$$\lambda_{i,j} = \lambda_{i,j} + \Delta\lambda_{i,j}^{best}\epsilon T^{best} \ ,\forall(i,j) \tag{2.9}$$

$$\Delta\lambda_{i,j}^{bs} = \begin{cases} \frac{1}{C^{best}} & \text{if } (i,j)\epsilon T^{best}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.10}$$

Equation (2.9) gives the pheromone update, where $\lambda_{i,j}^{best}$ is the pheromone deposit by the best-so-far ant and the cost is C^{best}. If we consider, the C^{best} to be the iteration best then C^{best} will be C^{ib}, the iteration best ant. If we consider, the C^{best} to be the best-so-far ant then C^{best} will be C^{bs}, the best-so-far ant.

The key principle of the MMAS is the pheromone trails that are bounded within the interval $[\lambda_{min}, \lambda_{max}]$, where$\lambda_{min}$, and λ_{max} are the lower and upper limits, respectively. This principle would generate suboptimal solutions by high intensity of pheromone by the best ant. This characteristic eliminates stagnation behaviour. If there is any stagnation, or there is no improvement in the solution, the pheromone trails can be re-initialised to re-estimate the upper limit and start exploring in a much wider space.

2.1.7.6 Best-Worst Ant System

The Best-Worst Ant System (BWAS) was first introduced by Cordon et al. [33]. In the BWAS, the pheromone updates are updated only by the best-so-far and the iteration worst ant. The initial phase and the construction of solutions phase are kept similar to the AS algorithm. The pheromone updates are defined as follows:

$$\lambda_{i,j} = \lambda_{i,j} + \Delta\lambda_{i,j}^{bs}, \forall(i,j)\epsilon T^{bs} \tag{2.11}$$

where, $\Delta\lambda_{i,j}^{bs}$ is the amount of pheromone deposited by the best-so-far ant that has been defined in equation (2.6). T^{bs} is the tour constructed by the best-so- far ant k and C^{bs} is the cost. The BWAS will penalise the iteration's worst ant, if they are not present in the T^{bs}, where T^{iw} is the solution of the iteration worst ant.

$$\lambda_{i,j} = (1-\rho)\lambda_{i,j}, \forall(i,j)\epsilon T^{iw}, (i,j) \notin T^{bs} \tag{2.12}$$

The re-initialisation of pheromone trail in the BWAS is different from the MMAS, where all pheromone trails are set to the initial pheromone value λ_0 only if for a given number

of algorithmic iterations no improvement is found. Furthermore, the BWAS uses concepts from evolutionary computation and introduces pheromone mutation to enhance the exploration. Therefore, for each iteration t, the pheromone trails are mutated with probability as follows:

$$\lambda_{i,j} = \begin{cases} \lambda_{i,j} + mut(it, \lambda_{threshoold}) & \text{if } R = 0, \\ \lambda_{i,j} - mut(it, \lambda_{threshoold}) & \text{if } R \neq 0. \end{cases} \tag{2.13}$$

where, R is a random value in $(0, 1)$, it being the current iteration, $\lambda_{threshold}$ being the average of the pheromone trail in the edges composing the global best solution and with $mut()$ being a function making a stronger mutation as the iteration counter increases.

2.1.7.7 The Ant Colony System

In the previous ACO variant, we have seen the variant differ only in the pheromone updates, with initialisation and solution construction as in the AS. The Ant Colony System (ACS) is different from other variants. The ACS was first introduced by Dorgio and Gambardella [41]. Considering the TSP problem, the ACS is the first algorithm that restricts the neighborhood of unvisited cities using candidate lists. In the AS, and its variant N_i^k was defined as all the cities that have not been visited by ant k yet. In the ACS, the unvisited cities are ranked with respect to the heuristic information η, and N_i^k contains only the best-ranked cities. The ACS uses more aggressive decision rules for construction of a solution, and during exploration than the AS:

$$j = \begin{cases} MAX_{l \epsilon N i^k} \lambda_{il} [\eta_{il}]^\beta & \text{if } R \leq q_0, \\ J & \text{otherwise.} \end{cases} \tag{2.15}$$

where R is a uniform random number in the interval $[0, 1], 0 \leq q_0 \leq 1$ is a parameter of the decision rule, called pseudo random proportional rule, and J is the random proportional rule defined in equation (2.1). In other words, with probability q_0 the ants make the best possible move, whereas with probability $(1 - q_0)$ they make a move probabilistically. The pheromone update of the ants from city i to j is given below:

$$\lambda_{i,j} = (1 - \xi)\lambda_{i,j} + \xi\lambda_0, \tag{2.16}$$

where, $0 < \xi < 1$ is a constant parameter and λ_0 is the initial pheromone value. When all ants construct their solution, the best-so-far ant is allowed to deposit pheromone as follows:

$$\lambda_{i,j} = (1 - \rho)\lambda_{i,j} + \rho\Delta\lambda_{i,j}^{bs}, \forall(i,j)\epsilon T^{bs} \tag{2.17}$$

where ρ is the evaporation rate which is performed only to the pheromone trails that belong to the best-so-far solution T^{bs}, and Δ_{ij}^{bs} is defined as in equation (2.6). The ACS has similar concepts of pheromone limit similar to MMAS but their limits are implicitly bounded in $[\lambda_0, (1/C^{bs})]$, while the MMAS are explicitly bounded.

2.1.7.8 Hyper-Cube Framework (HCF)

The hyper-cube framework is the most recent development in the ACO algorithms. It was introduced by Dorigo and Blum [38]. The HCF acts like a framework rather than a new variant. The HCF is a paradigm to implement ACO algorithms. The ACO variants such as the AS, the ACS or the MMAS can be implemented in the HCF.

The HCF has several benefits. One of the key benefits is the automatic handling of scaling of the objective function values, and it limits the pheromone value within the interval $[0, 1]$. Another key benefit is that the framework improves the average quality of the solutions produced continuously, and increases asymptotically in the case of the AS algorithm applied to unconstrained problems.

2.1.7.9 Applications of ACO

In recent years, researchers have shown that the ACO, a metaheuristic algorithm, is a competitive technique in terms of performance and time efficiency and used for various combinatorial optimisation problems. ACO was initially proposed by Dorigo and Caro [39], and has been successfully implemented to solve many NP-hard problems, like the TSP [40], network routing. Furthermore, Dorigo and Stutzle [43], suggested various ACO approaches for quadratic assignment problems (QAP), e.g. AS, RAS, MMAS and compared the results. Other researchers have implemented these ant based algorithms for various optimisation problems like scheduling, vehicle routing, networking, option pricing, etc. Colorni et al. [31], implement ACO for the job shop scheduling problem. Rajendran

and Ziegler [117], proposed ACO for permutation flow shop scheduling to minimise the make-span/total flow time of jobs. Huang and Liao [65], presented a hybrid algorithm, combining an ACO algorithm with a tabu search algorithm to improve the solution quality for the job shop problem . Zhang et al. [154], implemented an ant system for a small job shop problem, and compared it with traditional optimisation methods. Eswaramurthy and Tamilarasi [49], proposed tabu search with ant colony optimisation for job shop scheduling. They used dynamic tabu search strategies with neighborhood search depending on the ant colony. Surekha and Sumathi [135], used a genetic algorithm and ant colony optimization for solving fuzzy-based job shop problems, so that the sequence of jobs are scheduled using fuzzy logic and optimized using a genetic algorithm and ACO. Ponnambalam et al. [111], proposed an ACO algorithm for flexible job shop scheduling problems. It focuses on scheduling tasks for the manufacturing industry to improve machine utilisation or reducing time.

ACO variants have performed well in many combinatorial problems, especially in specific problems where heuristic information is available, such as the TSP and its variations. Zecchin et al. [152], have implemented ACO in solving water distribution systems. They have made a comparative study of five algorithms. Gajapal et al. [52], have implemented ACO for solving flow shop scheduling problems. They have developed and implemented a new ant colony algorithm to solve the flow shop scheduling problem with the consideration of sequence-dependent setup times of jobs. Their proposed ant colony algorithm gives promising and better results, as compared to those solutions given by the existing ant colony algorithm and the existing heuristics. Gao et al. [53], have implemented ACO in max-cut problems and introduced an antcut algorithm and showed their algorithm can solve max-cut problems in an efficient and effective way.

2.1.7.10 Summary and Conclusion

In this section, we have presented a biological metaheuristic ACO algorithm that was inspired by the behaviour of real ants. The construction of artificial ants led to three key phases, initialisation, construction, and update of the pheromone trails. The ACO was first implemented for the TSP by Dorgio et al.[42]. Since then, the development and implementation of the ACO algorithm has grown rapidly in various areas especially in combinatorial optimization problems.

Depending on the problem, many researchers have developed various ACO variants that have produced efficient results for their respective problems. When compared to other

meta-heuristic algorithms. ACO metaheuristics may have a some advantage in the combinatorial problems that have known heuristic information.

2.2 Scheduling Problems

2.2.1 Introduction

Scheduling is one of the key processes that allocates resources to tasks over a certain period of time. Scheduling occurs in various industries, and plays an important role to improve the productivity and efficiency. It is a decision-making process with the aim to optimise one or more objectives Pinedo [106].

In production, scheduling deals with allocation of operations on machines (i.e., a sequence of operations on machines) to minimise some performance issues, such as flow time, tardiness, lateness and makespan [107]. Scheduling problems deal with optimisation that may have many different performance measures. For example, to shorten completion time of a given last job, to minimise a number of jobs to be completed after their respective due dates [155], or to reduce total completion time for all jobs. This means that there could be various measures to optimise. Various scheduling problems have been, and a wide range of methodologies have been proposed. In this chapter, we will review some of the key literature describing scheduling problems and their classification [129].

Scheduling problems are classified depending on the resources and tasks. For example, if a job or a set of jobs has to be processed on one machine, then it has only one stage. These problems are identified as single machine stage scheduling problems. If a set of jobs has to be processed on multiple machines, there are multiple stages, and these are classified as multiple machine stages scheduling problems. In addition, if all jobs are available at the beginning of the scheduling process, then the scheduling is static. If the jobs are continuously changing over time, the scheduling problem is dynamic. The general parameters of the jobs are release times, routings and processing times. If these parameters are unknown in advance, they are referred to as stochastic parameters. If the parameters are known with certainty, it is a deterministic, while if there is an uncertainty about any parameter, it is stochastic [64].

In most scheduling problems, it is assumed that the setup times are negligible or a part of the processing time. However, if the set up times are considered, it is known as scheduling with sequence dependent setup times. Another characteristic of scheduling problems is

no-wait or blocking constraints between consecutive operations of the jobs. A no-wait constraint occurs when two consecutive operations must be performed without any interruption.

2.2.2 Taxonomy of Scheduling

There are different kind of scheduling problems, depending on their application and objectives. Manufacturing industry would focus on completion times, to ensure efficient production and ability to deliver customer orders. Scheduling a timetable for students or project management plan to complete the project on time are further examples of scheduling problems. Scheduling problems can also be very complex depending on their constraints - the classic challenging scheduling problems are shop scheduling problems. The basic shop scheduling model comprises a set of jobs and machines, and deals with determining an optimal or near optimal job sequence on each machine under some constraints. All shop scheduling problems belong to the NP-hard class [10] [59].

The basic taxonomy of scheduling problems are as follows:

- Single machine scheduling.
- Flow shop scheduling.
- Job shop scheduling.
- Open shop scheduling.
- Vector job scheduling.

2.2.3 Single Machine Scheduling

The jobs may be dependent or independent. If a set of jobs is independent and processed on a single machine, the problems are identified as Single Machine Scheduling problems (SMS) with independent jobs. If the jobs are dependent, it is classified as a Single Machine Scheduling problem (SMS) with dependent jobs.

In general, SMS problems consist of a set of jobs J where, $J = j_1, j_2, ..j_n$ jobs that has a single operations or tasks that has to be processed by a single machine M. The objective would be to minimize the completion time and find the best job sequence.

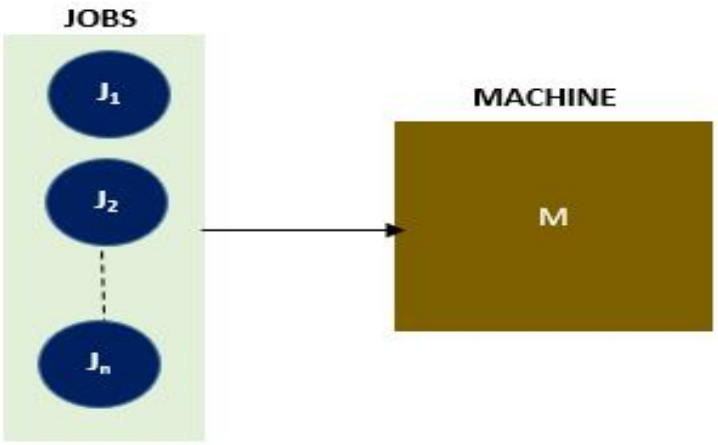

FIGURE 2.5: Single Machine Job

In general, SMS is NP-complete, see Pinedo, [106], has discussed the importance of theoretical models for scheduling. However, there are numerous complexity results on refinements of the SMS problem see Durr and Hurrand [47] and Sgall [125].

More generally, a valid schedule can be found in quasilinear time when the processing times are identical Simons [128], Garey et al. [54], and in polynomial time when the processing times are restricted to be either one or some arbitrary but fixed constant Sgall [125]. When the processing times are restricted to two fixed constants greater than one, SMS remains NP-complete [48]. Selvarajah, et al. [122], have used the polynomial time solution and have given the preemptive version of the problem. They have also presented an evolutionary metaheuristic algorithm for the general case for single machine batch scheduling with release times Lu, et al. [85], have minimized the makespan, i.e., the maximum delivery completion time of the jobs. When preemptions are allowed to all jobs, they have given a polynomial-time algorithm for this problem. Barzokia, et al. [13], have implemented and investigated and used to devise a branch-and-bound solution method by scheduling a set of jobs on a single machine for delivery in batches to one customer or to another machine for further processing and have minimized the sum of the total weighted number of tardy jobs and delivery costs. They have further investigated the structural properties of the problem for a single machine and special cases of the two-machine flow shop problem and used them to set up a new branch and bound algorithm.

2.2.4 Flow Shop Scheduling

Flow shop scheduling problems (FSP) consist of jobs $j_1, j_2, ..., j_n$ and machines $M_1, M_2, ..., M_m$. The objective is to find the optimal sequence of n jobs on m machines. Each job has to be processed on all machines, and the machines are sequenced as per the problem constraints. Once a job is processed on one machine, it is placed into a queue to be processed on the next machine. Depending on the problem constraints, the job can be scheduled ahead before the actual job queue depending on its priority. In general, the sequence of the jobs are first-in, first-out (FIFO) basis.

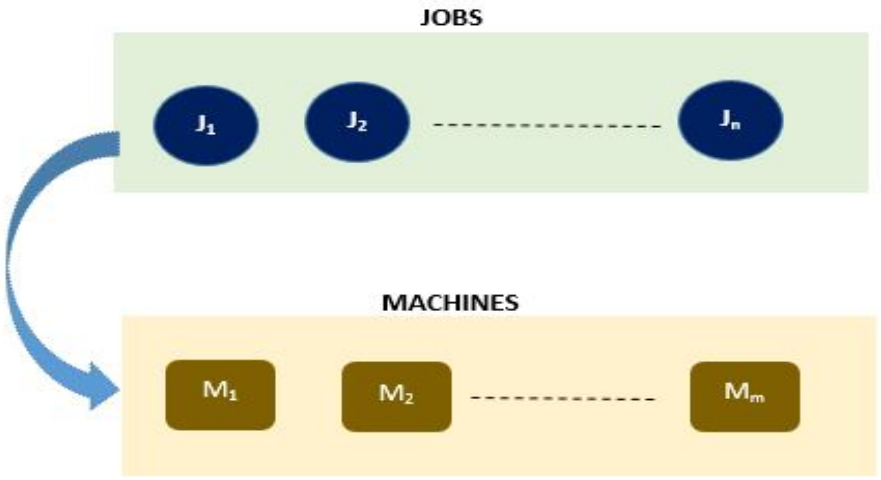

FIGURE 2.6: Flow Shop Scheduling

Figure 2.6, is an example, where jobs $j_1, j_2, ..., j_n$ are ordered in a series that has to be processed on machines $M_1, M_2, ..., M_m$. The machines are order in their respective series, so that each job is processed on each machine and moved to the next one either sequentially or as per a priority defined by the job order.

The FSP has been an interesting area of research for more than three decades, ever since Johnson, [71], anticipated a two stage scheduling problem with the makespan as an objective. Earlier, the FSP research is mostly based on Johnson's theory that provides a procedure to obtain an optimal solution with two or three machines with certain characteristics. Palmer, [101], has proposed a slope index based on the processing time to sequence the jobs on the available machines. Campbell and Smith [22], have proposed a heuristic by extending Johnson's theory. Their algorithm splits into a series of an equivalent to two machine FSP for the M machine problem, and solves each equivalent problem by Johnson's theory. Gupta [59], has recommended a new heuristic approach, similar to

Palmer's, [101], considering some exciting facts about optimality of Johnson's rule. Nawaz et al. [96], introduced a heuristic approach that was based on the assumption that a job with the highest processing time on all machines should be given a higher priority than a job with low total processing time.

