A HYBRID APPROACH FOR SOLVING MIXED BINARY INTEGER PROGRAMMING PROBLEMS

A thesis submitted for the degree of Doctor of Philosophy

by

Mastura Binti Mat Shariff

Department of Mathematics

Brunel University

September 2018

Abstract

Optimisation appears in many aspects of day to day life and more often, involves integer optimisation of very large scales. Although technology advancements have enabled many combinatorial optimisation problems to be solved exactly, this is only true for small and some of the medium instances. For large instances, they require high computational times and worse, fail to be solved due to the massive usage of the machine's memory.

In this research, we aim to develop a hybrid technique, focusing on solving the MBIP problem rather than finding the best solution for individual problem's application. Therefore, we proposed a general framework of a hybrid technique that may need minor adjustment when applied to various optimisation problems, in particular to the mixed binary integer programming (MBIP) problems.

The hybrid approach proposed in this research is the collaborative combination of the linear programming (LP) relaxation with variable neighbourhood search (VNS). We use LP relaxation solutions to generate initial solutions and use VNS to improve the solutions obtained. To illustrate the flexibility of the proposed method, we implement the proposed method on two similar MBIP problems; the constrained index tracking problem (CITP) and the gas supply chain problem.

The proposed hybrid technique generates satisfactory solutions within significantly shorter amount of computational time. For the CITP problem, we compare the obtained solutions with the solutions provided by the CPLEX solver (with time and solution limit imposed) and a genetic algorithm (GA) approach. For most of the instances, our proposed hybrid technique gives better solutions with significant reduction of the computational time compared to the time taken by the CPLEX solver and the GA approach.

For the gas supply chain problem, the proposed hybrid technique manage to replicate the solutions generated by the CPLEX solver (with time and solution limit imposed) within a shorter computational time. When we decrease the number of locations that were allowed to supply gas

to a specific location, the proposed hybrid technique generated better solutions with lower total costs than the solutions given by the CPLEX solver.

The proposed hybrid technique was successfully implemented for both problems by adjusting the optimal LP solutions of the decision variables that are used to guide the search process. Satisfactory solutions were obtained for both problems within a relatively shorter computational time.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude towards Allah (my God) that has given me strength to complete this research.

I would like to thank my supervisor, Dr Cormac Lucas for his continuous support and supervision towards the completion of this research. Advices given have been a great help in improving this research.

I would also like to express my great appreciation to my friend and colleague, Miss Lilysuriazna, who has always provided me with comments and suggestions.

To the most important people, my family, especially my parents who have always support me, I am deeply grateful.

I also want to extend my special thanks to the staffs and technical support personnel, Mr Nalin Soni of the Mathematics Department for the assistance given.

Lastly, I would like to thank the Majlis Amanah Rakyat MARA for providing me the financial assistance to complete my Phd.

AUTHOR'S DECLARATION

I confirm that this work is my own except where references are cited. Once again, I would like to thank Dr Cormac Lucas for his support and guidance during this research.

Table of Contents

List of Figures

List of Figures (cont)

List of Tables

List of Algorithms

Glossary

B&B	Branch and bound
BVNS	Basic variable neighbourhood search
CITP	Constrained index tracking problem
CO	Combinatorial optimisation
GA	Genetic algorithms
GAMIP	Genetic algorithm and a mixed integer program
GAP	Generalised assignment problem
GLS	Guided local search
GRASP	Greedy randomised adaptive search procedure
GVNS	General variable neighbourhood search
ILP	Integer linear programming
ILS	Iterative local search
IP	Integer programming
IPM	Interior point method
LP	Linear programming
LS	Local search
MA	Memetic algorithms
MBIP	Mixed binary integer programming
MILP	Mixed integer linear programming
MINLP	Mixed integer non-linear programming
MIP	Mixed integer programming
MKP01	0-1 multidimensional knapsack problem
MP	Mathematical programming
POMDP	Partially observed Markov decision process
RVNS	Reduced variable neighbourhood search
SA	Simulated annealing
SWO	Squeaky wheel optimisation
TE	Tracking error
TS	Tabu search
TSP	Traveling salesman problem
VND	Variable neighbourhood descent
VNS	Variable neighbourhood search

Chapter 1.0: Introduction

1.1 An optimisation problem

An optimisation problem is a set of independent variables or parameters that often include conditions or restrictions termed as the constraints of the problem which define acceptable values of the variables (Gill et al, 1982). Many challenging applications in science and industry can be formulated as optimisation problems and optimisation occurs in the minimisation of time, cost, and risk or the maximisation of profit, quality, and efficiency (Talbi, 2009).

The aim of an optimisation model is to find the value of the decision variables that will maximise or minimise an objective function among the set of all values for the decision variables that satisfy the given constraints and the following are the components of an optimisation model (Winston, 2004):

1. Objective function(s): Function which is to be maximised or minimised and in many situations, an organisation may have more than one objective function (multiple objective decision-making problems). Also known as the cost, utility or fitness function.
2. Decision variables: Variables whose values are under the modeller's control and influence the performance of the system.
3. Constraints: Restrictions that are imposed on the values of the decision variables.

In the context of this research[1], an optimisation problem can be defined as (Rajab, 2012)

$$\boldsymbol{Q = \min\{f(x) | x \in X \subseteq S\}} \qquad \boldsymbol{(1.0)}$$

[1] From this point onwards, all discussions assume a minimisation problem unless mentioned otherwise.

where $\boldsymbol{S}$ represents the solution space, $\boldsymbol{f}: \boldsymbol{S} \rightarrow \mathbb{R}$ is the objective function to be minimised, and $\boldsymbol{x} \in \boldsymbol{X}$ is a feasible solution. The objective function $\boldsymbol{f}$ assigns a real number value to each solution $\boldsymbol{x} \in \boldsymbol{X}$ in the search space and a solution $\boldsymbol{x}^* \in \boldsymbol{X}$ is said to be the global minimum if

$$\boldsymbol{f}(\boldsymbol{x}^*) \leq \boldsymbol{f}(\boldsymbol{x}), \forall \boldsymbol{x} \in \boldsymbol{X} \qquad (\mathbf{1.1})$$