Solimanpur et al. [131], implemented tabu search combined with neural networks for permutation FSP. They have used the modified NEH algorithm, proposed by Taillard [138], to generate the initial solution. They have used the insertion mechanism to generate the neighborhood structure, as it was found to be more effective than a random swap mechanism. Santos et al. [120], have implemented an exchange heuristic to improve the makespan of the multi-stage parallel FSP. Lian et al. [80], have proposed a particle swarm optimisation algorithm for solving the permutation FSP in relation to minimisation of makespan. Computational experiments show that it is more efficient than GA. Kuo et al. [76], have recommended a new hybrid particle swarm optimisation model that combines the random-key encoding scheme (RK), individual enhancement scheme, and particle swarm optimisation to solve the FSP and obtain a sequence of jobs that minimises the makespan. The experimental results indicate that the FSP based on the proposed HPSO produces a better solution when compared with GA. Wang et al. [146], proposed a hybrid approach of ordinal optimisation and a genetic algorithm called order based genetic algorithm to solve FSP. They have also tested various parameters of OGA and provided statistical results.

Rajendran and Ziegler [117], have proposed a tabu search algorithm, along with neural networks, and considered the permutation FSP by using ACO algorithms, with the objective to reduce the sum of the total flow time of jobs and makespan. The efficiency of the recommended ACO algorithm was assessed by considering the benchmark problems and upper bound values for makespan given by Taillard [138]. Shyu et al. [127], have developed the ACO algorithm to solve the two machine FSP with no waiting between operations, including the setup time. They have randomly chosen job processing times between 0 to 100. They have shown that the ACO algorithms outperform other algorithms.

Pan et al. [103], have proposed a discrete particle swarm optimisation algorithm for solving the no-wait FSP with both makespan and total flow time criteria. Solution quality was improved by hybridizing the DPSO algorithm with the variable neighborhood descent (VND) algorithm. A hybrid genetic algorithm for the FSP was proposed by Tseng and Lin, [143]. A modified version of NEH was used to generate the initial population, and a new orthogonal array crossover was developed as the crossover operator of the genetic algorithm. Hsu [66], has implemented an approximation algorithm for the assembly line crew scheduling problem in order to minimise the row sum by independently permuting the

elements in the column. Chong et al. [30], compared random generated populations and heuristic created populations with a genetic algorithm for assembly line balancing problems. Webster and Azizgolu [147], used dynamic programming algorithms for scheduling parallel machines, incorporating a family setup time. In this approach, jobs are partitioned into families, where setup time is required for the first job of each family but not for the latter jobs of the same family. Ishibuchi et al. [68], implemented genetic algorithms with neighborhood search algorithms for fuzzy FSP.

2.2.5 Job shop scheduling

In a basic job shop scheduling problem (JSP), each job is processed on its corresponding machines in their respective order with a given processing time, and each machine can process only one job at a time. The objective is to find the optimal ordering of all jobs with respect to their order requirements. Each job must visit the machine in a sequence. The difference between job shop and flow shop is that this sequence might differ for each job (multidirectional flow).

Figure 2.7 illustrates an example with $j_1, j_2, ..., j_n$ jobs are processed through various machines $M_1, M_2, ..., M_m$. The corresponding jobs are routed to their respective machines. In other words, in machine M_{ik}, the i represent the set of jobs $1, 2, ..., i$ and k represent a set of machine $1, 2, ..., k$. For example, M_{11} means job 1 is processed on machine 1 and M_{21} represents job 2 processed on machine 1. All jobs may not require the same number of machines. In order to show that each route may have a different number of machines, each route ends with a different variable for k. Each row represents the ordering of a job with respect to the same m number of machines.

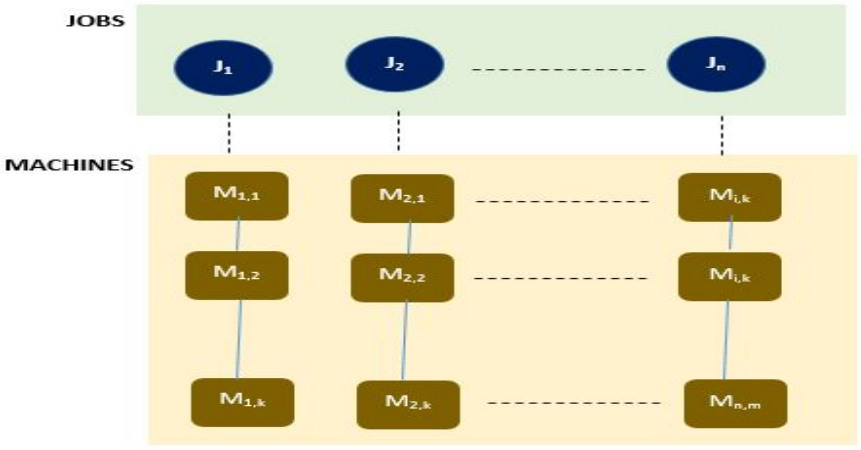

FIGURE 2.7: Job Shop Scheduling

The JSP is a classical NP-hard problem, especially difficult to solve even in relatively small instances [79]. Carlier and Pinson [23], have solved problem instances of 10 machines and 10 job. It was unsolved for nearly 20 years, and their work led to more research in the job scheduling field. Choi and Choi [29], introduced a mixed integer program integrated with local search scheme to study JSP with alternative operations and sequence-dependent setup times. Artigues and Roubellat [9], have proposed a polynomial insertion algorithm for multi-resource job-shop JSP with sequence-dependent setup times to minimise maximum lateness. First, they described the algorithm for pure JSP and then multi-resource requirements were introduced for the operations.

Low et al. [84], have developed a mathematical programming approach with the objective to reduce the sum material processing cost, setup time cost and inventory cost. Subra-Maniam et al. [134], have developed a framework to solve and optimize JSP problem with uncertain processing times, in which imprecise processing times are modeled as triangular fuzzy numbers. Fandel and Stammen-Hegene [50], have investigated an integrated job shop production planning and scheduling problem. Heinonen and Pettersson [63], solved JSP by using four different variants of the ACO algorithm. They further implemented a hybrid model which uses a postprocessing algorithm to improve the resulting schedule.

Yang et al. [151] introduced a new probabilistic model to solve JSP. Yamada and Nakano [149], have proposed a GA that uses problem-specific representation of solutions with crossover and mutation which are based on the Giffler and Thompson algorithm. Jaszkiewicz

[69], has proposed genetic local search (GLS) which is a hybridization of GA and local search.

The first ACO algorithm was proposed by Colorni et al. [31]. The performance of ACO algorithm was unsatisfactory due to slow convergence, long computing time and falling into a local optimum easily. Steinhofel et al. [130], have presented simulated annealing based algorithms for the classical JSP problem where the objective is to minimise the makespan. Kolonko, [74], has proposed a new approach that used a small population of SA embedded with the GA framework. Moreover, SA algorithm was used in three schemes, i.e., pairwise exchange, insertion, and random insertion.

2.2.6 Open Shop Scheduling

In the open shop, there are m machines, and each job has to be processed again on each one of the m machines. However, some of these processing times may be zero. There are no restrictions with regard to the routing of each job through the machine environment. The scheduler is allowed to determine the route for each job, and different jobs may have different routes [27]. The basic assumptions are that each job may visit a certain machines at most once. The processing times are independent of the sequence. There is no randomness; all data is known and fixed. All jobs are ready for processing at time zero, in which machines are idle and immediately available for work. No pre-emption is allowed, i.e., once an operation is started, it must be completed before another operation can be started on that machine. Machines never breakdown and are available throughout the scheduling period. There is only one of each type of machine.

In figure 2.8, jobs $j_1, j_2, ..., j_n$ are to be sequenced for processing by machines $M_1, M_2, ..., M_m$.

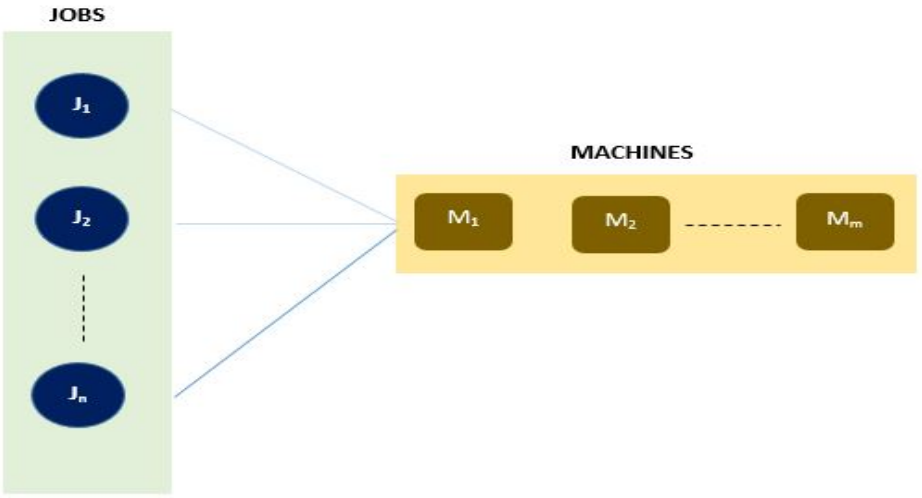

FIGURE 2.8: Open Shop Scheduling

Akker et al. [3], studied two machine open shop scheduling problems and proposed two non-dominated points D1 and D2 and a feasible schedule. The objective of this study was to minimise the makespan. They considered the algorithm by Gonzalez and Sahni [57], as a pre-requisite to the conditions that are proposed in their research. Kubale and Nadolski [95], compared the computational complexity results of cyclic open shop problem and compact cyclic open shop problem by considering the computational complexity of a cyclic open shop scheduling problem, and also a modification of cyclic open shop called as compact cyclic open shop with the objective of minimizing the makespan. Brasel et al. [18], reduced the sum of completion times or mean flow times by proposing several constructive algorithms, like matching algorithm, beam insert and beam append procedures with beam search, and generated active and non-active delay schedules.

Naderi et al. [95], explored scheduling open shops with parallel machines at each stage of processing. The objective was to shorten the total completion time. They proposed a Mixed-Integer Linear Programming (MILP) model to formulate an open shop with parallel machine. Also, they developed a hybrid of genetic algorithm. Sha et al. [126], proposed a multi-objective particle swarm optimisation technique with modified parameters like particle position, particle movement and particle velocity. They conducted experiments and reported their results. This algorithm has not been compared with any of the existing algorithms.

2.2.7 Vector Job Scheduling

Vector scheduling (VJS) is a multi-dimensional extension of traditional machine scheduling problems. Whereas in traditional machine scheduling a job only uses a single resource, normally, time, in vector scheduling a job uses several resources. In traditional scheduling, the load of a machine is the total resource consumption by the jobs that it serves [145]. In vector scheduling, we define the load of a machine as the maximum resource usage over all resources of the jobs that are served by this machine [11]. In the setting that we consider here, the makespan, which is normally defined to be the time by which all jobs are completed, is equal to the maximum machine load [12]. Al-Anzi and Allahverdi [4], implemented a hybrid tabu search algorithm to minimise the completion time for two stage assembly problems.

In recent years, VJS has become a complex combinatorial problem faced by different industries. Several researchers have implemented various tools for these kind of complex scheduling problems [17] [25].

VJS problem is one of the key studies in our thesis. Our focus is on solving separately the first stage of the two stage assembly scheduling problems [109, 110]. This problem is, for example, faced by the computer hardware industry. Computers are produced by assembling various components depending on customer specifications. These components are monitors, hard disks, CD/DVD ROM's, keyboards, mouse etc. Each component, having its own specifications, is made separately by an appropriate machine. As computers are manufactured based on specific requirements, a variety of computers are needed, which can be managed by combining the components in many different ways. When all of the components are ready, they are sent to a customer who does the second stage of assembling. Our objective is to minimise customer's waiting time. The specific components are considered to be a vector of jobs, which are independent and manufactured on a specific machine.

The two stage models have been studied in Potts et al. [110]. The results of Chen and Hall [28], give the lower bound of the performance guarantee, whereas we study upper bounds by using metaheuristic approaches. Potts et al. [110], investigated the second stage objective function to minimise the makespan, including the time for assembling. The assembly departments are spread out, and they are not implicitly included in our model. The completion time of a job is defined as the maximum time undertaken to complete the manufacturing of all of its components. The problem is to schedule the vector of jobs such that the sum of the completion times of all the jobs is minimised. This way, the assembling

of all final products will be completed as early as possible. We consider a deterministic formulation of the VJS. Chen and Hall [28], have already shown the NP hardness of the problem. Therefore, to solve this problem, it is natural to apply ACO and VNS, and general heuristic approaches. In Chapter 3, we have defined our mathematical model for the VJS problem and provided a solution to resolve these problems, and discussed the results.

2.2.8 Financial Derivative Problem

Financial market surveillance has become one of the key focus areas for financial regulators. Various techniques have been used to detect market abuse behaviour in the financial markets. Punnuyamoorthy et al. [113], introduced a hybrid data mining technique for detection of stock price manipulation. They used GA with an Artificial Neural Network technique to classify activities that would have potential manipulation. Pirrong [108], examined the Ferruzi Soy bean episode of 1989 and demonstrated how to detect manipulation in the commodity market. He concluded that the regulation in the US market was complex, confusing, and inefficient in futures and securities. It means that the market relies on costly preventative measures rather than ex post deterrence. Ogut et al. [98], investigated the best technique to detect stock price manipulation. They developed a data mining technique (ANN and SVM), and multivariate statistical technique (discriminant analysis, logistic regression) for the Istanbul stock market. They concluded that the performance of the data mining technique in terms of total classification accuracy and sensitivity statistics was better than those of the multivariate techniques. Comerton-Forde et al. [32], demonstrated the impact on an equity market by using the close price manipulation cases. They further constructed an index to measure the probability and intensity of closing price manipulation and estimated its classification accuracy. David et al. [35], modeled cross-border market surveillance activities as service systems that interact in a service oriented economy. In the paper, the market surveillance activities are described as user or customer driven service value networks. The cases were considered as configuration of value networks and value propositions in which the provider and the customers of the service are assumed to be the regulator.

Toumi et al. [142], proposed an efficient method using two variants of the VNS heuristic to solve the (0-1) quadratic knapsack problem. They compared large size instance with 1000 and 2000 binary variables, and compared the results with other results in the literature. Pererira al et. [105], investigated a test assembly design problem. They solved the problem by implementing various neighborhood and variable neighborhood search methods, and

found their results outperformed the results obtained in the previous literature. Puchinger et al. [112], proposed Relaxation guided variable neighborhood search. That is based on a general VNS scheme and a new variable neighborhood descent (VND) algorithm. The relaxations are used as an indicator for the potential gains of searching the corresponding neighborhood. The algorithm was tested on multiple dimensional knapsack problems and achieved promising results. Durate et al. [46], explored the adaptation of VNS to solve multi-objective combinatorial optimisation problems. They described how to design the shake procedures, the improvement methods and acceptance criteria for different VNS algorithms with more than one objective. They validated their proposed design on multi-objective combinatorial problems.

In chapter 4, we have introduced a unique financial derivative problem, that is currently being faced by financial regulators around the world. The logic of the problem is to match the trades with the corresponding CFD's. We have defined the mathematical model and have implemented the variants of VNS, and have discussed the results.

2.2.9 Performance Measures in Scheduling

Performance measures have become a vital issue in scheduling problems. Stating an objective for a scheduling problem is not complex as it has various parameter and the industrial requirement varies depending on their needs. A large number of scheduling problems have been studied with regular performance measures. The most widely considered regular performance measures are [94].

- Makespan: The objective is to minimize the maximum completion time of the schedule.
- Mean flow time: The objective is to minimise the average time spent by a job in the system. Flow time is defined as the elapsed time from when the job is ready to be processed until it has finished.
- Total tardiness: The objective is to minimise the summed lateness of all jobs in the system. Lateness is defined by how much later a job has finished after its deadline.

Makespan play a key role in maximising the productivity and resources. Makespan and total flow time are the key measure for maximising the system utilisation and work in progress, while the tardiness is related to job due dates. In recent times, delivery has

become a significant factor while industries offer a variety of products to their customers. The need of efficient delivery has become a key issue as delay in delivery may result in loss of customers. Hence, scheduling problems with due dates have become an important parameter in most industries and areas looked at by researchers.

Most of the research reported in the literature is focused on the single objective case of shop scheduling problems, in which the makespan is minimized, while fewer researchers have investigated multi-objective scheduling problems. However, multi-objective problems have become an important factor in the current dynamic competitive environment. Some researchers have considered the multi-objective nature of scheduling problems but restricted to two or three criteria of performance measures. Most of the research work tends to be based on highly unrealistic assumptions, implementing it is almost infeasible, to deal with scheduling problems in a real world manufacturing environment. This environment is complex, dynamic, and stochastic, and subjected to different disruptions due to a wide range of stochastic uncertainties.