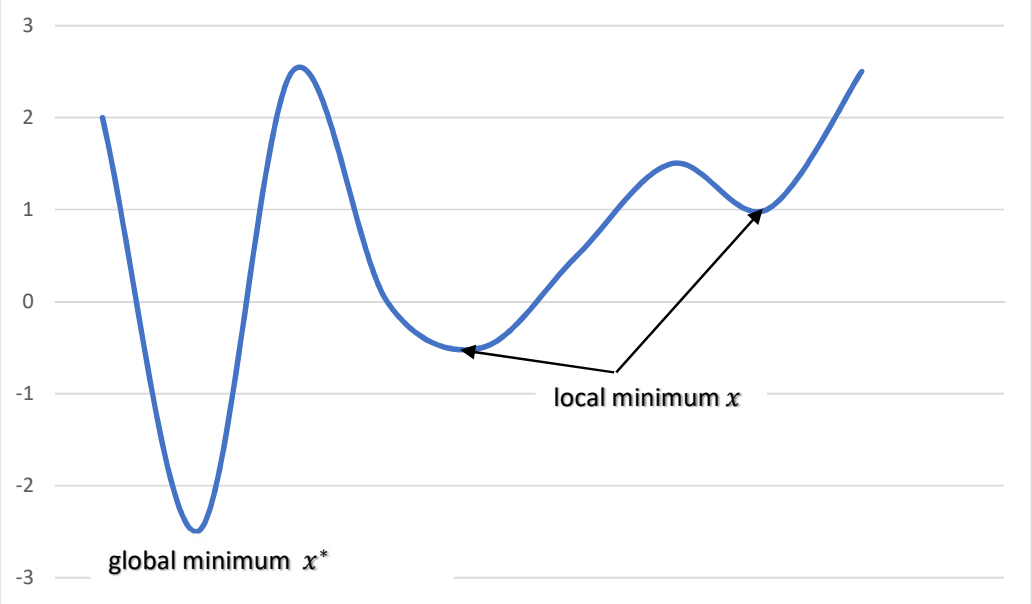

Figure 1.0: Global minimum vs local minimum

1.2 Combinatorial optimisation

Combinatorial optimisation (CO) aims to use combinatorial techniques to solve discrete optimisation problems, and from the computer science perspective, CO seeks to reduce the size of the possible solutions' size or making the search process faster by improving the algorithm with the use of mathematical methods (Brilliant.org, 2019). CO problems are concerned with a study of the best selection, arrangement, sequence, etc., subjected to some appropriately chosen objective function (Bennell, 2015). CO problems include cardinality constrained portfolio

tracking problem, supply chain planning problem, travelling salesman problem (TSP), timetable arrangements and many more.

The solutions to CO problems may be in the form of arrangements, choices of objects, sequences, assignments, route in a network, job schedules, etc. In other words, CO problems can be described as searching for the best configurations from a countable set of candidate solutions. The following are some definitions taken from the literature, describing CO problems:

1. *CO is a coined term to describe the mathematical programming areas that are concerned with the solution of optimisation problems that have a pronounced combinatorial or discrete structure (Christofides et al, 1979).*
2. *CO is a mathematical study to find an optimal arrangement, grouping, ordering, or selection of discrete objects that are usually finite in numbers (Osman and Laporte, 1996).*
3. *CO is a class of problems characterised by discrete decision variables and a finite search space (Talbi, 2009).*

Christofides et al (1979) pointed out that the areas covered are becoming increasingly important due to the large number of practical problems that can be formulated and solved as CO problems. However, CO problems are computationally challenging in nature due to their large sizes. Solving large CO problems using exact algorithms such as the simplex method can be really expensive, resulting in high computational time that is not practical for solving real life problems.

1.3 Linear programming

Linear programming (LP) is a powerful tool in modelling many real world applications due to its (MIT, 2013):

1. Applicability to model real life problems.

2. Solvability with the existence of theoretically and practically efficient techniques for solving large-scale problems.

A simple way to describe LP is as an optimisation problem where the objective function and all the constraints (equalities or inequalities) are linear (Gil et al, 1982).

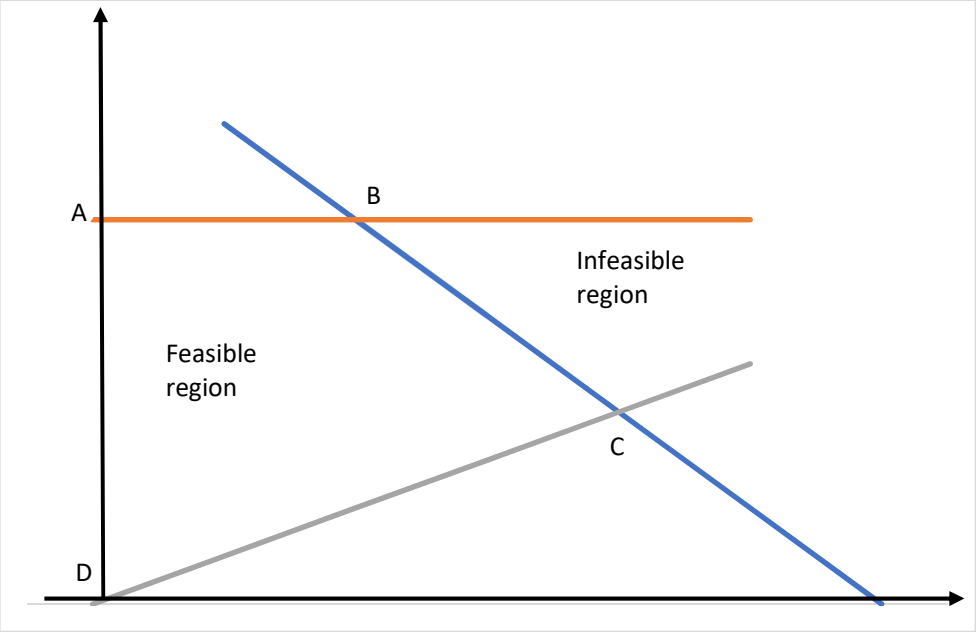

Figure 1.1: The feasible and infeasible region of a linear programming

The feasible region **(ABCD)** in **Figure 1.1** represents the possible solutions that satisfy the problem constraints. LP is a commonly used model in mathematical programming (Talbi, 2009) and its history can be traced back to 1827 when Fourier published a method to solve a system of linear inequalities (Sierksma and Zwols, 2015).

The standard LP can be formulated as:

$$\textbf{Min } \boldsymbol{f(x) = c^T x} \qquad \textbf{(1.2)}$$

subject to

$$\boldsymbol{Ax} \geq \boldsymbol{b} \tag{1.3}$$

$$\boldsymbol{x} \geq \boldsymbol{0} \tag{1.4}$$

where $\boldsymbol{x}$ is a vector of continuous decision variables, $\boldsymbol{A}$ is a matrix and $\boldsymbol{c^T}$ and $\boldsymbol{b}$ are constant vectors of coefficients. The objective function in **(1.2)** and constraints in **(1.3)** are linear functions. Continuous LP problems can be efficiently solved using exact algorithms such as a simplex-type method or interior point methods because the feasible region of the problem and the objective function are convex (Talbi, 2009).

However, many real-life applications must be modelled with discrete variables and noticeably in many practical optimisation problems, the resources such as machinery and people are indivisible (Talbi, 2009).

1.4 Integer programming

Integer programming (IP) is a mathematical optimisation problem that deals with solving linear models in which some of the decision variables are restricted to take only integer values. There are four types of IP problems based on the decision variables characteristics:

(a) Pure integer programming: all of the decision variables have to be integer.

(b) Binary integer programming: all the decision variables' values are binary (0 or 1).

(c) Mixed integer programming (MIP): some of the decision variables have to be integer.

(d) Mixed binary integer programming (MBIP): some of the decision variables are binary variables.

The importance of IP is that it allows the modelling of the indivisible decision variables that is not possible under LP. Another important feature of IP is that it permits the modelling of (McCarl and Spreen, 1997):

1. Fixed cost: usually arising in the production process.
2. Logical conditions such as:
   (i) Conditional use
   (ii) Complementary products
   (iii) Complementary investment
   (iv) Sequencing
3. Discrete levels of resources where variables are constrained by discrete resource conditions.
4. Distinct variable values where situations may require that certain decision variables are only permitted to take certain distinct values.
5. Nonlinear representations.
6. Approximation of nonlinear functions.

IP problems are difficult to solve. Compared to LP whose solutions are proven to be at the constraint intersections (Dantzig, 1963), an IP problem has an unknown number of possible solutions and no general statement can be made about the location of the solutions (McCarl and Spreen, 1997).

1.5 Mixed integer programming

Mixed integer programming (MIP) is an optimisation problem where the decision variables are both discrete and continuous, generalising LP and IP models (Talbi, 2009). There are two categories of MIP; the mixed integer linear programming (MILP) and mixed integer non-linear programming (MINLP). The difference between the two lies in the properties of the objective function and the constraints of the MILP and MINLP where they are linear in the former but non-linear in the latter. Since MINLP is very difficult to solve, MIP usually refers to MILP.

MIP can be formulated as follows:

$$\textbf{Min } \boldsymbol{f}(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{c^T x} + \boldsymbol{d^T y} \tag{1.5}$$

subject to

$$\boldsymbol{Ax} + \boldsymbol{Gy} \leq \boldsymbol{b} \tag{1.6}$$

$$\boldsymbol{x} \geq \mathbf{0} \in \mathbb{R} \tag{1.7}$$

$$\boldsymbol{y} \geq \mathbf{0} \in \mathbb{Z} \tag{1.8}$$

where $\boldsymbol{x}$ is a vector of continuous decision variables, $\boldsymbol{y}$ is a vector of discrete decision variables, $\boldsymbol{c^T}$, $\boldsymbol{d^T}$ and $\boldsymbol{b}$ are constant vectors of coefficients, and $\boldsymbol{A}$ and $\boldsymbol{G}$ are matrices.

1.5.1 Mixed binary integer programming

Mixed Binary Integer Programming (MBIP) is the special case of the MIP where the discrete decision variables of the problem can only take either the value of 0 or 1. These binary variables are usually used to answer the yes/no question to the problem. The difference of MBIP can be seen if **(1.8)** is rewritten as follows:

$$\boldsymbol{y} \in [\mathbf{0}, \mathbf{1}] \tag{1.9}$$

The element of the integer variables in an optimisation problem makes the problem become more difficult to solve. This is due to the many combinations of the integer decision variables needed to be tested and the number of these combinations can rise exponentially with the problem's size (Frontline Solvers, 2018).

1.5.2 MIP-solvers for MIP

Many commercial MIP solvers give the flexibility to the users to have certain control to provide heuristic control over some of the parameters that affect exploration of the branching rules, the frequency of application of the internal heuristics, the fact of emphasizing the solution integrality rather than its optimality, etc (Fischetti and Lodi, 2003). Nevertheless, there are some cases where these flexibilities are inadequate, leaving the users to opt for a specialised ad-hoc heuristic, losing the advantage of working on the generic framework of MIP.

One function of the general-purpose MIP solvers is to serve as a black-box for the local search metaheuristics to effectively explore the solution space.

1.5.2.1 Large neighbourhood search using MIP-solvers

Large neighbourhood search using the MIP-solvers offers the advantage of easy application for CO problems that can be expressed in the form of a MIP problem. With the availability of the effective general purpose MIP-solvers such as CPLEX, IBM ILOG, GUROBI, etc, these MIP-solvers are based on a search tree framework but further include the solution of LP relaxations of a given MIP model for the given problem to obtain its lower and upper bounds (Blum et al., 2011).

Although such MIP solvers work well for small and medium sized problems, often it will be too computationally challenging for them to solve large scale CO problems. This has always been an issue and raised interest among modellers and many research investigations have been dedicated to developing approaches to overcome this matter. Therefore, MIP can be very useful in searching large neighbourhoods within a metaheuristic framework. One approach is to partially fix values to some variables with the remaining unfixed variables being later determined through optimisation of the MIP.

However, the percentage of variables to be fixed plays a crucial role in determining good neighbourhood search size because the number of variables left unfixed reflects the size of the

search space. Having too large a neighbourhood might result in high computational time whilst too small a neighbourhood may hinder the chances of getting good or near optimal solutions.

1.6 Heuristics

Many real life problems are typically complex and require settings or decision making to be made in a short amount of time. Thus, making it not possible or practical to rely on the exact methods that usually need high computational time, especially for large size problems, and often fail to solve them. Unlike the classical exact methods, heuristic techniques offer relatively good solutions within a reasonable amount of time but at the expense of the solution quality.

Originating from the Greek verb *heuriskein*, the word heuristic means to find or search. Therefore, heuristic techniques can be viewed as the search techniques that depend on the past experience or knowledge to deliver reasonably good solutions. Despite the inability of providing optimality to the produced solutions, heuristic techniques continue to be a popular method among researchers due to the following reasons (Aickelin and Clark, 2011):

1. The unlikeliness of finding optimal solutions in a reasonable time for large size complex problems.
2. An optimal solution based on estimated data will almost certainly not be optimal for the actual data due to the problem be ill-defined or because of imprecise data. For such cases, obtaining a robust solution that will be near optimal over most scenarios is preferable.
3. A mathematical model should be used as a guidance in a decision making process, and to make the final choice, the user may use several different solutions and in some cases, human justifications rather than technical measures to balance several criteria.

1.7 Metaheuristic

Emerging from the success of the heuristic techniques, metaheuristics have become increasingly popular in recent years due to the many successful applications to a wide range of CO problems. The word *meta* carries the meaning of "beyond, in an upper level". Setting apart from the heuristic techniques, a metaheuristic is a higher-level guided search technique, aiming to explore the search space for solutions that are near-optimal.

Metaheuristics can be described as general frameworks to build heuristics for combinatorial and global optimisation problems (Hansen et al, 2008). Metaheuristics are a branch of optimisation in computer science and applied mathematics that are related to algorithm and computational complexity theory and have developed into various communities that sit at the intersection of several fields, including artificial intelligence, computational intelligence, soft computing[2], mathematical programming, and operations research (Talbi, 2009).

Talbi (2009) pointed out that for more than the past two decades, metaheuristics have become more popular in different research areas and industries, proven by the existence of a large number of sessions, workshops, and conferences that deal with the design and application of metaheuristics. In practice, metaheuristics have been gaining a lot of interest in diverse technologies, industries, and services due to their ability to solve a wide range of complex real-life optimisation problems. These typically are in logistics, bioinformatics and computational biology, engineering design, networking, environment, transportation, data mining, finance, business to name but a few.

Metaheuristics offer a more simple and easy to implement technique that makes this approach a popular choice among modellers. Hansen and Mladenović (2003) listed the following as the characteristics of a desired metaheuristic technique:

[2] The use of approximate calculations to provide satisfactory solutions to the computationally hard complex problem and sometimes, referred to computational intelligence (Rouse, 2018).

1. Simplicity: the metaheuristic should be based on a simple and clear principle, which should be largely applicable.
2. Precision: steps of a metaheuristic should be formulated in precise mathematical terms, independent from the possible physical or biological analogy which was an initial source of inspiration.
3. Coherence: all steps of heuristics for particular problems should follow naturally from the metaheuristic's principle.
4. Efficiency: heuristics for particular problems should provide optimal or near-optimal solutions for all or at least most realistic instances. Preferably, they should find optimal solutions for most benchmark problems for which such solutions are known, when available.
5. Effectiveness: heuristics for particular problems should take moderate computing time to provide optimal or near-optimal solutions.
6. Robustness: the performance of heuristics should be consistent over a variety of instances, i.e. not just fine-tuned to some training set and less good elsewhere.
7. User-friendliness: heuristics should be clearly expressed, easy to understand and, most important, easy to use. This implies they should have very few parameters and and ideally none.
8. Innovation: preferably, the metaheuristic's principle and/or the efficiency and effectiveness of the heuristics derived from it should lead to new types of applications.