2.2.10 Summary and Conclusion

From our study of the existing literature, we see that the scheduling problem is a complex and important field of research with different areas of application in the industrial sectors. We can see that the scheduling problem has become more complex due to the current requirements and evolution of the industry.

Despite the relative implementation of exact algorithms and heuristic methods, they are still incapable of solving medium and large instances. It is essential to study non-exact but efficient heuristics [[116],[86], [5]]. Therefore, efficient metaheuristic procedures, such as local search methods, simulated annealing (SA), tabu search (TS), ant colony optimization (ACO) and variable neighborhood search system (VNS), particle swarm optimization (PSO) and genetic algorithm (GA), have been proposed to find an approximate solution, close to the optimum, with considerably less computational time.

We have found from the literature that the VJS problems are the least explored and require more focus. Therefore, our attention in this research is focused on the VJS problems. We have used metaheuristic neighborhood and biological algorithm with hybrid solution to resolve these scheduling problems. Details of our research is provided in Chapter 3.

Chapter 3

Vector Job Scheduling

3.1 Introduction

In this chapter, we first describe the mathematical formulation of the problem, then, we implement biological algorithm, ACO, proposed by Dorigo and Caro [39], and then VNS proposed by Mladenovic and Hansen [90]. We, propose a hybrid algorithm that combines ACO and VNS for vector job scheduling problem. Finally, we compare our results with an integer programming solver, CPLEX.

3.2 Mathematical Programming Formulation

The vector job shop problem (VJS) is to schedule the jobs on the corresponding machines so that the time of the assembly machine to process the jobs is minimizsed. The processing of the jobs should remain in the same sequence on each machine, and no job is allowed to be interrupted. The mathematical programming formulation is given below:

Indices

- $J = \{1, 2, \ldots, j, \ldots, n\}$ denotes a set of jobs,
- $M = \{1, 2, .., i, ., m\}$ denotes a set of machines.

Data

- t_{ij} be the time spent on machine i by job j. $\forall$ i ϵJ, $j \epsilon M$

Variables

- C_j be the time when job j is completed, $\forall$j ϵM
- x_{jk} a binary variable that gets value 1 if job j is performed before job k.

$$x_{jk} = \begin{cases} 1 & \text{if job } j \text{ is performed before } k \\ 0 & \text{otherwise.} \end{cases}$$

The complete mathematical programming formulation can be written as:

$$minimise \sum_{j=1}^{n} C_j \tag{3.1}$$

subject to

$$C_j \geq t_{ij} + \sum_{k=1 k \neq j}^{n} t_{ik} x_{kj}, \ \forall j \in J, \forall i \in M \tag{3.2}$$

$$x_{jk} + x_{kj} = 1, \ \forall j, k \in J, \ j \neq k \tag{3.3}$$

$$x_{jk} + x_{kl} + x_{lj} \leq 2, \ \forall j, k, l \in J, \ j \neq l \neq k \tag{3.4}$$

$$x_{jk} \in \{0, 1\}, \forall j, k \in J. \tag{3.5}$$

The objective function is to minimise the total completion time. The constraint group (3.2) determines the earliest completion time of each job by considering the time of its completion on every machine. The summation term ensures that all earlier tasks on this machine are completed before this task. The constraint group (3.3) ensures that job j is performed always before job k, or job k is completed before job j. If job j is before job k then, it must be true for all machines. The constraint group (3.4) is a cycle breaking constraint, i.e., it ensures that the ordering of jobs does not fall into a cycle. Thus, if job j is before job k and job k is before job l, then job j must be before job l.

3.3 Construction for VJS

To solve the VJS problem, in this section we first present two metaheuristic based approaches, ACO and VNS, and later we propose a hybrid heuristic ACOVNS, that combines the both ACO and VNS approaches.

3.3.1 ACO

The ACO [39], was discussed in detail in Chapter 2 of this thesis. We use this approach in our Vector Job Scheduling (VJS) problem.

In our VJS problem, we establish the sequence of jobs in order to minimise the completion time. We could formulate the VJS problem by constructing an edge between the jobs, only if the corresponding job can be performed by the corresponding machine component. This process could be modelled in the form of a matrix, where Table 3.1 is a sample of the constructed matrix with m machines and n jobs, and the rows represent the machines and the columns represent the jobs. Each matrix cell represents the processing time t_{ij}. Our objective is to sequence the jobs to minimise the total processing time C is minimized.

Machine/Job	1	2	3	4	..	j	..	n
1	t_{11}	t_{12}	t_{13}	t_{14}	..	t_{1j}	..	t_{1n}
2	t_{21}	t_{22}	t_{23}	t_{24}	..	t_{2j}	..	t_{2n}
3	t_{31}	t_{32}	t_{33}	t_{34}	..	t_{3j}	..	t_{3n}
4	t_{41}	t_{42}	t_{43}	t_{44}	..	t_{4j}	..	t_{4n}
.	..	..	..	..	..	$t..$	..	..
i	t_{i1}	t_{i2}	t_{i3}	t_{i4}	..	t_{ij}	..	t_{in}
.	..	..	..	..	..	$t..$	..	..
m	t_{m1}	t_{m2}	t_{m3}	t_{m4}	..	t_{mj}	..	t_{mn}

TABLE 3.1: m machines and n jobs

3.3.2 Solution Construction

We construct a disjunctive graph $G = (N, E)$ where N is the set of nodes that represent processing times t_{ij}, job j on machine i. The initial node N_0 and the final node N_{n+1} are considered dummy nodes that are used to indicate the start and the end node for the ants. E is a set of edges. Each edge has a measure associated with the amount of pheromone

λ scent deposited, while the heuristic information is the inverse of processing time on the nodes. For example, In Figure 3.1, we consider 3 set of jobs, J_1, J_2, J_3 to be processed on a machine i_1. Each set of job consists of permutation of jobs $j'1, j'2, j'3$. The circle represents the processing time t_{ij} in which a job j is performed on a machine i. An ant k starts from the initial node, finds its own path and creates a solution by forming a sequence of visiting nodes. We have to verify that the ants do not violate any of the constraints such as maintaining the correct job ordering on each machine. The visiting of nodes is in sequential ordering. Each path on the graph from node N_0 to node N_{10}, which does not violate any of the constraints, represents one permutation of jobs, i.e., one solution of the problem. Thus, if the first node is job j_1 on machine i_1 then the ant k might have a choice of choosing j_2 or j_3 on machine i_1, allocating all corresponding jobs in machine i_1. The same job sequence is followed on all other machines.

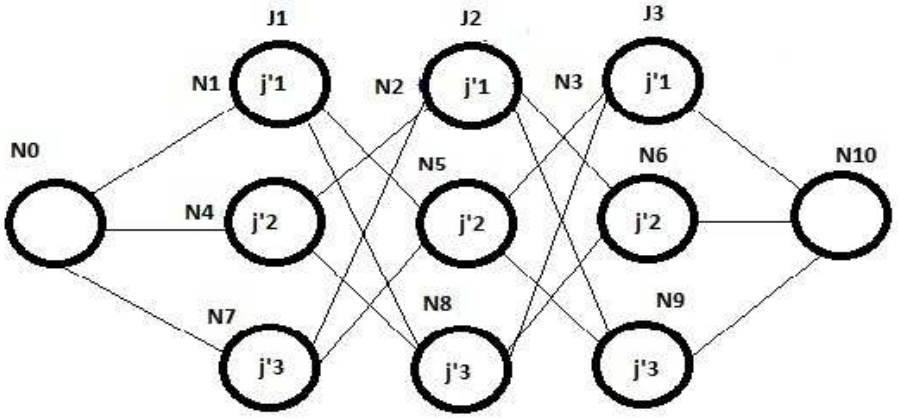

FIGURE 3.1: 3 set job sequence for machine 1(i1)

Initially, the memory is empty and is updated according to the ants movements through different nodes. The ant memory can be separated into three parts. Part one is the set of nodes not yet visited. Part two consists of the set of nodes that the ants are able to visit in the current iteration, to satisfy the constraints, and finally, part three contains a set of all nodes that have been visited so far. When the ant is placed at the initial node, it travels accordingly after completion of one cycle, each ant should have a sequence of nodes in group three that satisfies the constraints of visiting one node only, and the job machine ordering constraints too. The parameters for the ACO are as follows:

- k: The number of ants,

- $\lambda(i,j)$: The pheromone deposit on the edge (i,j),
- ρ: The pheromone evaporation parameter,
- α: The level of pheromone scent deposited by the current ants,
- β: The heuristic information determined by the ants,
- $\eta(j)$: The heuristic information stored at node j,
- N: The set of all nodes,
- V: The set of visited nodes,
- U: The set of unvisited nodes.

Starting from the initial node, each ant chooses the next node by using the transition probability rule, which uses the amount of pheromone deposited on its path and the heuristic information. Given U is the set of unvisited nodes, we consider an ant that has travelled to node i. The probability of choosing the next node j is given by;

$$P_k(i,j) = \begin{cases} \frac{[\lambda(i,j)]^\alpha . \eta(j)^\beta}{\sum_{j \epsilon U}[\lambda(i,j)]^\alpha . \eta(j)^\beta} & \text{if } j \epsilon U, \\ 0 & \text{otherwise.} \end{cases}$$

where $P_k(i,j)$ is the probability with which ant k chooses to move from node i to node j; $\eta(j) = 1/p(j)$, which is the inverse of the processing time of operations j, $p(j) = t_{ij}$ and j represents job j on machine i.

3.3.3 Pheromone Update

We have discussed the pheromone in chapter 2 under ACO section. Here we extend the pheromone update for the VJS problem. Once the ant completes the tour we use a procedure called pheromone update. The pheromone trail updates globally and locally. A global pheromone update, focuses on edges belonging to the trail traveled by the ant within the shortest time. Once the time is completed, the best ant deposits pheromone on visited edges, while the other edges remain unchanged. The amount of pheromone deposited, $\Delta\ \lambda(i,j)$, on each visited edge (i,j) by the best ant is inversely proportional to the length of the time of the tour. The shorter the time, the greater the amount of pheromone deposited on the edges. The global updating of pheromone is given by:

$$\lambda(i,j) = (1-\rho)\lambda(i,j) + \rho\Delta\lambda(i,j)^{best}$$

where,

$$\Delta\lambda(i,j)^{best} = \begin{cases} \frac{1}{L_{best}} & \text{if the best ant uses the edge (i,j) in its sequence,} \\ 0 & \text{otherwise.} \end{cases}$$

where ρ is the evaporating parameter, $0 < \rho < 1$. L_{best} is the length of the best sequence in the current iteration. The best sequence determines the amount of pheromone deposited in each iteration. Searching continues while the current pheromone level is below the previous value. The local pheromone update helps to avoid a strong edge being chosen by all the ants. This is done by changing the amount of pheromone update when an edge is chosen by a specific ant. The local update is given by

$\Delta\lambda(i,j) = (1-\rho)\lambda(i,j) + \rho\Delta\lambda_0$

where, $\Delta\lambda_0$ is a parameter which is set to 1.

Algorithm 12: ACO

```
Initialize all edges to pheromone level λ0 = 1;
Place each ant k on a randomly chosen job j on a particular machine i;
for each iteration t do
    while each ant k has not completed its set of jobs J do
        for each ant k do
            Move ant k to the next job j by the probability function Pk;
        end
        for each ant k with a complete set of jobs J do
            Evaporate pheromone ρ;
            Apply pheromone updates;
            if ant k time is the shortest then
                global pheromone update;
                Update global solution;
            end
        end
    end
end
```

3.3.4 Variable Neighborhood Search (VNS)

Variable Neighborhood Search (VNS) have been discussed in detail in Chapter 2, under VNS section in this thesis. For more details about VNS and its applications see e.g. [61, 19, 93, 91]. Here implement VNS for the VJS problem.

In our implementation, the solution of the problem is represented by two arrays:

- A sequence of jobs $j_1, j_2, ..., j_n$.
- The sequence of corresponding completion times, C_j.

Therefore, we present a solution s as $s = (\acute{J}, C)$. Our proposed algorithm is based on general VNS that comprises shaking and local search techniques. The proposed algorithm uses three neighborhood structures. The first is the 1-opt moves, the second is the insertion,

and the third is the swap. Below is the description of all these operations. Let s be the initial solution and $\ddot{s}$ the new solution after applying the respective neighborhoods, then:

- *1-Opt neighborhood*: A new solution is reached by interchanging two consecutive elements in the current array $(\acute{J}, C)$.

- *Insert neighborhood*: A new sequence is generated by inserting one job between any two other jobs. For example job 7 is inserted between jobs 4 and 5 to get a new sequence of jobs,$\acute{J}$.

Initial solution	3	4	5	6	7	1

After Insert	3	4	7	5	6	1

- *Swap neighborhood structure*: A new sequence is generated by interchanging the position of jobs in the sequence of jobs $\acute{J}$. For example, job 6 in the 4^{th} position is interchanged with job 3 which is in the 1^{st} position.

Initial Solution	3	4	5	6	7	1

After Swap	6	4	5	3	7	1

The swap neighborhood structure is used in the shaking step to improve the initial solution s i.e., neighborhood $N_k(s)$ is defined by consecutive swaps of jobs $\acute{J}$ followed by updating its corresponding values of C_j to get a better solution $\dot{s}$. In order to improve the solution $\dot{s}$, we use a local search step, consisting of a one opt neighborhood, by changing the consecutive jobs $\acute{J}$; then, we change the neighborhood to insertion by inserting job j_i between any two jobs j_u, j_v in the job sequence $\acute{J}$. At each neighborhood, a solution $\ddot{s}$ is generated and the best solution, if improved, is updated. The process of VNS pseudo code is presented below. The shaking is executed in step 4 with local search in steps in 5-16,

which are used to improve the solution; n represents the maximum number of loops.

Algorithm 13: VNS

```
1  Generate an initial solution s randomly;
2  Iteration t ← 0;
3  do
4      ṡ = Swap(s);
5      loop ← 0;
6      do
7          count ← 0;
8          max ← 2;
9          do
10             if count ← 0 then s̈ = Swap(ṡ);
11             if count ← 1 then s̈ = Insert(one − Opt(ṡ));
12             if f(s̈) ≤ f(ṡ) then count ← 0; ṡ ← s̈ ;
13             else count + +;
14         while count < max;
15         loop + +;
16     while loop < n(n − 1);
17     if (f(ṡ) ≤ f(s)) then s ← ṡ ;
18     t + +;
19 while stopping condition is not met;
```

3.3.5 Hybrid Algorithm

We propose a hybrid approach, ACO-VNS, that uses the ACO algorithm to generate the best/good solution, and then VNS to improve that solution.

We have proposed this approach because of the distinct features of the ACO and VNS. As discussed, ACO uses ants to construct a solution by exploring a search space and updating its pheromone using existing heuristic information. The wider,the ant explores the better the solution. It does not require any initial solution - ants are able to explore in the search space without it. But, on contrary, VNS requires an initial solution - it is able to find a better solution each time during a local search until a local optimum is reached. Hence, it would be beneficial to use an ACO-VNS approach, where ACO initially generates a good or best solution, which is then used as the initial solution for VNS. Since ACO does

not require an initial solution, a VNS-ACO approach may not be helpful as ants would still explore a search space discarding the initial solution produced by VNS. Therefore, it is better to use ACO first, and then improve initially by using various metaheuristic algorithms, such as VNS. This is the reason why we are not exploring VNS-ACO approach, and are focusing on the ACO-VNS approach.

Algorithm 14: Hybrid ACO+VNS

```
1  Generate solution from ACO algorithm 1;
2  Iteration t ← 0;
3  do
4      ṡ = Swap(s);
5      loop ← 0;
6      do
7          count ← 0;
8          max ← 2;
9          do
10             if count ← 0 then s̈ = Swap(ṡ);
11             if count ← 1 then s̈ = Insert(one − Opt(ṡ));
12             if f(S̈) ≤ f(ṡ) then count ← 0; ṡ ← s̈ ;
13             else count + +;
14         while count < max;
15         loop + +;
16     while loop < n(n − 1);
17     if (f(ṡ) ≤ f(s)) then s ← ṡ ;
18     t + +;
19 while stopping condition is not met;
```

3.4 Computational Results

We have used an Intel core dual i7 processor, 3.4GHz with windows 7, 64-bit operating system with 16GB RAM. The ACO and VNS have been developed in the $C++$ language. As, our problem is new, we are unable to find any benchmark instances.

We have used instances from other scheduling problems used by various researchers. As discussed in chapter 2, the scheduling problem can be classified into various types. We have carefully chosen 14 benchmark instances from different types of scheduling problems

so that our results are constructed in a efficient way. We have restricted ourselves to basic scheduling problems, the processing times are fixed, there are neither set-up times nor due dates nor release dates, etc. The selected instances are used in other scheduling problems.

- Fisher and Thompson [51], have used a genetic algorithm that focuses on initial population to obtain an optimal or near optimal solution. They have introduced 66 benchmark instances. We have chosen the 10x10 instance.
- Lawrence instances [77], have investigated heuristic scheduling techniques for constrained project scheduling problems. He has given various benchmark instances. We have chosen 15x15, 30x10, 15x10, 20x5 of his instance for our problem.
- Storer, Wu and Vaccari [132], have proposed two methods both of which are based on novel definitions of solution spaces and of neighborhoods in these spaces. The proposed methodology are developed for job shop scheduling problems. They have further given various benchmark instances and we have chosen 20x10, 20x15, 50x10 instances.
- Talliard [137], have proposed 260 randomly generated scheduling problems, their size is greater than that of the rare examples published. These sizes correspond with real dimensions of industrial problems. The types of problems that are proposed are: the permutation flow shop, the job shop and the open shop scheduling problems. We have chosen 100x5,100x10, 100x20, 200x10, 500x20 instances.
- Yamada and Nakano [148], have implemented genetic algorithm to solve large scale scheduling problems and have given various benchmark instances. We have chosen 20x20 instance.

The following section contains the parameter setting for the ant system and the results discussion.

3.4.1 Setting Parameter Values for ACO

The choice of parameter values of the ACO algorithm play an important role in the quality of thefinal solution. The ACO algorithm literature proposed by Liouane et al. [83], reviews various parameter values. In VJS we set various parameter values accordingly to obtain the best solution. We only consider the values in the range detailed in [83].

- α determines the quantity of pheromone deposited by the ants when they build their solution. The higher the value, the more restrictive the ants ability becomes to exploring new sequences. The lower the value, the more the ant is able to travel to more unvisited edges, but it uses more computational time. From literature, it has been found the value to be in the range of 1-10, and overall, the algorithm performs best with a value of approximately 1 depending on the problem. We have used 1. [83].

- β determines the heuristic information used by the ants. The value ranges between 1- 4. It has been observed from the literature review the value 1 might give the best results. [83].

- ρ is the pheromone evaporation parameter. A higher value enables the ants to carry out more searches. It has been noted that the value ranges between 0.1 - 0.7 for obtaining good solutions. we have used 0.7 [83].

- k defines the number of ants belonging to the colony. There is a trade-off between creating too many sequences, namely too many ants, versus fewer sequences namely fewer ants. If the number of ants is low, the algorithm speeds up because of fewer searches; in contrast, if the number of ants is high, the algorithm slows down because of more search, but many more sequences are created with the scope of finding a better solution. Our experiments suggest that the number of ants in the colony is best to be between the ranges of 8 - 10. We have used 10 ants for our entire job instances [83].

3.4.2 Results

We have computed the solutions for ACO, VNS, ACO+VNS and CPLEX. Three heuristic methods are run for 1000 iterations and the best solution obtained is reported in the Table 3.4, with computational time in brackets. We have computed the results for the ACO parameters α and β parameter while ρ and the number of ants are kept constant at 0.7 and 10 respectively. The results are computed for each 14 benchmark instances that has been selected through various literatures.

Parameter	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	11392	11505	12657	14175	15876	17781	19915	22305	24982	27980
(sec)	331	328	311	296	281	235	254	241	229	217
β=2	12759	12887	14175	15876	17781	19915	22305	24982	27980	31337
(sec)	291	288	274	260	247	235	223	212	201	191
β=3	14290	14433	15876	17781	19915	22305	24982	27980	31337	35098
(sec)	326	323	307	291	277	263	250	237	226	214
β=4	16005	16165	17781	19915	22305	24982	27980	31337	35098	39309
(sec)	365	362	344	326	310	295	280	266	253	240

TABLE 3.2: [20] $\times$ [5] ACO

Table 3.2 represents the ACO results for various values of α and β. The solutions are reported for each β value in the row and its corresponding computational times are reported in sec. In this table, we clearly see that the best solution is when α and β are 1. In this case, the pheromone trail α is less and the evaporation rate ρ is set constant at 0.7. The pheromones are evaporated in such a way that the ants are able to explore more search space, because of less pheromone attraction to the ants, while the available heuristic information β is less. Further, because of the ants exploration in the wider search space, the computation time is high. We can see that the worst solution is when α and β are 10 and 4 respectively. In this case, the pheromone trail α is high and the evaporation rate ρ is set constant at 0.7. The pheromones are evaporated in such a way that the ants are unable to explore more search space, because of more pheromone attraction to the ants, while the available heuristic information β is high. Further, the ants exploration is minimised in the search space, meaning less exploration resulting in less computational time.

When α is less and β increases, we see that the solution and the computational time increase. This is because the ants may not be able to explore the wider search space, as the heuristic information have increased, even though the pheromone rate is less. This has resulted in more computational time and high solution. But on the contrary, when β is less and α increases, we see the computational time decreases and the solution increases. In this case, the ants are restricted to less exploration in the search space because of high concentration of pheromone, which leads to high solution and less computational time. Please refer, Appendix A for further results on other 13 benchmarks.

Parameter	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	10883	10992	12091	13300	14630	16093	17702	19473	21420	23562
(sec)	9844	9746	9063	8429	7839	7290	6780	6305	5864	5453
β=2	12080	12201	13421	14763	16239	17863	19650	21615	23776	26154
(sec)	8860	8771	8157	7586	7055	6561	6102	5675	5278	4908
β=3	13530	13665	15032	16535	18188	20007	22008	24208	26629	29292
(sec)	9923	9824	9136	8496	7902	7349	6834	6356	5911	5497
β=4	15153	15305	16835	18519	20371	22408	24649	27113	29825	32807
(sec)	11113	11002	10232	9516	8850	8230	7654	7118	6620	6157

TABLE 3.3: [20] $\times$ [5] ACOVNS

Table 3.3 represents the ACOVNS results for various values of α and β for 20×5 instance. For other instance please refer, appendix B. In the hybrid algorithms ACOVNS, we use the ACO results as the initial solution for VNS. The shaking logic explores the local neighborhood to improve the initial solution or to obtain the best solution, if there is any improvement then the initial solution is replace by new solution and further exploration is done to obtain any new solution that is better than the pervious solution. This exploration in the neighborhood find the best solution and as a result increases the computational time. In table 3.3, we clearly see, the solution improves at every stage of α and β, when compared with Table 3.2, this is because of VNS shaking trying to find the best solution. Further, we see the computational time increase when α and β increases when compared to Table 3.2. This is because when α and β increases, the initial solution given by ant may not be the best solution, and VNS take significant computational time to find the best solution.

Constraint	Variable	Instance	ACO	VNS	ACO+VNS	CPLEX
7320	400	[20] × [5]	11392 (331)	10883 (9654)	10883 (9844)	10883 (46020)
910	100	[10] × [10]	3211 (30)	3205 (9321)	3205 (9735)	3205 (1170)
3090	225	[15] × [10]	7637 (35)	7530 (9638)	7530 (9975)	7530 (25522)
3165	225	[15] × [15]	6958 (96)	6870 (9644)	6870 (9980)	6870 (31278)
7420	400	[20] × [10]	11794 (238)	11341 (9875)	11341 (10677)	11341 (2067290)
7520	400	[20] × [15]	12402 (351)	12021 (9967)	12021 (10831)	NA
7620	400	[20] × [20]	6664 (181)	6486 (9998)	6486 (11456)	NA
25530	900	[30] × [10]	24824 (461)	23268 (14771)	23268 (16543)	NA
120550	2500	[50] × [10]	68243 (2538)	62841 (65244)	62781 (68022)	NA
980600	10000	[100] × [5]	263505 (13205)	233971 (302640)	233944 (351108)	NA
982100	10000	[100] × [10]	267499 (20193)	242959 (1041607)	242930 (1053421)	NA
981100	10000	[100] × [20]	274356 (16010)	248109 (566441)	248025 (598007)	NA
7922200	40000	[200] × [10]	1056487 (122576)	931889 (4930348)	931429 (5035668)	NA
124510500	250000	[500] × [20]	6418210 (672757)	5847759 (19968785)	5846398 (20851539)	NA

Table 3.4: Best solutions for VJS instances

In Table 3.4, we have provided the best solution from ACO and ACOVNS. We have also produced the solution from VNS and Cplex.

(i) In terms of solution quality, it is clear that ACO + VNS performs better than the other algorithms, especially for larger problem instances. However, it uses more computational effort than our general VNS algorithm;

(ii) CPLEX is unable to find results for large instances;

(iii) VNS outperforms ACO in all instances significantly,but uses more cpu time.

The result shows the computation time is high. One of the reasons could be our computer system that has basic configuration with basic RAM and processing speed. we believe an higher processing speed will definitely improve the computation time. Further, we have used a single thread processing, if we were able to use the additional cores by developing parallel processing this may reduce the computation time.

3.5 Summary

In this chapter, we observed that the first phase of the known assembly scheduling problem [110], may have its own practical implementation, i.e., when the second assembly phase is omitted. We call the first phase, proposed by Patkar et al. [104], a vector scheduling problem. We have proposed a new mathematical programming formulation and solved it with a commercial solver CPLEX. Further, we have presented four different types of computational techniques to solve the VJS problem: (i) an exact method based on our mathematical programming formulation; (ii) a heuristic method based on ACO meta-heuristic; (iii) a heuristic method based on VNS and finally (iv) as a hybrid of ACO that uses VNS as the local search method. Our goal was to identify the performance of these techniques for the vector job scheduling problem. The performance analysis was carried out by using the proposed method with our data sets and the results were compared. Our results show that our newly constructed hybrid algorithm ACOVNS that uses ACO to identify the best solution, which is used as an initial solution for VNS. Then, VNS uses shaking logic to find the best solution in their respective neighborhoods when compared to standalone algorithms liks ACO and VNS. Even though ACOVNS performs well, it depends upon setting the right parameter values and selecting the number of ants for each colony. From our analysis, we have shown that the pheromone trail updates between the nodes plays a vital role in constructing the best solution.

Chapter 4

Financial Derivative Hedging

4.1 Introduction

A CFD, or Contract for Difference, is a leverage derivative product that allows speculation/trade on price movements of the underlying instruments such as commodities, market indices, shares etc. It is initiated by entering into a contract at an opening price of an underlying instrument and betting on whether the price of that instrument increases or decreases. To bet that the price increases then you would 'go long' by buying the CFD expecting the underlying instrument price to gain in value. If you are betting that the price decreases then you would 'go short' by selling the CFD expecting the underlying instrument price to lose its value. In, either case when the contract is closed, depending on the price difference between the open and close of the contract you make either a profit or a loss.

CFD is a leveraging product. A contract can be bought for a fraction of the market value of the underlying instrument. This fraction can be as small as 1%. The rest is covered by the CFD broker. Even though only a fraction of the market value has been deposited it can still be possible to gain 100% profit or loss when closing the contract if bought.

For example, Company A share price is x. If a buyer, buy a long CFD, say N shares of company A, then the buyer pays a fraction (1 %) of $N.x$, with an expectation the share price of company A increases. If the price increases from x to $x + u$, then the seller of the CFD will pay the buyer the price difference $u.N$. However, if the price decreases then the buyer pay the seller at the close of the contract.

FIGURE 4.1: CFD Structure(ws-alerts.com)

The buyer and seller of the CFD will enter into a contract. The buyer of the CFD is generally classified as the counter party (client) and the seller as a broker. In theory, a contract is opened when the broker sells a CFD to the counterparty/client. The broker (seller) is expected to buy or hold the equivalent number of shares (underlying). Similarly, the contract is closed when the counterparty (buyer) sells back the CFD to the broker. When the broker buys the CFD, the broker can go and sell the equivalent share (underlying) in the market.

Regulators around the world especially in Europe have an objective to ensure the financial market works well and to improve market integrity. European regulation has established market abuse regulation to increase market integrity and investor protection. As a result monitoring the market across all regulated asset classes has been a key functionality for regulators.

In this chapter, we will be analyzing the CFD derivatives for underlain equity market. We use intraday transaction data from January 2013 until January 2015 that has been reported to one of the key European regulators. The transaction report contains all intraday transactions that have taken place across various trading platforms. Our aim is to establish the corresponding match for CFD with its underlying equity hence to detect any

unhedged CFD with its underlying equity. We have developed two different local search methods and embedded them into BVNS to generate new variants, BVNS-LS- Type1 and BVNS-LS-Type2 to find a better solution.

4.2 Problem Example

We illustrate our problem with a simple example. In figure 4.2, we have 6 trades and 5 CFD's that contains volumes and prices. Our aim is to match all the 5 CFD with the trades and list all the unmatched CFDs, in case, the trades are not matched. Once trades are matched to a CFD they cannot be reused for any other CFD.

FIGURE 4.2: Example Trade and CFD's

In our illustrative example, we generate an initial solution set of trades with volume and price. We then use the first and the second neighborhood structure as our shaking procedure. In figure 4.3, We can see Volume 1000 and 3000 are removed and volume 200 and 100 are added. Similarly, their corresponding prices 5.35 are removed and 5.10 are added. We calculate the trade volume 300 by adding trade volumes 200 and 100 since their mean price is 5.10, which exactly matches the first CFD volume 300 and price 5.10. Therefore, the trade volume 100, 200 and their corresponding price 5.10 will be matched against CFD volume 300 and price 5.10. But our aim is to find the unmatched CFD, hence the matched volume and price for trades are removed, and the rest of the trade volume and price are considered for the next CFDs.

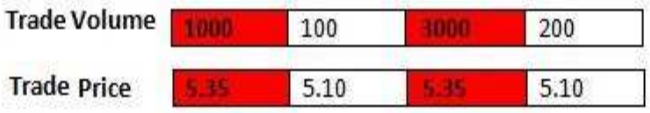

FIGURE 4.3: Example

By repeating the above shaking procedure for each CFD's in our example we get the below results, after the shaking procedure.

- CFD volume 300 and price 5.10 are matched with two TRADE volume (100 + 200) and its price 5.10.
- CFD volume 1500 and price 5.35 are matched with one TRADE volume 1500 and its price 5.35.
- CFD volume 1100 and price 5.35 are unmatched with any TRADE, even though the price 5.35 matches with the CFD price but the volume 1000 and 3000 do not match.
- CFD volume 200 and price 5.11 are unmatched with any TRADE, even though the volume 200 matches with the volume of CFD, the price 5.12 does not match.
- CFD volume 3000 and price 5.45 are unmatched with any TRADE, even though the volume 3000 matches with the volume of CFD, the price 5.45 does not match.

In order to improve our solution for the shaking procedure, we use our two newly developed algorithms as our local search procedure. We concentrate more on the unmatched CFD's. We have tuned our local search algorithm to identify the reason for the unmatched CFD's. This reason could be either an over/under volume or over/under price. Further, we would match the best possible trades in order to have the minumum mismatch value for the unmatched CFD's. Our example Figure 4.4, represents the list of unmatched CFD's and the best possible trade match that has minimum mismatch value with the reason for the unmatch.

Unmatched CFD

Volume	Price	Reason
1100	5.35	Under Volume by 100
200	5.11	Over Price by 0.01
3000	5.45	Under price by 0.10

FIGURE 4.4: Unmatched

- In the above table,CFD volume 1100 is unmatched, the reason is the under volume trade of 1000. Hence, the minimum difference is 100 or the match is missed by 100.
- In the above table, CFD price 5.11 is unmatched, the reason is the over price trade of 5.12. Hence, the minimum difference is 0.01 or the match is missed by 0.01.
- In the above table, CFD price 5.45 is unmatched, the reason is the under price trade of 5.35. Hence, the minimum difference is 0.10 or the match is missed by 0.10.

4.3 Data

The data used is provided by one of the key European regulators. The transaction data is the implementation of transaction reporting that has been described in the Markets in Financial Instruments Directive (MiFID). It is an obligation for trading firms to report their trades to their local regulators that have been set out in the office of the journal of the European Union, Commission Regulation EC No. 1287/2006 (Article 13/Annex 1). Firms must report transactions when they execute a trade that is reportable. The report must contain mandatory details of their transactions by the end of the following business day (T+1) as specified in (Article 13/Annex 1). Transaction reports received from firms are loaded into the transaction monitoring system. The purpose of the transaction reporting is to detect and investigate suspected market abuse and also to maintain confidence in financial markets and reduce financial crimes.

Our primary analysis is based on intraday transaction data for all the FTSE 100 stocks over the period January 2013 until January 2015 in the UK equity market. For our analysis we have considered only the CFD transaction data. The transaction data is reported on a stock-by-stock basis that consists of all the executed trades across multiple regulated

platforms and are reported in seconds. Further, we have used the price mode function that would convert the price currency to GBP in case they were reported in a different currency.

4.4 Mathematical Programming Formulation

Our problem is to find the mismatch CFD's from a given set of CFD with its corresponding trades. We structure our mathematical model in a way, to identify the mismatch CFD that has the minimum cost in a given set of CFD's with its respective trades. The identified minimum cost CFD's may not be the actual mismatch but it gives a guarantee result that there is an unmatched CFD in the given CFD set.

Sets

- $C = \{1, 2, \ldots, c, \ldots, n\}$ denotes a set of CFD's,
- $T = \{1, 2, .., i, ., t\}$ denotes a set of trades.

Data

- V_t be the volume of trades.
- P_t be the price of trades.
- $\acute{V}_c$ be the volume of CFD.
- $\acute{P}_c$ be the price of CFD.
- W^+ be the weights on volume of CFD.
- W^- be the weights on price of CFD.