This list was later updated with the following additions (Hansen et al., 2010):

9. Generality: the metaheuristic should lead to good results for a wide variety of problems.
10. Interactivity: the metaheuristic should allow the user to incorporate his knowledge to improve the resolution process.
11. Multiplicity: the metaheuristic should be able to present several near optimal solutions from which the user can choose one.

Although metaheuristic techniques offer simple application, significant knowledge is crucial to make sure effective implementation of the techniques to the studied problem. One aspect that needs careful consideration is the selection of the search space and the neighbourhood structure.

1.7.1 Why metaheuristics for optimisation problems

Talbi (2009) anticipated that metaheuristics will continuously gain more interest in the future due to the increasing of size as well as complexity in the optimisation problems. The author justifies the use of metaheuristics with the main characteristics of optimisation problems as follows:

1. An easy problem with very large instances but the known exact polynomial-time algorithms are too expensive due to the size of the instances.
2. An easy problem with hard real-time constraints where metaheuristics are widely used to reduce the search time in the real-time optimisation problems although efficient exact algorithms are available to solve the problem.
3. A difficult problem with moderate size and/or difficult structures of the input instances.
4. Optimisation problems with nonanalytic models that cannot be solved in an exhaustive manner and many of the practical problems are defined by a black box scenario of the objective function.
5. Nondeterministic models of optimisation (problems with uncertainty and robust optimisation) may intensify those conditions and for some noisy problems, uncertainty and robustness cannot be modelled analytically. Exact algorithms are not the preferred approach due to the ambiguity of the model and given the fuzziness of the data, optimal solutions are not necessarily found.

Gendreau and Potvin (2005) explained that, in solving the complex combinatorial optimisation problems, the challenge from developing specialised heuristics has shifted to adapt a metaheuristic to a specific problem or problem class, which usually requires much

less work compared to developing a specialised heuristic from scratch and a good metaheuristic implementation is likely to provide near-optimal solutions in a reasonable computational time.

1.8 Research motivations and thesis structure

This research is driven by the following reasons;

1. The successes of many hybrid techniques that exploit the strengths of the exact and approximate methods.
2. IP or MIP formulations are rarely solved by the metaheuristic techniques despite the fact that many combinatorial problems are solved exactly using IP. According to Gendreau and Potvin (2005), the exploitation of the IP or MIP formulations mostly involves tabu search. Motivated by this fact, this research attempts to exploit the MIP formulation of the studied problems using other metaheuristic search techniques in looking for a more powerful framework.
3. Variable neighbourhood search (VNS) offers great potential as a metaheuristic search process.

The contributions of this research will be twofold;

1. Algorithm: to provide an alternative technique in approaching how to solve the CO problems studied.
2. Computational results: generate satisfactory empirical results in reasonable computational time for the CO problems studied.

The remaining chapters of this thesis will be organised as follows. **Chapter 2** will be focused on discussing the IP methods and local search methods (and its extension) used in solving CO problems. Some of the commonly used methods will be briefly discussed, focusing more on the LP relaxation and VNS.

Chapter 3 introduce the readers to the general idea of the hybrid metaheuristics. Readers will be able to gain an overview of this technique and how it has been implemented in the literature. This chapter also provides the successful implementations of this technique in the previous researches and studies.

To exhibit the flexibility of the proposed solution approach, two CO problems that are different yet similar are considered in this research; the constrained index tracking problem (CITP) and the gas supply problem. In **Chapter 4** of this thesis, the properties of the two problems are described, to see the differences and similarities, and how the proposed method is adjusted to solve the CO problems.

The proposed solution method is presented and elaborated on in **Chapter 5**. Readers will be able to see how the proposed hybrid technique is implemented on both of the studied MBIP problems discussed in the previous chapter. This chapter gives details on the general framework of the proposed technique and how it is modified to solve the two similar yet different MBIP problems.

All the experimental results are discussed in **Chapter 6** and **Chapter 7** concludes the findings of this research. In **Chapter 8** of this thesis, future direction of this research are discussed.

Chapter 2.0: Review of literature

Many real-life problems can be formulated as a CO problem and with the numerous successes of applications, the CO field has increasingly become more important.

2.1 Overview

There have been numerous approaches dedicated to solving CO problems efficiently and they can be classified into two broad classes; exact and approximate methods. Whilst exact methods guarantee the optimality of the obtained solutions, they might be too expensive from a practical point of view due to the large computational time enumerating every possible combination of solutions. Exact methods obtain optimal solutions and can be successfully applied to small instances of difficult problems (Talbi, 2009). Approximate methods on the other hand provide potential suboptimal solutions but only an approximate-guarantee of the solution's quality (Festa, 2014; Galli, 2014).

Dumitrescu and Stützle (2003) stated that CO problems are often easy to state but then might be difficult to be solved. In their paper, "*A Survey of methods that combine local search and exact algorithm"*, the authors identified that IP methods are the exact approach that has significant success in solving the CO problems and for the approximate methods, local search (and extension thereof) is the most successful method.

The authors further discussed the important advantages that contribute to the success as well as the drawbacks that might limit the effectiveness of both methods. IP methods have the following important advantages:

1. Given a successful algorithm, optimal solutions are guaranteed in the IP methods.

2. Even if the algorithms are stopped before completion, valuable information on the upper/lower bounds of the optimal solution can be obtained (IP methods can become approximate if a stopping criterion is defined before solving them).
3. For the parts of the search space where optimal solutions cannot be found, IP methods allow provable pruning which many researchers find desirable as this will reduce the computational time.

However, for large scale CO problems, IP methods might require extensive computational time which is not practical and due to the large memory consumption, the early abortion of a program is inevitable. For many cases of successful solutions to CO problems, the algorithms are problem specific that requires immense efforts by the IP experts and hence if the problem's formulation changes, the algorithms are often hard to retune for solving the new problems.

Local search methods yield high-quality solutions by iteratively applying small modifications to a solution, hoping to search for a better solution. This approach has proven to be successful in achieving near-optimal (and sometimes optimal) solutions to numerous difficult CO problems when being embedded into higher-level guidance mechanisms such as metaheuristics (Aarts and Lenstra, 1997; Toth and Vigo, 2003; Hoos and Stützle, 2004). The advantages of local search is that it is the best performing algorithm used in practice for many CO problems and can examine enormous amounts of possible solutions in a short computational time. Also, local search methods offer more flexibility as they are easily adapted to slight variants of problems and the algorithms are typically easier to understand and implement compared to the exact methods. Nevertheless, local search methods do have several disadvantages. Local search methods cannot guarantee optimality and the reduction of the search space is not provable. They do not have well defined stopping criteria and often have problems with highly constrained CO problems where feasible areas of the solution space are disconnected. In practice, there are no efficient general-purpose local search solvers available and although often less than the exact algorithms, considerable programming efforts are often required for most applications of the local search algorithms.

Blum et al (2011), explained that in general, MIP solvers use a tree search framework that includes the solution of LP relaxations of a given MIP model for the studied problem to obtain lower and upper bounds. Various kinds of additional inequalities are used to tighten the obtained bounds resulting in a branch and cut algorithm and this kind of MIP approaches are very effective for small to medium sizes of CO problems but often fail to solve large instances of CO problems in practice. Using MIP-solvers such as CPLEX, MIP is likely to be very useful in searching large neighbourhoods within a metaheuristic framework.

The authors further stated that one way of defining the large neighbourhoods that are to be solved by the MIP-solver is to fix an appropriate portion of the decision variables to the values that they have in an incumbent solution and the remaining, referred to as free variables, to be determined by the MIP-solver. If there is an improvement in the solution obtained, it becomes the new incumbent solution and a new larger neighbourhood is defined around it and this process is repeated. The number of the free variables implies the size of the neighbourhood where too restricted neighbourhood might result in no improvement in the solution and too large a neighbourhood may require enormous run time. Therefore, selecting the number of fixed and free variables is crucial and this selection process can be random or found through a more sophisticated, guided way that considers the variables' potential impact on the objective function.

2.2 IP methods

Under this section, the discussion is focused on some of the popular IP methods including relaxation based methods, branch and bound, branch and cut, and cutting planes.

2.2.1 Relaxation methods

The most commonly used relaxation techniques are LP relaxation and Lagrangian relaxation. LP relaxation offers a much simpler application whilst the Lagrangian relaxation generally yields tighter bounds.

2.2.1.1 LP relaxation

LP relaxation is one of the commonly used IP methods. LP relaxation ignores the integrality conditions of an IP so that the optimisation problem can be solved using the LP solvers. If the solution found for the relaxed problem satisfies the integer conditions, which in general is not true; the solution is considered as the optimal solution to the IP problem. If the solution found is infeasible, then the IP problem is also infeasible.

The main purpose of this technique is to obtain bounds and good approximate solutions for the original problem, by solving the related, simpler relaxed problem.

The LP relaxation of ILP is obtained by relaxing the integrality constraint, yielding the LP (Blum et al., 2008)

$$\boldsymbol{z_{LP} = \min\{c^T x | Ax \geq b, x \geq 0, x \in \mathbb{R}^n\}} \qquad \boldsymbol{(2.1)}$$

For large size LP problems in **(2.1)**, the available exact algorithms such as the simplex method or interior-point algorithms can efficiently solve the problem to optimality. Observe that $\boldsymbol{z_{ILP} \geq z_{LP}}$

because the search space of ILP is contained within the LP search space. Therefore, the LP relaxation always provides lower bound to the original minimisation problem.

In the context of MBIP, LP relaxations can be described as transforming **(1.9)** into the following:

$$\boldsymbol{y_i} \in (\mathbf{0}, \mathbf{1}) \qquad (\mathbf{2.2})$$

Other than to satisfy the criterion of relatively easy computability, LP relaxation also gives a good approximation to the ILP solution for many cases (Christofides et al, 1979). LP relaxations provide a lower bound for the problem. Lin and Rardin (1977) demonstrated that the most important parameter in the success of branch-and-bound codes for ILP's is the 'distance' from the LP optimum to the MILP optimum.

2.2.1.1.1 LP relaxations for guiding metaheuristic search

LP relaxations provide bounds and a good approximation to the original IP problems which indicate a promising search space or possibly where an optimal solution might lie. There are several ways of exploiting the LP relaxations (Blum et al., 2008):

1. Creating promising initial solutions.
2. Guiding local improvement or the repairing of infeasible candidate solutions.
3. Exploiting dual variables.
4. Variable fixing: reduction to core problems.

Apart from securing the feasibility for the original IP problems, the repaired optimal solution (using a problem-specific procedure) of the LP relaxations is used as the starting point for the subsequent metaheuristics search. Usually, a simple rounding technique is chosen. Both Raidl and Feltl (2004) and Plateau et al. (2002) use simple rounding to respectively create an initial population of promising integer solutions and a population of different feasible candidate solutions.

2.2.1.2 Lagrangian relaxation

Another popular relaxation technique is Lagrangian relaxation that often yields a tighter bound for the related CO problem. The main idea behind this technique is to remove some complicating constraints and incorporate them in the objective function (Talbi, 2009).

Consider the ILP (Blum et al., 2008)

$$\boldsymbol{z_{ILP} = \min\{c^T x | Ax \geq b, Dx \geq d, x \geq 0, x \in \mathbb{Z}^n\}} \qquad \boldsymbol{(2.3)}$$

where constraints $\boldsymbol{Ax \geq b}$ are "nice" as the problem can be efficiently solved if the $\boldsymbol{m'}$ "complicating" constraints $\boldsymbol{Dx \geq d}$ are dropped. But, simply dropping the constraints may result in a weak bound and hence the constraints $\boldsymbol{Dx \geq d}$ are replaced by corresponding additional terms in the objective function

$$\boldsymbol{z_{LR} = \min\{c^T x + \lambda(d^T - Dx) | Ax \geq b, x \geq 0, x \in \mathbb{Z}^n\}} \qquad \boldsymbol{(2.4)}$$

The aim is now to find a specific vector $\boldsymbol{\lambda}$ that yields the best possible bound, leading to the Lagrangian dual problem

$$\boldsymbol{z^*_{LR} = \max_{\lambda \geq 0}\{z_{LR}(\lambda)\}} \qquad \boldsymbol{(2.5)}$$

This Lagrangian dual is a piecewise linear and convex function and can be solved by an iterative procedure such as a subgradient method.

2.2.2 Branch-and-bound methods

Branch-and-bound methods (B&B) were first introduced by Land and Doig in 1960 to solve discrete programming problems. The name B&B however, was first used by Little et al (1963) in their work, "*An algorithm for the travelling salesman problem*" and since has been a widely used terminology.

B&B methods are based on an implicit enumeration of all the solutions of the considered optimisation problem (Talbi, 2009). The search space is dynamically explored by building a tree whose root node represents the problem at hand and the leaf nodes are the potential solutions where the internal nodes are the sub problems of the total solution space. Subtrees containing no optimal solution are pruned from the search tree.

This approach is usually referred to as the 'divide and conquer' method. The idea behind B&B is to subdivide the feasible region of the problem and solve a relaxed problem over each divided region, making this method rely on efficient solution methods for LPs (Gustavsson, 2015). It uses bounds on the optimal solution; instead of exploring the entire feasible solution space, it only considers certain parts of the set of the feasible solutions (Galli, 2014).

The B&B mechanism can be described using the following fundamentals:

a) Branching.

   Let $\boldsymbol{X_0, Y_0} = \{\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{m}}, \boldsymbol{y} \in \mathbb{Z}^{\boldsymbol{n}}: \boldsymbol{Ax + Gy \leq b}, \boldsymbol{l' \leq x \leq u', l'' \leq y \leq u''}\}$ be the feasible solutions to the MIP problem in **(1.5)** where $\boldsymbol{l'}$, $\boldsymbol{u'}$, $\boldsymbol{l''}$ and $\boldsymbol{u''}$ are the lower and upper bounds of the variables $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively. $\boldsymbol{X_0, Y_0}$ is split into $\boldsymbol{k}$ finite disjoint subproblems of $(\boldsymbol{X_1, Y_1}), (\boldsymbol{X_2, Y_2}), \dots, (\boldsymbol{X_k, Y_k})$ and each subproblem is individually solved. $\boldsymbol{X_0, Y_0}$ is the root node of the search tree and the subproblems $(\boldsymbol{X_1, Y_1}), (\boldsymbol{X_2, Y_2}), \dots, (\boldsymbol{X_k, Y_k})$ are the leaf nodes. The subproblems are further recursively split into smaller subproblems (each subproblems is denoted as $\boldsymbol{I}$) and $\boldsymbol{f(x, y)}$ is minimised using the smaller search space.

b) Bounding (Pruning).