Variables

- $\dot{V}_c$ be the over volume of CFD.
- $\ddot{V}_c$ be the under volume of CFD.
- $\dot{P}_c$ be the over price of CFD.
- $\ddot{P}_c$ be the under price of CFD.

$$Y_{tc} = \begin{cases} 1 & \text{if trade } t \text{ is used in balancing CFD } c, \text{ t } \epsilon T, \ c \epsilon C \\ 0 & \text{otherwise.} \end{cases}$$

The complete mathematical programming formulation can be written as:

$$Minimize \sum_{c=1}^{c} (\dot{V}_c W^+ + \ddot{V}_c W^+ + \dot{P}_c W^- + \ddot{P}_c W^-) \tag{4.1}$$

subject to

$$\sum_t V_t Y_{tc} = \acute{V}_c + \dot{V}_c - \ddot{V}_c, \ \forall c \tag{4.2}$$

$$\sum_t V_t P_t Y_{tc} = \acute{V}_c(\acute{P}_c + \dot{P}_c - \ddot{P}_c), \ \forall c \tag{4.3}$$

$$\sum_c Y_{tc} \leqslant 1, \ \forall t \tag{4.4}$$

The objective function is to minimize the total CFD mismatch. The constraint group(4.2) determines the over and under volume.(4.3) determines the over and under volume with price of the CFD and (4.4) specifies that a trade can be used for at most one CFD.

4.5 Basic Variable Neighborhood Search (BVNS)

We have discussed several variants of VNS in Chapter 2. In our matching problem, we will be using Basic Variable Neighborhood Search (BVNS) [90]. It uses a process to find the next optimal solution from the most fitting neighborhood structure, then the solution is further refined and improved by using a local search technique. This improved solution will be the current solution from the neighborhood in the iteration. This process will provide a good solution and save computational time without analysing the full neighborhood structure.

Our proposed BVNS initially generates a random solution s, then it uses two neighborhood structures namely Remove Fill and Add Remove as a shaking procedure to generate a solution $\dot{s}$, and a local search to improve the shaking solution $\dot{s}$ as input solution to get a newly improved solution $\ddot{s}$. We then compare $\ddot{s}$ solution with the s in term of objective function. If there is an improvement, we replace the current solution s with $\ddot{s}$. We define the stopping criteria as a maximum number of iterations as 500 for shaking and local search.

Let us assume k neighborhood structures $N_1, N_2, ., N_{kmax}$. The process starts with the initial solution s. Performing the shaking procedure for local changes in the neighborhood

we can obtain a better solution $\dot{s}$ from $N(s)$. Later, we perform a local search procedure with a different neighborhood until a local optimum is obtained. Below is the general working algorithm for BVNS:

Algorithm 15: BVNS

Initialization: select the neighborhood structure sets $N_k, k = 1, 2, ..., k_{max}$;
Generate a random initial solution s;
Set $k = 1$;
Repeat the following steps until $k = k_{max}$;
Shaking: generate a point $\dot{s}$ randomly from $N_k(s)$;
LS: implement Local search method to obtain local optimum $\ddot{s}$ from $\dot{s}$;
if $\ddot{s}$ *is better than* $\dot{s}$ **then** set $s = \ddot{s}$ and $k = 1$;
else $k = k + 1$;
stop ;

4.5.1 Neighborhood Structure

Initially, group the trades into various random subsets. We match these subset groups to the CFD. We define C as the set of trades that are considered to match each CFD and $\overline{\mathrm{C}}$ is its complement. We remove and add certain trades to match the CFD in order to obtain the solution s. We use two type of neighborhoods to perform the shaking.

- The First neighborhood N_1k, Remove and Fill, here trade volume V_t and trade price P_t, are removed from C and V_t, P_t are added to $\overline{\mathrm{C}}$ to get a better match.
- In the second neighborhood N_2k, , Add and Remove, here trade volume V_t, trade price P_t that are not in $\overline{\mathrm{C}}$ are added to match the CFD by removing the combination of trades V_t, P_t from C.

4.5.2 Local Search Neighborhood

We construct two new search algorithms in order to improve the solution. We implement these algorithms as local search procedures in BVNS. We then compare the results of these two local searches and report the solutions in the results.

4.5.3 Search Type-1 (STYPE-1)

In this approach, we improve the solution by matching the trades to the CFDs to generate new solutions that contain various CFD mismatches. We weight these mismatches with a cost function. Later, we start with the worst CFD cost to re-match the CFD with the trades, to either attain a better solution or achieve the same solution. Below is the pseudo code for the STYPE-1.

Algorithm 16: STYPE-1

```
1  Initialize Set T = 0;
2  Set Ť = ∞;
3  Set O_i = i;
4  for each i in CFDs do
5  |  solve(O_i);
6  |  Let E_{O_i} = Z;
7  |  T = T + Z;
8  end
9  if T ≥ Ť then break ;
10 else Ť = T ;
11 Let Ó_i, É_i = sort(E_i)and O_i;
12 Let O_i = Ó_i;
13 Display Ť;
```

- Line 1: Initialise set $T = 0$,
- Line 2: Initialise all set of trades $\acute{T}$ to maximum,
- Line 3: O_i is the ordering of disaggregated sequence of each CFD's.
- Line 4-8: For each CFD, i, we solve to match the corresponding CFD to the set of trades. we capture the mismatch CFD and its corresponding trades in the new order sequence E_i
- Line 9-10: We break, if there is no mismatch.
- Line 11-13: Depending on the mismatch cost, we sort and rematch the trades T to CFD's, in order to obtain a better sequence E_i or to minimise the CFD mismatch cost.

Algorithm 17: BVNS-STYPE-1

Initialization: select the neighborhood structure sets $N_k, k = 1, 2, ..., k_{max}$;
Generate a random initial solution s;
Set $k = 1$;
Repeat the following steps until $k = k_{max}$;
Shaking: N_1k, N_2k, ;
Local Search: STYPE-1;
if $\ddot{s}$ *is better than* $\dot{s}$ **then** set $s = \ddot{s}$ and $k = 1$;
else $k = k + 1$;
stop ;

4.5.4 Search Type-2 (STYPE-2)

We reconstruct another search algorithm, STYPE-2, to solve each CFD independently. Any trades used in matching the CFD are not available for the next CFD. If no mismatch is found we exit otherwise, we sort the CFD mismatch from the largest to the smallest. This constitutes the new ordering. This process is repeated until either no mismatch is achieved or the current sequence has occurred previously. In which case, we know that we have a mismatch and we also know its minimum value. Below is the pseudo code for the STYPE-2.

Algorithm 18: STYPE-2

Initialize Set $c = 0$;
for *each* i *in* CO_i **do**
    Z= solve(CO_i);
    $CS_i = Z$;
end
if $\sum_{j=1}^{c} CS_i = 0$ **then** break ;
else sort(CS_i, CO_i) ;
for k *in* 1 *to* $c - 1$ **do**
    if $CO_i = OO_k$ **then** break ;
    else sort(CS_i, CO_i) ;
end

- Line 2-5: CO_i is the ordering of disaggregated sequence of each CFD's. Z is the objective function of solving the disaggregated CFD's. For each CFDs, We solve CO_i to get the unmatched CFD and its corresponding trades, CS_i
- Line 6-7: If CS_i is zero we break and exit the loop. We then sort the CS_i to find the worst cost of the unmatched CFD.
- Line 8-10: If there is a match we break, if not we repeat the process until we find a better solution.

Algorithm 19: BVNS-STYPE-2

1 Initialization: select the neighborhood structure sets $N_k, k = 1, 2, ..., k_{max}$;
2 Generate a random initial solution s;
3 Set $k = 1$;
4 Repeat the following steps until $k = k_{max}$;
5 Shaking: N_1k, N_2k, ;
6 Local Search: STYPE 2;
7 **if** $\ddot{s}$ *is better than* $\dot{s}$ **then** set $s = \ddot{s}$ and $k = 1$;
8 **else** $k = k + 1$;
9 stop ;

4.6 Numerical Results

Our experiment is performed on an Intel core i5 processor, 3.20GHZ, windows 7 with 64 bit operating system. We collected the transaction data from the European regulator. Clearly, we are only interested in solving the problem with more than one CFD. The reason for including a problem set with one CFD is to validate our results. We deliberately included data sets that contain unhedged data and grouped the data to increase the trade size. We used AMPL as our coding language and our intention was to solve these problems using solver.

Constraint	Variable	Problem Instance	Trades	CFD	BVNS-STYPE-1	BVNS-STYPE-2	CPLEX
26847	214680	$new-CFD-1$	26831	8	0.00249122 (68.569)	0.00444 (39.39)	0.00444 (2278.63)
11996	47968	$new-CFD-2$	11988	4	0.00084 (25.191)	0.00084 (8.127)	0.00084 (727.511)
6634	39756	$new-CFD-3$	6622	6	0.067 (18.928)	0.0622 (13.135)	0.0622 (10687.1)
12485	112239	$new-CFD-4$	12467	9	0.0417 (48.4523)	0.0415 (68.843)	0.0418 (10803)
53659	321906	$new-CFD-5$	53647	6	0 (34.32094)	0 (13.712)	0.00025 (10378.1)
10454	114796	$new-CFD-6$	10432	11	0.00476 (36.959)	0.00279 (72.228)	0.01016 (10197.5)
1146	7952	$new-CFD-7$	1132	7	0 (10.5535)	0 (4.992)	0.0000 (10408.2)
598096	2392368	$CFD-9$	598088	4	0.00044 (2973.8)	0.000346 (1258.66)	0.000946 (3859.59)
9219	46065	$CFD-6$	9209	5	0 (702.8)	0 (702.8)	0.00515 (10659.8)
629	1881	$CFD-11$	623	3	0.000485 (12.4937)	0.000485 (7.846)	0.000485 (655.625)

TABLE 4.1: Best Solution for CFD-Trades Matching

Table 4.1, contains the results which represents the problem size, the total number of trades in an instance, the total number of CFD, the optimal CFD error cost for each problem instance and the CPU seconds are given in brackets for BVNS-STYPE-1, BVNS-STYPE-2,CPLEX with variable and constraints. We have compared the results between two different search approaches. We found that type 2 search is more efficient than type 1 search. In all instances, we have validated our results in the following ways. For every problem we have fixed the binary variables to the values determined by our algorithm and then solved. After this solve, we unfix our assignments and resolve using the previous optimal basis as a warm start for this resolve. For all the problems these two separate solutions have produced the same results as our local search STYPE-2 algorithm produced. Because of the combinatorial nature of these problems we decided not to parameterize benchmark results. We report the best solution found by CPLEX. The stopping criteria is a mixed integer solution limit of 100 and a default time limit. In the majority of cases CPLEXwas unable to find a better solution, in case if a solution is found the time is extensive.

CFD	Match	Trades
1	Y	11
2	Y	22
3	Y	1
4	Y	7

TABLE 4.2: new-CFD-2

CFD	Match	Trades
1	Y	11
2	N	5
3	Y	5

TABLE 4.3: new-CFD-11

Table 4.2 and 4.3 contains the results of individual problem instance. The 'CFD' column represent the number of CFD to be matched. The 'Match' column represent whether the CFD volume and price are matched with its corresponding trades, Y indicates a successful match and N indicates an unsuccessful match. The 'Trade' column represent the total trades used to match the CFD. In case, they are unmatched, the trades represents the closed mismatch of the CFD.

In table 4.1 , we see that the BVNS STYPE-2 produces a better solution and efficient computational time when compared to BVNS STYPE-1. This is because BVNS STYPE-1 matches the trade to each CFD from its corresponding CFD set. If any of the CFDs are mismatched, the process of rematching all trades to their CFDs restarts, and continues until all CFDs are matched or the CFD mismatch cost has a better solution. But, on the contrary, using BVNS STYPE-2, if we match the trades to corresponding CFDs, we discard the matched trades and its corresponding CFDs from their respective sets. We then, continue to match the remaining trades to the remaining mismatch CFDs, and repeat this process the mismatch CFD cost is minimised.

Let us consider Table 4.2, where all CFD is matched. We see that there are 4 CFDs with total number of trades 11988. Using BVNS STYPE-1, we consider all 11988 trades to match with each CFD. In the table, we see the CFD_1 is matched with 11 trades. We now discard this CFD_1 and the 11 trades, and choose CFD_2 and try to find the match with the remaining 11977 trades. As there are 22 trades being matched, we further discard these 22 trades and CFD_2. We now select CFD_3 and match with the remaining 11955, as there is match with one trade, we discard this trade and its corresponding CFD_3. We finally select CFD_4 and match with the remaining 11954 trades which matches with seven trades. Now, considering BVNS STYPE-2, it follows the same process as for BVNS STYPE-2. Hence, if all CFDs are matched, the processes of BVNS STYPE-1 and BVNS STYPE-2 are the same as mismatch cost function is not involved.

lets us consider Table 4.3, where there is one mismatched CFD. In this table, we see that there are three CFDs with the total number of trades 623. Using BVNS STYPE-1, we consider all the 623 trades to match with each CFD. In the table, we see that CFD_1 is matched with 11 trades. We now discard this CFD_1 and the 11 trades, and consider CFD_2 and try to find the match with the remaining 612 trades. We see that we are unable to match this CFD_2, which is a mismatch, and a mismatch cost is associated with this CFD_2 mismatch. In this scenario, the previous matched trades and CFD_1 is discarded and CFD_2 is considered with all trades, 623 to be matched. But, still CFD_2 would not be able to match, which would give a mismatch cost. CFD_2 with minimum mismatch cost is considered, that are associated with five trades which will be discarded, when CFD_3 is considered, which is matched with 6 trades and, similarly, CFD_1, which has been matched with 11 trades.

Now, using BVNS STYPE-2, we consider all 623 trades to match with each CFD. In the table, we see that CFD_1 is matched with 11 trades. We now discard this CFD_1 and these 11 trades. We then consider CFD_2 and try to find the match with the remaining 612 trades. We see that we are unable to match this CFD_2, which is a mismatch and a mismatch cost is associated with this CFD_2 that consist of 5 trades. Further, we try to match CFD_3 with the remaining 607 trades. We see that 6 trades are matched, and we discard CFD_3 with six trades. We reconsider the mismatch CFD_2 and try to match with the remaining trades 606, where we have eliminated the trades that matched CFD_1 and CFD_2. If CFD_2 is still mismatch, there will be a cost associated with it, we will choose the minimum cost associated trades, which is 5 in this case.

4.7 Summary and Conclusion

We have introduced a neighborhood structure for shaking and two different local search approaches for the local search technique. We have combined each of these local search types with our shaking neighborhood and attained two new Variable Neighborhood Search (VNS) variants for these types of matching problems. We further compared the results of these two search methods. From our comparison the STYPE-2 local search approach identifies the most optimal CFD-trades mismatch and minimise the cost with efficient computational time. While STYPE-1 has taken significant time to find the solution, this is because of the continuous rematching of the trades with the CFD's.

Chapter 5

Conclusion

5.0.1 Overview

This thesis focuses on the metaheuristic algorithms used to resolve complex combinatorial problems. During the research, wehave concentrated on two major metaheuristic algorithms:

- Ant Colony Optimization (ACO).
- Variable Neighbourhood Search (VNS).

We have derived the problems from the manufacturing/scheduling and financial sectors, and they are relevant to current real-time application in these industries.

Our research have introduced three new algorithms metaheurisitc algorithms. We have implemented our newly developed algorithms to address two different combinatorial problems,and compared our results with other metaheuristic algorithms. Our results, shows that our newly constructed algorithms, produced more efficient results in solving these type of complex combinatorial problems that have multiple constraints when compared to the stand-alone metaheuristic algorithms such as VNS and ACO.

5.0.2 The Hybrid Algorithms

In this thesis, we described a complex combinatorial problem VJS, and implemented biological algorithm ACO and neighborhood algorithm VNS. These two algorithms have their

own unique features - ACO uses ant to explore the space and VNS needs a random initial solution, and then uses shaking logic that performs local changes in the neighborhood to obtain the best solution.

We combined these two unique features of ACO and VNS into one hybrid algorithm. To construct this hybrid algorithm, we used ACO feature to find the initial solution, for VNS, then later VNS uses shaking principle to find the best solution

The results indicate clearly that the hybrid algorithm ACOVNS is more effective than stand alone algorithms such as VNS, ACO. The VJS is a complex scheduling problem faced by various manufacturing industries. Our research provides a new hybrid algorithm that achieves the best solution to resolve similar combinatorial problems faced by various scheduling industries.

5.0.3 VNS Variants

Further in our research, we have considered a financial derivative problem. It is similar to a goal programming problem that has been faced recently faced by financial regulators. Our aim was to identity the mismatch between the trades and CFD's. We have used VNS as our base algorithm, and further extended by constructing two new local search methods. This led to construction of two new VNS variant algorithms.

Our newly constructed VNS variant algorithms, has been implemented in the financial sector. Our results show that our algorithms are able to provide best solutions by identifying mismatch trades, and, further, are able to find the closest mismatch. These algorithms could solve similar combinatorial problems faced by organisations in the financial sector, especially that focus on market surveillance and monitoring area in order to maintain market integrity.

5.0.4 Future Work

There are so many challenges to address these kind of combinatorial problems,especially in the scheduling and financial derivatives sectors. To expand several new research directions and further developments, these are the possible proposals that could be considered and investigated:

- To further analyse, the ACO parameters α, β, ρ and to explore the heuristic information could help in more efficient exploration by ants through local and global pheromone updates.

- To investigate other optimisation algorithms such as Partial Swarm Optimization (PSO) algorithms, which could be combined with VNS to construct a new hybrid algorithm that may be able to reduce the computational time by exploring the search space in an efficient manner.

- To explore other variants of VNS, especially Parallel Variable Neighborhood search (PVNS) that may be able to reduce computational time or may increase exploration in the search space especially for large contraint scheduling job shop problems.

- To examine the new VNS variant, Variable Neighborhood Program(VNP), that acts like an artificial intelligence algorithm similar to machine learning algorithm. The VNP could be able to understand the characteristic of data and may be able to generate/determine some key relationship among the data, that could enable to solve the problem, especially in financial sector that deals with high volume of data.

Appendix A

Results ACO for VJS

The results of ACO for parameter α and β

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	3211	3275	3668	4105	4603	5162	5789	6491	7279	8163
(sec)	30	27	26	24	22	21	19	18	17	15
β=2	3596	3668	4108	4597	5156	5782	6483	7270	8153	9142
(sec)	27	26	25	22	21	19	18	17	15	14
β=3	4028	4108	4601	5149	5774	6475	7261	8143	9131	10239
(sec)	26	23	22	20	19	17	15	13	12	11
β=4	4511	4601	5154	5767	6467	7252	8133	9120	10227	11468
(sec)	24	22	21	19	18	16	14	13	11	10

TABLE A.1: [10] × [10] ACO

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	7637	7828	8957	10467	12226	14275	16660	19438	22671	26434
(sec)	35	33	31	30	28	25	23	21	20	18
β=2	8783	9002	10262	11800	13784	16096	18789	21924	25573	29821
(sec)	33	31	30	29	27	24	21	22	19	15
β=3	9836	10082	11494	13216	15439	18028	21044	24555	28642	33399
(sec)	32	29	29	28	25	21	19	20	17	13
β=4	11017	11292	12873	14802	17291	20191	23569	27501	32079	37407
(sec)	31	28	27	25	23	20	17	16	14	12

TABLE A.2: [15] × [10] ACO

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	6958	7062	8192	9175	10339	11815	13502	15430	17631	20146
(sec)	96	94	91	88	85	82	77	72	67	61
β=2	7932	8051	9339	10460	11786	13470	15393	17590	20099	22966
(sec)	95	93	89	87	83	76	74	68	63	59
β=3	8884	9017	10460	11715	13200	15086	17240	19701	22511	25722
(sec)	93	91	88	86	81	73	71	64	61	55
β=4	9950	10099	11715	13121	14784	16896	19309	22065	25213	28809
(sec)	91	89	86	84	78	72	69	61	59	45

TABLE A.3: [15] $\times$ [15] ACO

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	11794	12065	13573	15338	17332	19585	22131	25008	28259	31933
(sec)	238	235	230	228	224	220	215	209	201	199
β=2	13681	13996	15745	17792	20105	22719	25672	29009	32781	37042
(sec)	233	231	228	221	219	218	210	207	197	194
β=3	15323	15675	17635	19927	22518	25445	28753	32491	36714	41487
(sec)	231	229	225	219	215	211	205	201	195	189
β=4	17161	17556	19751	22318	25220	28498	32203	36389	41120	46466
(sec)	229	226	220	214	210	208	203	199	191	183

TABLE A.4: [20] $\times$ [10] ACO

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	12402	12638	14393	16860	19749	23133	27097	31739	37177	43545
(sec)	351	345	338	330	325	318	313	305	300	295
β=2	14262	14533	16552	19389	22712	26603	31162	36500	42753	50077
(sec)	343	339	336	326	321	312	309	301	295	286
β=3	15974	16277	18539	21716	25437	29796	34901	40880	47884	56086
(sec)	340	335	330	321	318	309	303	299	292	281
β=4	17891	18231	20763	24322	28489	33371	39089	45786	53630	62817
(sec)	339	330	325	319	315	305	301	295	291	279

TABLE A.5: [20] $\times$ [15] ACO

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	6664	6844	7994	9605	11539	13863	16655	20009	24038	28878
(sec)	181	174	169	165	159	151	147	141	138	130
β=2	7997	8213	9584	11416	13715	16478	19796	23783	28572	34326
(sec)	172	170	168	160	155	150	144	138	135	126
β=3	8956	9198	10734	12786	15361	18455	22172	26637	32001	38445
(sec)	168	166	165	157	152	148	142	135	131	121
β=4	10031	10302	12022	14320	17204	20670	24833	29833	35841	43058
(sec)	165	161	159	152	149	145	140	133	128	119

TABLE A.6: [20] × [20] ACO

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	24824	25296	29242	33920	40042	47819	57100	68176	81392	97162
(sec)	461	457	450	445	440	435	428	425	418	410
β=2	30037	30608	35383	41044	47611	56774	67798	80954	96653	115387
(sec)	455	452	448	441	436	432	421	417	415	405
β=3	33641	34281	39628	45969	53324	63587	75933	90668	108252	129233
(sec)	451	449	443	439	431	429	419	413	409	401
β=4	37678	38394	44384	51485	59723	71217	85045	101548	121242	144741
(sec)	450	445	439	433	429	423	415	410	405	395

TABLE A.7: [30] × [10] ACO

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	68243	69881	82180	96972	114427	135024	160060	192527	231570	278517
(sec)	2538	2530	2523	2520	2514	2510	2495	2490	2485	2475
β=2	81209	83158	97794	115397	136168	160679	189601	223729	264000	312481
(sec)	2533	2529	2519	2515	2511	2503	2493	2484	2479	2467
β=3	90954	93137	109529	129245	152509	179960	212353	250577	295680	349979
(sec)	2529	2521	2511	2505	2501	2498	2491	2481	2475	2461
β=4	101869	104314	122673	144754	170810	201555	237835	280646	331162	391976
(sec)	2525	2520	2509	2501	2497	2491	2487	2478	2469	2451

TABLE A.8: [50] × [10] ACO

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	263505	267194	317159	377737	449885	535812	638153	760040	905207	1078102
(sec)	13205	13194	13185	13173	13157	13151	13147	13142	13137	13133
β=2	318841	323305	383763	457062	544360	648333	772165	919648	1095301	1304503
(sec)	13200	13189	13181	13170	13151	13142	13139	13137	13134	13125
β=3	357102	362101	429814	511909	609684	726133	864824	1030006	1226737	1461044
(sec)	13195	13181	13174	13165	13146	13140	13131	13128	13121	13117
β=4	399954	405554	481392	573338	682846	813269	968603	1153607	1373946	1636369
(sec)	13191	13175	13170	13161	13139	13133	13129	13125	13111	13101

Table A.9: [100] $\times$ [5] ACO

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	267499	275256	333060	419656	528767	666246	839470	1057732	1332742	1679255
(sec)	20193	20188	20181	20175	20167	20161	20151	20146	20137	20125
β=2	326349	335813	406334	511980	645095	812820	1024153	1290433	1625946	2048691
(sec)	20189	20184	20175	20169	20163	20152	20146	20138	20131	20128
β=3	365511	376110	455094	573418	722507	910358	1147052	1445285	1821059	2294534
(sec)	20185	20176	20164	20155	20151	20147	20141	20131	20125	20115
β=4	409372	421244	509705	642228	809207	1019601	1284698	1618719	2039586	2569879
(sec)	20179	20171	20161	20149	20146	20141	20133	20128	20119	20100

Table A.10: [100] $\times$ [10] ACO

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	274356	280117	333340	423342	537644	682808	867166	1101300	1398651	1776287
(sec)	16010	15990	15981	15973	15964	15955	15932	15926	15914	15902
β=2	331971	338942	403341	512243	650549	826197	1049270	1332573	1692368	2149308
(sec)	16001	15985	15975	15969	15959	15943	15923	15917	15904	15893
β=3	371807	379615	451742	573712	728615	925341	1175183	1492482	1895452	2407225
(sec)	15991	15981	15971	15965	15953	15932	15921	15910	15899	15887
β=4	416424	425169	505951	642558	816049	1036382	1316205	1671580	2122907	2696092
(sec)	15985	15979	15964	15962	15942	15923	15919	15901	15890	15876

Table A.11: [100] $\times$ [20] ACO

Parameter	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	1056487	1080786	1296943	1673057	2158244	2784134	3591533	4633078	5976670	7709905
(sec)	122576	122563	122548	122528	122502	122471	122454	122434	122409	122374
β=2	1267784	1296943	1556332	2007668	2589892	3340961	4309840	5559693	7172004	9251886
(sec)	122566	122549	122530	122514	122478	122439	122430	122405	122373	122324
β=3	1419919	1452577	1743092	2248589	2900679	3741876	4827021	6226857	8032645	10362112
(sec)	122551	122532	122522	122503	122453	122410	122399	122368	122334	122267
β=4	1590309	1626886	1952263	2518419	3248761	4190902	5406263	6974079	8996562	11605565
(sec)	122533	122513	122496	122484	122430	122382	122376	122335	122278	122201

TABLE A.12: [200] $\times$ [10] ACO

Parameter	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	6418210	6597920	8115441	10793537	14355404	19092688	25393275	33773056	44918164	59741158
(sec)	672757	672719	672671	672611	672530	672418	672285	672140	671971	671767
β=2	7766034	7983483	9819684	13060180	17370039	23102152	30725863	40865397	54350978	72286801
(sec)	672731	672688	672638	672578	672492	672374	672230	672068	671879	671651
β=3	8697958	8941501	10998046	14627402	19454444	25874411	34412966	45769245	60873096	80961217
(sec)	672704	672657	672605	672547	672461	672336	672186	672021	671827	671592
β=4	9741713	10014481	12317812	16382690	21788977	28979340	38542522	51261554	68177867	90676563
(sec)	672676	672624	672569	672499	672399	672258	672088	671899	671680	671407

TABLE A.13: [500] $\times$ [20] ACO

Appendix B

Results ACOVNS for VJS

The results of ACOVNS for parameter α and β

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	3205	3269	3661	4101	4593	5144	5761	6453	7227	8094
(sec)	9735	9871	10464	11091	11757	12462	13210	14003	14843	15733
β=2	3590	3661	4101	4593	5144	5761	6453	7227	8094	9065
(sec)	11501	11674	12376	13142	13949	14809	15838	16804	17841	18930
β=3	4020	4101	4593	5144	5761	6453	7227	8094	9065	10153
(sec)	12911	13125	13950	14816	15747	16727	17780	18888	20056	21299
β=4	4503	4593	5144	5761	6453	7227	8094	9065	10153	11372
(sec)	14494	14740	15669	16650	17701	18818	20009	21269	22613	24034

Table B.1: [10] $\times$ [10] ACOVNS

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	7530	7718	8953	10386	12047	13975	16211	18805	21813	25304
(sec)	9975	10145	10804	11506	10758	11458	12202	12996	13840	14740
β=2	8509	8722	10117	11736	13614	15792	18318	21249	24649	28593
(sec)	11787	11999	12788	13638	14541	15512	16397	17479	18641	19869
β=3	9530	9768	11331	13144	15247	17687	20517	23799	27607	32024
(sec)	13230	13491	14407	15370	16415	17519	18700	19958	21294	22711
β=4	10674	10940	12691	14721	17077	19809	22979	26655	30920	35867
(sec)	14854	15153	16186	17280	18460	19719	21070	22490	24013	25623

Table B.2: [15] $\times$ [10] ACOVNS

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	6870	6973	7949	9062	10331	11777	13426	15306	17448	19891
(sec)	9980	10170	10932	11752	12634	13581	14600	15695	16872	18137
β=2	7832	7949	9062	10331	11777	13426	15306	17448	19891	22676
(sec)	10991	11211	12064	12993	13986	15062	16062	17286	18606	20015
β=3	8772	8903	10150	11571	13190	15037	17142	19542	22278	25397
(sec)	12331	12602	13588	14643	15788	17010	18322	19745	21228	22852
β=4	9824	9972	11368	12959	14773	16842	19199	21887	24952	28445
(sec)	13849	14159	15272	16466	17759	19142	20642	22243	23975	25801

TABLE B.3: [15] × [15] ACOVNS

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	11341	11602	11718	11835	11953	12073	12194	12316	12439	12563
(sec)	10677	10901	11664	12481	13354	14289	15290	16360	17505	18730
β=2	13156	13458	13593	13729	13866	14005	14145	14286	14429	14573
(sec)	12086	12400	13278	14226	15237	16332	17560	18815	20153	21574
β=3	14734	15073	15224	15376	15530	15685	15842	16000	16160	16322
(sec)	13555	13881	14895	15975	17127	18364	19695	21112	22630	24251
β=4	16502	16882	17051	17221	17393	17567	17743	17920	18100	18281
(sec)	15223	15591	16737	17960	19269	20683	22199	23815	25560	27417

TABLE B.4: [20] × [10] ACOVNS

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	12021	12249	14332	16768	19619	22954	26856	31422	36763	43013
(sec)	10831	11102	11979	12925	13946	15048	16237	17519	18904	20397
β=2	13824	14087	16482	19283	22562	26397	30885	36135	42278	49465
(sec)	12369	13421	14482	15653	16890	18249	19779	21360	23066	24906
β=3	15483	15777	18459	21597	25269	29565	34591	40471	47351	55401
(sec)	13885	14264	15434	16689	18042	19508	21093	22788	24643	26628
β=4	17341	17670	20674	24189	28301	33113	38742	45328	53033	62049
(sec)	15588	16027	17351	18776	20319	21987	23795	25744	27822	28867

TABLE B.5: [20] × [15] ACOVNS

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	6486	6661	7993	9592	11510	13813	16575	19890	23868	28642
(sec)	11456	11742	12564	13444	14385	15392	16469	17622	18856	20176
β=2	7718	7927	9512	11414	13697	16437	19724	23669	28403	34084
(sec)	12726	13019	13936	14939	16005	17190	18542	19859	21266	22773
β=3	8645	8878	10654	12784	15341	18409	22091	26509	31811	38174
(sec)	14292	14686	15755	16888	18108	19408	20809	22314	23933	25650
β=4	9682	9943	11932	14318	17182	20618	24742	29691	35629	42754
(sec)	16042	16494	17708	19007	20397	21894	23499	25208	27029	28990

TABLE B.6: [20] × [20] ACOVNS

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	23268	23710	28215	33576	39955	47547	56581	67331	80124	95347
(sec)	16543	16146	17454	18868	20396	22048	23834	25764	27851	30107
β=2	27689	28215	33576	39955	47547	56581	67331	80124	95347	113463
(sec)	19201	19758	21365	23122	21266	22563	24581	26592	28757	31111
β=3	31012	31601	37605	44750	53252	63370	75411	89739	106789	127079
(sec)	21538	22084	23917	25887	28022	30324	32832	35542	38479	41640
β=4	34733	35393	42118	50120	59643	70975	84460	100507	119604	142329
(sec)	24161	24793	26863	29091	31513	34130	36972	40038	43357	47946

TABLE B.7: [30] × [10] ACOVNS

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	62781	64288	77145	92574	111089	133307	159968	191962	230355	276426
(sec)	68022	69859	75657	81936	88737	96102	104079	112717	122073	132205
β=2	70943	72645	87174	104609	125531	150637	180764	216917	260301	312361
(sec)	77563	79425	86029	93190	100944	104275	112767	122154	132306	143316
β=3	79456	81363	97635	117162	140595	168713	202456	242947	291537	349844
(sec)	86912	89286	96736	104792	113529	122990	133251	144366	156407	160955
β=4	88990	91126	109351	131222	157466	188959	226751	272101	326521	391825
(sec)	97385	100069	108439	117490	127305	137940	149468	161950	165440	179436

TABLE B.8: [50] × [10] ACOVNS

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	233944	237219	282291	335926	399752	475705	566089	673646	801639	953950
(sec)	351108	357779	382824	409621	380948	407614	436147	466677	499345	534299
β=2	266696	270430	321812	382956	455717	542304	645341	767956	913868	1087503
(sec)	386238	392804	420321	449767	481279	519300	558790	597924	639796	684610
β=3	298700	302881	360429	428910	510403	607380	722782	860111	1023532	1218003
(sec)	432637	440884	471784	504835	540214	578068	618587	661946	708342	757967
β=4	334544	339227	403680	480380	571652	680266	809516	963324	1146356	1364164
(sec)	484601	493865	528503	565541	605198	647623	693028	741613	793585	849216

TABLE B.9: [100] $\times$ [5] ACOVNS

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	248025	253234	306413	370759	448619	542829	656823	794755	961654	1163601
(sec)	1053421	1083970	1180444	1285503	1399913	1524505	1660186	1807943	1968849	2144077
β=2	287709	293751	355439	430081	520398	629681	761914	921916	1115519	1349777
(sec)	1211446	1247790	1358866	1237954	1348160	1481627	1598697	1741005	1895975	2064744
β=3	322234	329001	398091	481690	582845	705243	853344	1032546	1249381	1511751
(sec)	1356881	1396256	1520560	1655922	1803341	1963882	2138714	2329112	2536465	2762259
β=4	360902	368481	445862	539493	652787	789872	955745	1156452	1399306	1693161
(sec)	1519748	1563879	1703133	1854768	2019904	2199741	2395593	2608880	2841149	3094098