The objective function of each subproblem $\boldsymbol{I}$ are calculated. Any subproblem that cannot contain the optimal solution or are infeasible will be discarded from the enumeration tree.

The B&B method is widely used in many commercial software, such as CPLEX, to solve the MILP problems.

2.2.3 Cutting planes

Cutting plane algorithms for IP problems were introduced by Gomory in 1963 and were neglected for many years due to the slower convergence of these algorithms. In the 1980's, due to the development of polyhedral theory and the consequent introduction of strong, problem specific cutting planes, this method was revived and has since become a popular choice among researchers (Mitchell, 2002).

The basic idea of the cutting plane method is to cut off parts of the feasible region of the LP relaxation. The cuts are made so that the optimal integer solution $\boldsymbol{x}^*$ becomes an extreme point that can be found by the simplex method (Pan, 2015). This method iteratively modifies the feasible solutions or objective function by adding linear inequalities, called cuts, that are valid for the IP problem but violated by the optimal solution $\boldsymbol{x}^*$.

In general, a cutting plane algorithm can be summarised as follows (Pan, 2015):

Step 1: Solve the LP relaxation and obtain $\boldsymbol{x}^*$.

Step 2: If $\boldsymbol{x}^*$ is integral, then stop. Else find a valid inequality that will exclude $\boldsymbol{x}^*$.

Step 3: Go to **Step 1**.

Apart from the slower convergence to reach the optimum, this method is also subjected to the round-off errors that may cause serious problems in large size instances.

2.2.4 Branch-and-cut methods

Branch-and-cut methods provide optimal solutions to many IP problems and by far are the most successful technique (Mitchell, 2002). Branch and cut is a hybrid of the following two techniques to tighten the LP relaxation:

1. Cutting planes: Adding new constraints to repeatedly cut away the parts of the polytope to obtain an integer solution.
2. Branch and bound: avoid parts of the search tree that are not possible in producing the optimal solution.

To solve an IP problem, a branch and cut approach first solves the corresponding LP relaxation. Using the B&B method, the problem is recursively split into subproblems and solved using the simplex method. The cutting plane method is then used to eliminate any fractional solution to the LP by adding linear inequalities, without removing any integer solutions.

One importance of the branch and cut method is that it can be used to provide a lower bound. This method can be used in conjunction with heuristics or metaheuristics especially for large sized problems where optimality is not guaranteed/proven, a lower bound on the optimal value can be deduced from the algorithm to provide a guarantee (on the distance from optimality) on the goodness of the solution obtained (Mitchell, 2002).

2.3 Local search methods

Since its introduction in 1947, the local search (LS) method has inspired and has been extended to some of the very successful metaheuristics techniques; tabu search, simulated annealing, GRASP etc.

LS improve the solutions using the iterative movement, searching for better solutions in other defined neighbourhoods of the given problem. A good selection of neighbourhoods is important for the LS methods to work efficiently. A neighbourhood structure is defined as follows (Blum et al, 2008);

Definition 1: *A neighbourhood structure is a function* $\mathcal{N}: \boldsymbol{X} \rightarrow \mathbf{2}^{\boldsymbol{X}}$ *that assigns to every* $\boldsymbol{x} \in \boldsymbol{X}$ *a set of neighbours* $\mathcal{N}(\boldsymbol{x}) \subseteq \boldsymbol{X}$. $\mathcal{N}(\boldsymbol{x})$ *is called the neighbourhood of* $\boldsymbol{x}$. *Often, neighbourhood structures are implicitly defined by specifying the changes that must be applied to a solution* $\boldsymbol{x}$ *in order to generate all its neighbours. The application of such an operator that produces a neighbour* $\boldsymbol{x}' \in \mathcal{N}(\boldsymbol{x})$ *of a solution* $\boldsymbol{x}$ *is commonly called a* **move**.

The introduction of a neighbourhood structure allows the definition of the local minimum (local optimum in **(1.3)**) concept (Blum et al, 2008);

Definition 2: *A locally minimal solution (or local minimum) with respect to a neighbourhood structure* $\mathcal{N}$ *is a solution* $\hat{\boldsymbol{x}}$ *such that* $\forall\, \boldsymbol{x} \in \mathcal{N}(\hat{\boldsymbol{x}});\ \boldsymbol{f}(\hat{\boldsymbol{x}}) \leq \boldsymbol{f}(\boldsymbol{x})$. *We call* $\hat{\boldsymbol{x}}$ *a strictly locally minimum solution if* $\boldsymbol{f}(\hat{\boldsymbol{x}}) < \boldsymbol{f}(\boldsymbol{x})\ \forall\, \boldsymbol{x} \in \mathcal{N}(\hat{\boldsymbol{x}})$.

The most basic local search method is the iterative improvement local search where a move is only made if there is a better solution than the current solution and the algorithm stops once a local minimum is found.

LS starts at a given initial solution, $\boldsymbol{x_0}$ replacing the current solution, $\boldsymbol{x}$ by a neighbour $\boldsymbol{x'}$ that improves the objective function at each iteration and the search process stops when all the candidate neighbours are worse than the current solution, indicating the local minimum is reached (Talbi, 2009). **Algorithm 2.0** illustrates the iterative improvement local search.

There are a few possible termination conditions for LS (Consoli and Darby-Dowman, 2007):

1. Maximum time is reached.
2. Maximum total number of iterations is reached.
3. Finding a solution with a better objective function compared to the threshold value.
4. Maximum number of iterations is reached without any improvement in the objective value.

$\boldsymbol{x} = \boldsymbol{x_0}$; /*Generate an initial solution $\boldsymbol{x_0}$ */
Generate $(\boldsymbol{N}(\boldsymbol{x}))$; /*Generation of candidate neighbours*/
Repeat
    $\boldsymbol{x} = \boldsymbol{x'}$; /*Select a better neighbour $\boldsymbol{x'} \in \boldsymbol{N}(\boldsymbol{x})$ */
    If $\boldsymbol{f}(\boldsymbol{x'}) < \boldsymbol{f}(\boldsymbol{x})$ then
        $\boldsymbol{x} \leftarrow \boldsymbol{x'}$
    End if
Until $\boldsymbol{f}(\boldsymbol{x'}) \geq \boldsymbol{f}(\boldsymbol{x}); \forall \boldsymbol{x'} \in \boldsymbol{N}(\boldsymbol{x})$
Output Final solution found (local optimum)

Algorithm 2.0: Template of a local search algorithm

LS methods are simple and require little usage of memory. Although it may be preferred for its simplicity, LS does possess some weaknesses.

The most disadvantageous characteristic of LS is its inability to escape from the local minimum (Pirlot, 1996). The following **Figure 2.0** illustrates the existence of a global minimum that cannot be obtained under the descent rule of LS.