TABLE B.10: [100] $\times$ [10] ACOVNS

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	242930	249975	304969	372063	453917	553778	675609	824243	1005577	1226804
(sec)	598007	615947	671382	731807	797669	869460	947711	1033005	1125976	1227313
β=2	269652	277472	338516	412990	503847	614694	749926	914910	1116190	1361752
(sec)	705658	724711	789964	860288	937738	1003379	1100732	1199825	1307835	1425566
β=3	302011	310769	379138	462548	564309	688457	839918	1024699	1250133	1525163
(sec)	790369	814113	887417	967317	1054425	1149368	1252856	1365665	1488630	1622647
β=4	338252	348061	424635	518054	632026	771072	940708	1147663	1400149	1708182
(sec)	885262	911878	994013	1083536	1181122	1287476	1403421	1529800	1667549	1817695

TABLE B.11: [100] $\times$ [20] ACOVNS

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	931429	952852	1181536	1465105	1816730	2252745	2793404	3463821	4295139	5325972
(sec) (sec)	5035688	5191794	4828369	5166355	5527999	5914959	6329006	6772037	7246079	7753305
β=2	1127029	1152951	1429659	1772777	2198244	2725822	3380019	4191224	5197118	6444426
(sec)	5639980	5848659	6258090	6696171	7164924	7809767	8426775	9016674	9647869	10323237
β=3	1262273	1291305	1601218	1985510	2462033	3052921	3785622	4694171	5820772	7217757
(sec)	6316818	6512678	6968615	7456459	7978461	8537004	9134634	9774113	10458367	11190502
β=4	1413745	1446261	1793364	2223772	2757477	3419271	4239896	5257471	6519264	8083888
(sec)	7074877	7294258	7804916	8351313	8935968	9561554	10230943	10947183	11713545	12533584

TABLE B.12: [200] $\times$ [10] ACOVNS

Instance	α=1	α=2	α=3	α=4	α=5	α=6	α=7	α=8	α=9	α=10
β=1	5846398	6010097	7512621	9390777	11738471	14673089	18341361	22926701	28658376	35822971
(sec)	20851539	21581343	23631570	25876570	28334844	31026654	33974186	37201734	40735898	44605809
β=2	7132606	7332319	9165398	11456748	14320935	17901168	22376460	27970575	34963219	43704024
(sec)	25021859	25997711	28467515	31171941	34133286	37034615	40552934	44405487	48624027	53243340
β=3	7988518	8212197	10265246	12831557	16039447	20049308	25061636	31327044	39158806	48948507
(sec)	28024529	29005431	31760995	34778337	38082328	41700204	45661772	49999698	54749736	59951022
β=4	8947140	9197660	11497075	14371344	17964180	22455225	28069032	35086290	43857862	54822328
(sec)	31387521	32486146	35572380	38951823	42652311	46704341	51141310	55999804	61319863	67145328

TABLE B.13: [500] $\times$ [20] ACOVNS

Appendix C

Results Financial Derivative Problem

The results of Financial Derivative Problem for individual problem instances.

CFD	Match	Trades
1	Y	23
2	N	73
3	N	11
4	N	5
5	Y	1
6	Y	1

Table C.1: new-CFD-3

CFD	Match	Trades
1	Y	25
2	Y	16
3	Y	15
4	Y	8
5	Y	19
6	Y	14

Table C.2: new-CFD-5

CFD	Match	Trades
1	Y	19
2	Y	22
3	Y	16
4	N	9
5	Y	1
6	Y	17
7	Y	19
8	Y	18

TABLE C.3: new-CFD-1

CFD	Match	Trades
1	Y	10
2	Y	8
3	Y	8
4	Y	11
5	Y	11
6	Y	6
7	Y	2

TABLE C.4: new-CFD-7

CFD	Match	Trades
1	Y	9
2	Y	12
3	N	10
4	Y	9
5	Y	8
6	Y	6
7	N	3
8	Y	17
9	Y	2

TABLE C.5: new-CFD-4

CFD	Match	Trades
1	Y	10
2	Y	10
3	Y	7
4	Y	9
5	Y	2
6	Y	3
7	N	6
8	Y	1
9	Y	10
10	Y	22
11	Y	16

TABLE C.6: new-CFD-6

CFD	Match	Trades
1	Y	14
2	Y	852
3	Y	29
4	Y	50
5	Y	241

TABLE C.7: new-CFD-6

CFD	Match	Trades
1	Y	11239
2	N	6578
3	Y	712
4	Y	9987

TABLE C.8: CFD-9

Bibliography

[1] Aarts, H.L.E., and Korst, J.H.M., "Simulated annealing and boltzmann machines: A stochastic approach to combinatorial optimization and neural computing", *John Wiley Sons Inc*,(1988).

[2] Aarts, H.L.E., Korst, J.H.M., and Van Laarhoven, P.J.M., "A quantitative analysis of the simulated annealing algorithm: A case study for the travelling salesman problem", *Journal of Statistical Physics*, 50(1) (1988) 187-201.

[3] Akker, M.V.D., Hoogeven, H., and Woeginger, G.J., "The Two-Machine Open Shop Problem: To Fit or Not to Fit, That Is the Question", *Operations Research Letters*, 31 (2003) 219-224.

[4] Al-Anzi, F.S., and Allahverdi, A., "A hybrid tabu search heuristic for the two-stage assembly scheduling problem", *International Journal of Operations Research*, 3(2) (2006) 106-119.

[5] Alaykyran, K., Engin, O. and Doyen, A., "Using ant colony optimization to solve hybrid flow shop scheduling problems", *International Journal of Advanced Manufacturing Technology*, 35(5-6) (2007) 541-550.

[6] Alkandari, A.M., "3D packing of balls in different containers by VNS", *PhD thesis School of Information Computing and Mathematics Brunel University UK*, (2013).

[7] Amaldass, N.I.L., Lucas, C., and Mladenovic, N., "A heuristic hybrid framework for vector job scheduling", *Yugoslav Journal of Operations Research*, 27 (2017) 31-45.

[8] Andreatt, A., and Ribeiro, C., "Heuristics for the phylogeny problem", *Journal of Heuristics*, 8(4) (2002) 429-447.

[9] Artigues, C., and Roubellat, O., "An efficient algorithm for operation insertion in a multi-resource job-shop schedule with sequence-dependent setup times", *Production Planning and Control*, 13(2) (2002) 175-186.

[10] Baker,K.R., and Kanet,J.J., "Job shop scheduling with modified due dates", *Journal of Operations Management*, 4(1) (1983) 11-22.

[11] Bansal, N., Vredeveld, T., and Van-Der-Zwaan R ., "Approximating vectorscheduling: almost matching upper and lower bounds", *Springer, Heidelberg*, 8392(3) (2014) 47-59.

[12] Baruah, S., Bonifaci, V., D'Angelo, G., Marchetti-Spaccamela, A., Van-Der-Ster, S., and Stougie, L., "Mixed criticality scheduling of sporadic task systems", *Proceedings of the 19th annual European symposium on algorithms (ESA)*, 6942(4) (2011) 555-566.

[13] Barzokia, R.M., Hejazia, R.S., and Mazdeh, M.M., "A Branch and Bound Algorithm to Minimize the Total Weighed Number of Tardy Jobs and Delivery Costs", *Applied Mathematical Modelling*, 37(7) (2013) 4924-4937.

[14] Battiti, R., and Tecchiolli, G., "The continuous reactive tabu search: Blending combinatorial optimization and stochastic search for global optimization", *Annals of Operations Research*, 63 (1996) 153-188.

[15] Bar-Yam, Y., "Dynamics of Complex Systems", *Studies in Nonlinearity Addison-Wesley Reading MA*,(1997).

[16] Bauer, A., Bullnheimer, B., Hartl, R.F., and Strauss C., "An ant colony optimization approach for the single machine total tardiness problem", *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, (1999) 1445-1450.

[17] Bonifaci, V., and Wiese, A., "Scheduling unrelated machines of few different types", *CoRR* (2012).

[18] Brasel, H., Herms, A., Morig, M., Tautenhahn, T., Tusch, J., and Werner, F., "Heuristic Constructive Algorithms for Open Shop Scheduling to Minimize Mean Flow Time", *European Journal of Operational Research*, 189 (2008) 856-870.

[19] Brimberg, J., Hansen, P., and Mladenovic, N., "Attraction probabilities in variable neighborhood search", *4OR*, 8(2) (2010) 181-194.

[20] Blum, C., "Beam-ACO hybridizing ant colony optimization with beam search: an application to open shop scheduling", *Computers and Operations Research*, 32(6) (2005) 1565-1591.

[21] Blum, C., and Roli, A., "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison", *ACM Computing Surveys*, 35(3) (2003) 268-308.

[22] Campbell, H.R., and Smith, D. M., "A heuristic algorithm for n-jobs m- machines sequencing problem", *Management Science*, 16 (1970) 630-637.

[23] Carlier, J. and Pinson, E., "An algorithm for solving the job-shop problem", *Management Science*, 35(2) (1989) 164-176.

[24] Cerny, V., "Thermodynamical approach to travelling salesman problem: an efficient simulation", *Journal of Optimization Theory and Applications*, 45 (1985) 41-51.

[25] Chekuri, C., and Khanna, S., "On multidimensional packing problems", *SIAM J Computer*, 33(4) (2004) 837-85.

[26] Chelouah, Y.R., and Siarry, P., "Tabu search applied to global optimization", *European Journal of Operations Research*, 123 (2000) 256-270.

[27] Chen, J., " An Integer Programming Model for Open Scheduling Problem", *The Far East Journal*, 20(1) (1991).

[28] Chen, Z.L., and Hall, N.G., "Supply chain scheduling conflict and cooperation in assembly systems", *Operation Research*, 55(6) (2007) 1072-1089.

[29] Choi, I.C., and Choi, D.S., "A local search algorithm for job-shop scheduling problems with alternative operations and sequence-dependent setups", *Computers and Industrial Engineering*, 42(1) (2002) 43-58.

[30] Chong, K.E., Omar, M.K., and Bakar, N.A., "Solving assembly line balancing problem using genetic algorithm with heuristics treated initial population", *Proceedings of The World Congress on Engineering*, 2 (2008) 1273-1277.

[31] Colorni, A., Dorigo, M., Maniezzo, V., and Trubian, M., "Ant system for job shop scheduling", *Belgian Journal of Operations Research Statistics and Computer Science*, 34(1) (1994) 39-53.

[32] Comerton-Forde, C., and Putnins, T.J., "Measuring of closing price manipulation" ,*Journal of Financial Intermediation*, 20(2) (2009) 135-58.

[33] Cordon, O., Fernndez, I., Viana, D., and Herrera, F., "Analysis of the best-worst ant system and its variants on the tsp", *Mathematic Soft Computer*, (2002).

[34] Crainic, T.G., Gendreau, M., Hansen, P., and Mladenovic, N., "Cooperative parallel variable neighborhood search for the p-median", *Journal of Heuristics*, 10 (2004) 293-314.

[35] David, D., Theodoulidisand, B., and Eliza, A., "Cross border challenges in financial markets monitoring and surveillance. A case study of customer driven service value networks", *Annual SRII Global Conference*, (2012) 146-157.

[36] Deneubourg, J.L., Aron, S., Goss, S., and Pasteels, J. M., "The self-organizing exploratory pattern of the Argentine ant", *Journal of Insect Behaviour*, 3(1990) 159-168.

[37] Dorigo, M., "Optimization learning and natural algorithms (in Italian)", *PhD thesis Dipartimento di Elettronica Politecnico di Milano Italy*, (1992)

[38] Dorigo, M., and Blum, C., "The hyper-cube framework for ant colony optimization", *IEEE Transactions on Systems Man and Cybernetics Part B (Cybernetics)*, 34(2) (2004) 1161-1172.

[39] Dorigo, M., and Caro, D.G., "The ant colony optimization meta-heuristic", *New Ideas in Optimization McGraw-Hill London UK*, (1999) 11-22.

[40] Dorigo, M., and Gambardella, L.M., "Ant colonies for the travelling salesman problem", *Biosystems*, 43 (1997) 73-81.

[41] Dorigo, M., and Gambardella, L.M., "Ant colony system: a cooperative learning approach to the travelling salesman problem", *IEEE Transactions in Evolutionary Computation*, 1(1) (1997) 53-66.

[42] Dorigo, M., Maniezzo, V., and Colorni, A., "Ant system: optimization by a colony of cooperating agents", *EEE Transactions on Systems Man and Cybernetics Part B: Cybernetics*, 26(1) (1996) 29-41.

[43] Dorigo, M., and Stutzle, T., "Ant colony algorithms for quadratic assignment problem", *New Ideas in Optimization McGraw-Hill London UK*, (1999).

[44] Dorigo, M., and Stutzle, T., "Ant Colony Optimization", *MIT Press-London*, (2004).

[45] Drezner, Z., Marcoulides, G.A., and Stohs, M.H., "Financial application of a Tabu search variable selection model", *Journal Application Mathematics Decision Science*, 5(4) (2001) 215-234.

[46] Duratem, A., Pantrigom, J.J., Pardo, G.E., and Mladenovic, N., "Multi-objective variable neighborhood search: an application to combinatorial optimization problems", *Journal of Global Optimization*, 63 (2015) 515-536.

[47] Durr, C., and Hurand, M., "Finding total unimodularity in optimization problems solved by linear programs", *Algorithmica* 59(2) (2011) 256-268.

[48] Elffers, J., and De Weerdt, M., "Scheduling with two non-unit task lengths is NP-complete", *arXiv preprint*, (2014) 1412-3095

[49] Eswaramurthy, V.P., and Tamilarasi, A., "Hybridizing tabu search with ant colony optimization for solving job shop scheduling problems", *International Journal of Advance Manufacturing Technology*, 40 (2008) 1004-1015.

[50] Fandel, G., and Stammen-Hegene, C., "Simultaneous lot sizing and scheduling for multi-product multi-level production", *International Journal of Production Economics*, 104(2) (2006) 308-316.

[51] Fisher, H., and Thompson, G.L., "Probabilistic learning combinations of local job-shop schedul-ing rules J.F. Muth G.L. Thompson (eds.)", *Industrial Scheduling Prentice Hall Englewood New Jersey*,(1963).

[52] Gajpal, Y., Rajendran, C., and Ziegler, H., "An ant colony algorithm for scheduling in flowshops with sequence- dependent setup times of jobs", *International Journal of Advanced Manufacturing Technology*, 30(5-6) (2006) 416-424.

[53] Gao, L., Zeng, Y., and Dong, A.G., "An ant colony algorithm for solving max-cut problem", *Progress in Natural Science*, 18(9) 2008) 1173-1178.

[54] Garey, M.R., Johnson, D.S., Simons, B.B., and Tarjan, R.E., "Scheduling unittime tasks with arbitrary release times and deadlines", *SIAM Journal on Computing*, 10(2) (1981) 256-269.

[55] Glover, F., "Future paths for integer programming and links to artificial intelligence", *Computers Operations Research*, 13(5) (1986) 533-549.

[56] Goffem, W.L., Ferrier, G., and Roger, J., "Simulated Annealing: An initial application in econometrics", *Computer Science in Economics and Management*, 5(2)(1992) 133-146.

[57] Gonzalez, T., and Sahni, S., "Open Shop Scheduling to Minimize Finish Time", *Journal of the Association for Computing Machinery*, 23 (1976) 665-679.

[58] Greening, D.R., "Simulated annealing with errors ", *PhD thesis, University of California* , (1995).

[59] Gupta, J.N., "A functional heuristic algorithm for the flow-shop scheduling problem", *Operational Research*, 22(1) (1971) 39-47.

[60] Hansen, P., Brimberg, J., Urosevic, D., and Mladenovic, N., "Primal-dual variable neighborhood search for the simple plant-location problem", *Informs Journal on Computing*, 19(4) (2007) 552-564.

[61] Hansen, P., Mladenovic, N., Brimberg, J., and Moreno Prez, J.A., "Variable neighborhood search Handbook of Metaheuristics 2nd edition (Gendreau and Potvin Eds)", *International Series in Operations Research Management Sciences*, 146 (2010)61-86.

[62] Hansen, P., Mladenovic, N., and Perez-Brito, D., "Variable neighborhood decomposition search", *Journal of Heuristics*, 7 (2001) 335-350.

[63] Heinonen, J., and Pettersson, F., "Job-shop scheduling and visibility studies with a hybrid ACO algorithm", *Applied Mathematics and Computation*, 187(3) (2007) 989-998.

[64] Holthaus, O., and Rajendran, C., "New dispatching rules for scheduling in a job shop-an experimental study", *International Journal of Advanced Manufacturing Technology*, 13(2) (1997) 148-153.

[65] Huang, K., and Liao, C., "Ant colony optimization combined with tabu search for the job shop scheduling problem", *Computers and Operations Research*, 35(4) (2008) 1038-1046.

[66] Hsu, W.N., "Approximation algorithms for the assembly line crew scheduling problem", *Mathematics of Operation Research*, 9(3) (1984) 376-383.

[67] Irawan, C., Salhi, S., and Drezner, Z., "Hybrid meta-heuristics with VNS and Exact Methods: Application to large unconditional and conditional vertex p-centre problems", *Journal of Heuristics*, 22(2016) 507-537

[68] Ishibuchi, H., Yamamoto, N., Murata, T., and Tanaka H., "Genetic algorithms and neighborhood search algorithms for fuzzy flowshop scheduling problems", *Fuzzy Sets and Systems*, 67(1) (1994) 81-100.

[69] Jaszkiewicz, A., "Genetic local search for multi-objective combinatorial optimization", *European Journal of Operational Research*, 137(1)(2002) 50-71.

[70] Johnson, S.D., Papadimtriou, H.C., and Yannakakis, M., "How easy is local search?", *Journal of Computer and System Sciences*, 37(1) (1988) 79-100.

[71] Johnson, S. M., "Optimal two-and three-stage production schedules with setup times included", *Naval Research Logistics Quarterly*, 1(1) (1954) 61-68.

[72] Kandavanam, G., Botvich, D., Balasubramaniam, S., and Jennings, B., "An efficient general variable neighborhood search for large travelling salesman problem with time windows", *International Conference on Artificial Evolution (Evolution Artificielle)*,5975 (2009) 49-60.

[73] Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P., "Optimization by simulated annealing", *Science*, 220 (1983) 3671-3680.

[74] Kolonko, M., "Some new results on simulated annealing applied to the job shop scheduling problem", *European Journal of Operational Research*, 113(1) (1999) 123-136.

[75] Kubale, M., and Nadolski, A., "Chromatic Scheduling in a Cyclic Open Shop", *European Journal of Operational Research*, 164 (2005) 585-591.

[76] Kuo, I., Horng, S.J., Kao, T. W., Lin, T.L., Lee, C.L., Terano, T., and Pan, Y., "An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model", *Expert Systems with Applications*, 36(3) (2009) 7027-7032.

[77] Lawrence, S., "Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement)", *Graduate School of Industrial Administration Carnegie-Mellon University Pittsburgh Pennsylvania*, (1984).

[78] Lazic, J., Hana, S., Mladenovic, N., and Urosevic, D., "Variable neighborhood decomposition search for 0-1 mixed integer programs", *Computers Operations Research*, 37(6) (2010) 1055-1067.

[79] Lenstra, J.K., and Kan, A.H.G., "Complexity of vehicle routing and scheduling problems", *Networks*, 11(2) (1981) 221-227.

[80] Lian, Z., Gu, X., and Jiao, B., "A similar particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan", *Applied Mathematics and Computation*, 175(1) (2006) 773-785.

[81] Liang, Y.C., and Wu, C.C., "A variable neighborhood descent algorithm for the redundancy allocation problem", *Industrial Engineering and Management Systems*, 4(1)(2005) 94-101.

[82] Liao, C., and Cheng, C., "Variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date", *Computers and Industrial Engineering*, 52(4) (2007) 404-413.

[83] Liouane, N., Saad, I., Hammadi, S., and Borne, P., "Ant systems and local search optimization for flexible job shop scheduling production", *International Journal of Computers Communications and Control*, 2(2) (2007) 174-184.

[84] Low, C., Hsu, C. M., and Huang, K. I., "Benefits of lot splitting in job-shop scheduling", *International Journal of Advanced Manufacturing Technology*, 24(9-10) (2004) 773-780.

[85] Lu, L., Yuan, J., and Zhang, L., "Single machine scheduling with release dates and job delivery to minimize the makespan?", *Theoretical Computer Science*, 393(1) (2008) 102-108.

[86] Manikas, A., and Chang, Y. L., "Multi-criteria sequence-dependent job shop scheduling using genetic algorithms", *Computers and Industrial Engineering*, 56(1) (2009) 179-185.

[87] Maric, M., Stanimirovic, Z., and Mladenovic, N., "Metaheuristic methods for solving the bi-level incapacitated facility location problem with clients preferences", *Faculty of Mathematics University of Belgrade Serbia Electric Notes in Discrete Mathematics*, 39 (2012) 43-50.

[88] Mavrovouniotis, M., "Ant colony optimization in stationary and dynamic environments", *PhD thesis Department of Computer Science University of Leicester UK*,(2013).

[89] Mazdeha, M.M., Shashaania, S., Ashouria, A., and Khalil-Hindib, S. K., "Single-machine batch scheduling minimizing weighted flow times and delivery costs", *Applied Mathematical Modelling*, 35(1) (2011) 563-570.

[90] Mladenovic, N., and Hansen, P., "Variable neighborhood Search", *Computers and Operations Research*, 24(11) (1997) 1097-1100.

[91] Mladenovic, N., Todosijevic, R., and Urosevic, D., "Two level general variable neighborhood search for attractive traveling salesman problem", *Yugoslav Journal of Operations Research*, 23 (2012) 19-30.

[92] Mladenovic, N., Todosijevic, R., and Uroevic, D., "An efficient general variable neighborhood search for large travelling salesman problem with time windows", *Yugoslav Journal of Operations Research*, 22 (2012).

[93] Mladenovic, N., Urosevic, D., and Perez-Brito, D., "Variable neighborhood search for minimum linear arrangement problem", *Yugoslav Journal of Operations Research*, 26(1) (2016) 3-16.

[94] Morton, T., "Heuristic scheduling systems: with applications to production systems and project management", *John Wiley Sons*, 3 (1993).

[95] Naderi, B., Fatemi, G.S.M.T., Aminnayeri, M., and Zandieh, M., "Scheduling Open Shops with Parallel Machines to Minimize Total Completion Time", *Journal of Computational and Applied Mathematics*, 235 (2011) 1275-1287.

[96] Nawaz, M., Enscore, J., and Ham, I., "A Heuristic algorithm for the M machine, n-job flow shop sequencing problem", *OMEGA*, 11(1) (1983) 91-95.

[97] Nowicki, E., and Smutnicki, C., "A fast taboo search algorithm for the job shop problem", *Management Science*, 42(6) (1996) 797-813.

[98] Ogut, H., Doganay, M., and Aktas, R., "Detecting stock price manipulation in an emerging market the case of turkey", *Expert Systems with application*, 36(9) (2009) 11944-11959.

[99] Osman, I.H., "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem", *Annals of Operations Research*, 41 (1993) 421-451.

[100] Osman, I.H., and Laporte, G., "Metaheuristics: a bibliography", *Annals of Operations Research*, 63(5) (1996) 511-623.

[101] Palmer, D.S., "Sequencing jobs through a multi-stage process in the minimum total time: a quick method of obtaining a near optimum", *Operations Research*, 16(1) (1965) 101-107.

[102] Palomo-Martnez, P.J., Salazar-Aguilar, M.A., Laporte, G., and Langevin, A., "A hybrid variable neighborhood search for the orienteering problem with mandatory visits and exclusionary constraints", *Computers Operations Research*, 78 (2017) 408-419.

[103] Pan, Q. K., Tasgetiren, F.M., and Liang, Y.C., "A discrete particle swarm optimization algorithm for the no-wait flow-shop scheduling problem", *Computers and Operations Research*, 35(9) (2008) 2807-2839.

[104] Patkar, S., Poojari, C., and Porwal, P., "An investigation into approximate solutions for deterministic and stochastic multi-dimensional sequencing", *Brunel University Archive Brunel*,(2005).

[105] Pereira, J., and Vila, M., "Variable neighborhood search heurisitcs for a test assembly design problem", *Expert Systems with application*, 42(10) (2015) 4805-4817.

[106] Pinedo, M., "Scheduling: Theory, Algorithms and Systems ", *2nd ed, Prentice Hall, Englewood Cliffs, NJ.*, (2002).

[107] Pinedo, M., "Planning and scheduling in manufacturing and services", *New York, Springer*,(2005).

[108] Pirrong, C., "Detecting manipulation in futures markets: the ferruzzisoybean episode", *American Law and Economics Review*, 6(1) (2004) 28-71.

[109] Potts, C.N., "An algorithm for the single machine sequencing problem with precedence constraints", *Mathematical Programming Studies*, 13 (1980) 78-87.

[110] Potts, C.N., Sevastjanov, S.V., Strusevich, V.A., Van Wassenhove, L.N., and Zwaneveld, C.M., "Two stage assembly scheduling problem:complexity and approximation", *Operation Research*, 43(2) (1995) 346-355.

[111] Ponnambalam, S.G.N., Jawahar, N., and Girish, B.S., "An ant colony optimization algorithm for flexible job shop scheduling problem", *New Advanced Technologies*, 4 (2010) 73-94.

[112] Puchinger, J., and Raidi, R.G., "Bringing order into the neighborhood: relaxation guided variable neighborhood search", *Journal of Heurisitics*, 14(5) (2008) 457-472.

[113] Punniyamoorthy, M., and Thoppan, J.J., "ANN-GA based Model for Stock market Surveillance", *Journal of Financial Crime*, 20(1) (2013) 52-66.

[114] Qu, R., Xu, Y., and Kendall, G., "A variable neighborhood descent search algorithm for delay-constrained least-cost multicast routing", *International Conference on Learning and Intelligent Optimization*, 5851 (2009) 15-29.

[115] Rajab, R.S., "Some application of continuous variable neighborhood search meta heuristic (mathematical modelling)", *PhD thesis School of Information Computing and Mathematics Brunel University UK.*

[116] Rajendran, C., and Chaudhuri, D., "An efficient heuristic approach to the scheduling of jobs in a flow-shop", *European Journal of Operational Research*, 61(3) (1992) 318-325.

[117] Rajendran, C., and Ziegler, H., "Ant colony algorithms for permutation flowshop scheduling to minimize makespan total flowtime of jobs", *European Journal of Operational Research*, 155(2) (2004) 426-438.

[118] Salmasi, N., Logendran, R., and Skandari, M.R., "Makespan minimization of a flow-shop sequence-dependent group scheduling problem", *International Journal of Advanced Manufacturing*, (2011).

[119] Salhi, S., "Heuristic Search: The Emerging Science of Problem Solving", *Springer*, (2017).

[120] Santos, D.L., Hunsucker, J.L., and Deal, D.E., "On makespan improvement in flow shops with multiple processors", *Production Planning and Control*, 12(3) (2001) 283-295.

[121] Seda, M., "Mathematical model for flow job and job shop problems", *World Academy of Science Engineering and Technology*, 18(3)(2007) 331-342.

[122] Selvarajah,E., Steiner, G., and Zhang, R., "Single machine batch scheduling with release times and delivery costs", *Journal of Scheduling*, 16(1) (2013) 69-79

[123] Sevkli, M., and Aydin, M., "Variable neighborhood search algorithm for job shop scheduling problems", *Evolutionary Computation in Combinatorial Optimization*, 3906 (2006) 261-271.

[124] Sevkli, M., and Aydin, E., "Variable neighborhood search for job shop scheduling problems", *Journal of Software*, 1(2) (2006).

[125] Sgall, J., "Open Problems in Throughput Scheduling", *Springer Berlin Heidelberg*, (2012).

[126] Sha, D.Y., Lin, H.H., and Hsu, C.Y., "A Modified Particle Swarm Optimization for Multi-Objective Open Shop Scheduling", *Proceedings of the International Multi Conference of Engineers and Computer Scientists*, 3 (2010) 17-19.

[127] Shyu, S.J., Lin, B.M.T., and Yin, P.Y., "Application of ant colony optimization for no-wait flow-shop scheduling problem to minimize the total completion time", *Computers and Industrial Engineering*, 47(2) (2004) 181-193.

[128] Simons, B., "A fast algorithm for single processor scheduling. In Foundations of Computer Science", *19th Annual Symposium on 1978*, (1978)

[129] Singh, R,M., "Thesis: A Study on Flexible Flow Shop and Job Shop Scheduling using Meta-heuristic Approaches", *National Institute of Technology, India* (2014).

[130] Steinhofel, K., Albrecht, A., and Wong, C. K., "Two simulated annealing-based heuristics for the job shop scheduling problem", *European Journal of Operational Research*, 118(3) (1999) 524-548.

[131] Solimanpur, M., Vrat, P., and Shankar, R., "A neuro-tabu search heuristic for the flow shop scheduling problem", *Computers and Operations Research*, 31(13) (2004) 2151-2164.

[132] Storer, R.H., Wu, S.D., and Vaccari, R., "New search spaces for sequencing instances with application to job shop scheduling", *Management Science*, 38 (1992)1495-1509.

[133] Stutzle, T., and Hoos, H., "Max-Min Ant System", *Future Generation Computer Systems*, 16(8) (2000) 889-914.

[134] Subramaniam, V., Ramesh, T., Lee, G.K., Wong, Y.S., and Hong, G.S., "Job shop scheduling with dynamic fuzzy selection of dispatching rules", *International Journal of Advanced Manufacturing Technology*, 16(10) (2000) 759-764.

[135] Surekha, P., and Sumathi, S., "Solving fuzzy based job shop scheduling problems using GA and ACO", *Journal of Emerging Trends in Computing and Information Sciences*, 1(2) (2010) 95-102.

[136] Taillard, E., "Robust taboo search for the quadratic assignment problem", *Parallel Computing*, 17(1991) 443-455.

[137] Taillard, E., "Benchmarks for basic scheduling problems", *European Journal of Operational Research*, 64 (1993) 278-285.

[138] Taillard, E., "Some efficient heuristic methods for the flow shop sequencing problem", *European Journal of Operational Research*, 47(1) (1990) 65-74.

[139] Tamiz, M., and Jones, F.D., "A General Purpose Interactive Goal Programming Algorithm", *Multiple Criteria Decision Making*, (1997) 433-444.

[140] Teh, Y.S., and Rangaiah, G.P., "Tabu search for global optimization of continuous functions with application to phase equilibrium calculations", *Computers and Chemical Engineering*, 27 (2003) 1665-1679.

[141] Todosijevic, R., Hanafi, S., Lazic, J., and Mladenovic, N., "Variable and single neighborhood diving for MIP feasibility", *Yugoslav Journal of Operations Research*, 26(2) (2016) 131-157.

[142] Toumi, S., Cheikh, M., and Jarboui, B., "0-1 Quadratic knapsack problem solved with VNS algorithm", *Electronic Notes in Discrete Mathematics*, 47 (2015) 269-276.

[143] Tseng, L.Y., and Lin, Y.T., "A hybrid genetic local search algorithm for the permutation flow-shop scheduling problem", *European Journal of Operational Research*, 198(1) (2009) 84-92.

[144] Van Laarhoven, J.M.P., Aarts, H.L.E., and Lenstra, K.J., "Job shop scheduling by simulated annealing", *Journal Operations Research*, 40(1) (1992) 113-125.

[145] Vredeveld, T., "Vector Scheduling Problems", *Springer Science+ Business Media New York*, (2004).

[146] Wang, J.B., Daniel Ng, C.T., Cheng, T.E., and Liu, L.L., "Minimizing total completion time in a two-machine flow shop with deteriorating jobs", *Applied Mathematics and Computation*, 180(1) (2006) 185-193.

[147] Webster, S., and Azizgolu, M., "Dynamic programming algorithm for scheduling parallel ma-chines with family setup time", *Computer and Operation Research*, 28(2) (2001) 127-137.

[148] Yamada, T., and Nakano, R., "A genetic algorithm applicable to large-scale job-shop instances R. Manner B. manderick (eds)", *Parallel instance solving from nature 2 North-Holland Amsterdam*, (1992) 281-290.

[149] Yamada, T. and Nakano, R., "Genetic algorithms for job-shop scheduling problems", *In Proceedings of the Modern Heuristics for Decision Support*, (1997) 67-81.

[150] Yang, S.X., Deb, S., and Fong, S., Metaheuristic algorithms: optimal balance of intensification and diversification", *Applied Mathematics Information Sciences*, 8(3) (2014) 977-983.

[151] Yang, H.A., Xu, Y.P., Sun, S.D., and Yu, J.J., "A Job Shop Scheduling Heuristic Algorithm Based on Probabilistic Model of the Search Space", *Materials Science Forum*, 532 (2006) 1084-1087.

[152] Zecchin, C. A., Holger, R., Ross, A., and John, B., "Ant colony optimization applied to water distribution system design: Comparative study of five algorithms", *Journal of Water Resources Planning and Management*, 133(1) (2007) 87-92.

[153] Zhang, G., Gao, L., Li, X., and Li, P., "Variable neighborhood genetic algorithm for the flexible job shop scheduling problem", *Intelligent Robotics and Applications*, (2008) 503-512.

[154] Zhang, J., Zhong, J.H., and Huang, Q., "Implementation of an ant colony optimization technique for job shop scheduling problem", *Transactions of the Institute of Measurement and Control*, 28(1) (2006) 93-108.

[155] Zhou, H., Cheung, W., and Leung, L.C., "Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm", *European Journal of Operational Research*, 194(3) (2009) 637-649.