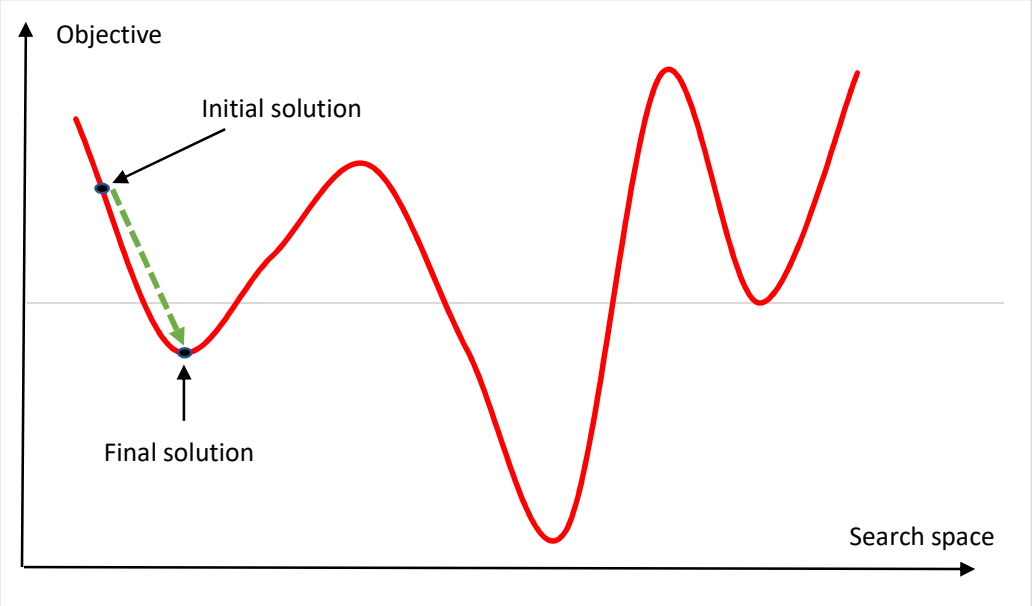

Figure 2.0: Local search (steepest descent) behaviour in a given landscape (Talbi, 2009)

2.4 Metaheuristics based on local search

Iterative improvement local search often resulted in unsatisfactory results because the quality of the solutions found relies heavily on the starting point of the local search process (Blum et al., 2008). The search might end up in a low quality local minimum if the starting point is poorly chosen.

Therefore, metaheuristics aimed to effectively and efficiently explore the search space. There is no standard definition of metaheuristics and the following are some definitions taken from the literature:

"A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions."

I. Osman and G. Laporte (1996)

"A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method."

S. Vos et al.(1999)

"Metaheuristics are typically high-level strategies which guide an underlying, more problem specific heuristic, to increase their performance. The main goal is to avoid the disadvantages of iterative improvement and, in particular, multiple descent by allowing the local search to escape from local minima. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more "intelligent" way than just providing random

initial solutions. Many of the methods can be interpreted as introducing a bias such that high quality solutions are produced quickly. This bias can be of various forms and can be cast as descent bias (based on the objective function), memory bias (based on previously made decisions) or experience bias (based on prior performance). Many of the metaheuristic approaches rely on probabilistic decisions made during the search. But, the main difference to pure random search is that in metaheuristic algorithms randomness is not used blindly but in an intelligent, biased form."

Stützle (1999)

"A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem."

Metaheuristics Network (2000)

"...metaheuristics are strategies, approximate and usually non deterministic, that guide the search process to efficiently explore the search space in order to find (near-) optimal solutions, using techniques which range from simple local search procedures to complex learning processes. They are not problem-specific, can incorporate mechanisms to avoid "traps" (local optima), may use domain-specific knowledge to explore the best promising areas and finally they can memorize the search experience in order to guide the future search (long/short-time form of memory)."

Consoli and Darby-Dowman (2007)

One characteristic that stands out from the definitions is that the authors describe metaheuristics to be a higher level of search technique. A good metaheuristic often aims to quickly find

promising regions containing high quality solutions but at the same time has to avoid spending too much time on a specific region by balancing the following concepts:

(1) Diversification: focusing on the search space exploration at the global scale to generate more diverse solutions.

(2) Intensification: exploiting the information of the current solution found, the search process focuses more in that specific local region.

This section will continue with discussion on the important metaheuristics that were based on the local search methods.

2.4.1 Simulated annealing

Simulated annealing (SA) is a search technique that was inspired by a statistical physics procedure. The work by Kirkpatrick et al. in 1983 marks the beginning of the application of this technique on CO problems and has since been widely used in the CO field. SA is based on the principles of statistical mechanics of the annealing process of the heating and cooling of a substance to obtain a strong crystalline structure (Talbi, 2009). SA aims to escape the local optima by allowing some degradation of a solution under a certain condition.

This search heuristic starts from an initial solution $\boldsymbol{x} \in \boldsymbol{X}$ and proceeds in several iterations where at each iteration, a random neighbour $\boldsymbol{x'}$ is generated. Any move that improves the objective function is always accepted, $\boldsymbol{f(x') < f(x)}$ and a non-improving neighbour whose solution is worse than the current solution is accepted given a probabilistic rule. The probabilistic rule uses a control parameter called temperature, $\boldsymbol{T}$. The search heuristic ceases when the stopping criterion is met. One example of the stopping criteria is the final temperature.

One important feature of SA is the cooling schedule. The cooling schedule plays an important role in the performance of SA and consists of the following parameters:

a) Initial temperature, $\boldsymbol{T_{max}}$: must not be too high to enable random search but high enough to allow moves to the neighbouring state.

b) Equilibrium state: a number of sufficient moves must be applied.

c) A cooling function: better solutions are achieved when the temperature is slowly decreasing but with substantial computational time.

d) Final temperature, $\boldsymbol{T_{min}}$: in theory, the final temperature is equal to 0.

The steps of a SA heuristic are illustrated in the following **Algorithm 2.1**.

Input: Cooling schedule.
$\boldsymbol{x = x_0}$**;** /* Generation of the initial solution */
$\boldsymbol{T = T_{max}}$; /* Starting temperature */
Repeat
  Repeat /* At a fixed temperature */
    Generate a random neighbour $\boldsymbol{x'}$;
    If $\boldsymbol{f(x') < f(x)}$ Then accept $\boldsymbol{x = x'}$ /* Accept the neighbour solution */
    Else Accept $\boldsymbol{x'}$ using a probabilistic rule;
  Until Equilibrium condition
  /* e.g. a given number of iterations executed at each temperature $\boldsymbol{T}$ */
  $\boldsymbol{T = g(T)}$; /* Temperature update */
Until Stopping criteria satisfied /* e.g. $\boldsymbol{T < T_{min}}$ */
Output: Best solution found.

Algorithm 2.1: Template of simulated annealing (Talbi, 2009).

Apart from its ability to escape the local optima, SA also has the advantage of being simple and easy to implement.

2.4.2 Tabu search

Tabu search (TS) is a deterministic local search strategy that was first proposed by Glover in 1986. This search heuristic is best known for its characteristic of using the short-term memory *tabu list* that stores the knowledge of the previous search process. Since TS explores the neighbourhood in a deterministic way, it discards any previous visited solution or move to escape from the local optimum by managing the search through the information in the *tabu list* to prevent cycling.

Another important feature of TS is the *aspiration criterion*. Since the tabu list may be too restrictive, there is a possibility of rejecting a "good" move that is tabu. The aspiration criterion enables the tabu solutions to be accepted under some condition. The commonly used aspiration criteria are (Talbi, 2009):

a) selecting tabu solutions that generate better solutions than the best found solution.
b) a tabu move that yields a better solution among the set of solutions of a given attribute.

Some advanced mechanisms are commonly included to deal with the intensification and diversification of the search (Talbi, 2009):

1. Intensification (medium-term memory): Stores the elite solutions found during the search to give priority to attributes of the set of elite solutions.
2. Diversification (long-term memory): Stores the information of the visited solutions along the search and explores the unvisited areas in the solution space.

TS starts by selecting an initial solution $\boldsymbol{x_0}$. A move is made from the current solution $\boldsymbol{x}$, whose neighbourhood $\boldsymbol{N(x) \subset X}$ to a better solution $\boldsymbol{x' \in N(x)}$. At each iteration, the tabu list is updated. When the local optimum is reached, the search continues by selecting worse candidate solutions from a modified neighbourhood $\boldsymbol{N^*(x)}$, composed by the short and long term memory structures by maintaining a selective history of the states encountered during search (Glover and Martí, 2006), until the stopping criteria are satisfied. **Algorithm 2.2** demonstrates a TS process.

$\boldsymbol{x} = \boldsymbol{x_0}$; /* Initial solution */

Initialise the tabu list, medium-term and long-term memories;

Repeat

Find best admissible neighbour $\boldsymbol{x'}$; /*non tabu or aspiration criterion holds*/

$\boldsymbol{x} = \boldsymbol{x'}$;

Update tabu list, aspiration conditions, medium and long-term memories;

If intensification_criterion holds **Then** intensification;

If diversification_criterion holds **Then** diversification;

Until Stopping criteria satisfied

Output: Best solution found.

Algorithm 2.2: Template of tabu search algorithm (Talbi, 2009)

Usually, the stopping criteria can be the maximum number of iterations or the maximum number of consecutive iterations without any improvement to the incumbent (best known) solution (Gendreau and Potvin, 2005).

2.4.3 Iterative local search

A local search method often gets stuck in local optimum with no improving neighbourhood available. Iterative local search (ILS) is the modification of the LS technique with repeated calls to the LS routine, each time starting from a different initial solution.

There are three basic elements that compose an ILS (Talbi, 2009):

1. LS: Any S-metaheuristics[3], either deterministic or stochastic, can be used in the ILS framework. The search procedure acts as the black box in the mechanism as illustrated in **Figure 2.1**.

[3] Single-solution based metaheuristics such as simulated annealing, tabu search, variable neighbourhood search.

2. Perturbation method: The perturbation operator may appear as a large random move from the current solution but it should keep part of the current solution and perturb strongly the other part of the solution.
3. Acceptance criteria: Defines the conditions that the new local optima must satisfy before replacing the current solution.

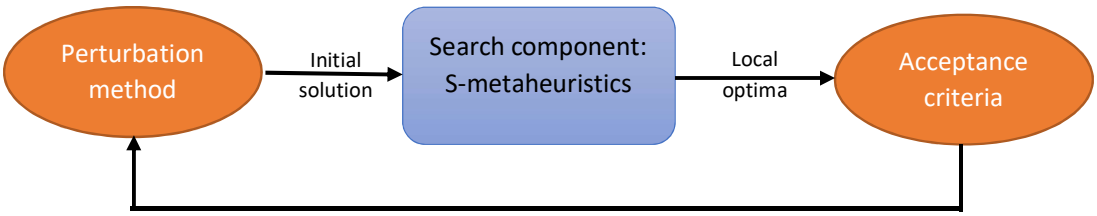

Figure 2.1: The search component as the black box in ILS (Talbi, 2009)

Let $\boldsymbol{f(x)}$ be the cost function of the optimisation problem to be minimised and $\boldsymbol{x \in X}$ are the candidate solutions. LS is applied to the initial solution $\boldsymbol{x_0}$, usually picked at random or is a solution returned by a greedy construction heuristic, and always returned the same solution $\boldsymbol{x_*}$ whose neighbourhood is $\boldsymbol{X^* \in X}$ (Gendrau and Potvin, 2010).

A move to an intermediate state $\boldsymbol{x' \in X}$ is obtained by applying a change or perturbation to the current $\boldsymbol{x_*}$. LS is then applied to the $\boldsymbol{x'}$, leading to the new local optimum $\boldsymbol{x'_* \in X^*}$. The solution $\boldsymbol{x'_*}$ becomes the next element in $\boldsymbol{X^*}$ if it passes an acceptance test. Otherwise, return to $\boldsymbol{x_*}$. **Figure 2.2** illustrates the walks of ILS and **Algorithm 2.3** gives the steps in ILS.

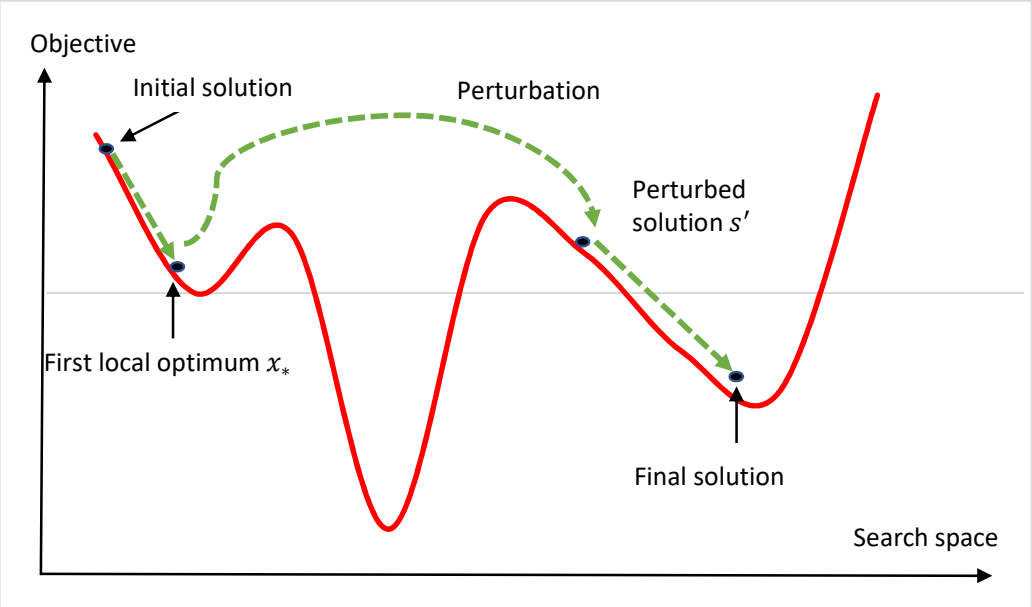

Figure 2.2: The principle of the iterated local search algorithm (Talbi, 2009)

$\boldsymbol{x}_*$ = local search $(\boldsymbol{x_0})$; /* Apply a given local search algorithm */

Repeat

$\boldsymbol{x}'$ = Perturb $(\boldsymbol{x}_*,$ search history); /* Perturb the obtained local optima */

$\boldsymbol{x}'_*$ = Local search $(\boldsymbol{x}')$; /* Apply local search on the perturb solution */

$\boldsymbol{x}_*$ = Accept $(\boldsymbol{x}_*, \boldsymbol{x}'_*,$ search memory); /* Accepting criteria */

Until Stopping criteria

Output: Best solution found.

Algorithm 2.3: Template of the iterated local search algorithm (Talbi, 2009)

The potential power of ILS lies in its biased sampling of the set of the local optima that depends both on (Gendrau and Potvin, 2010; Rajab, 2012):

1. the kinds of perturbations (change) – this can include as much possible problem-specific information by the modeller. Changes must not be too strong as it may lead to a random start, and not too small as it may return the same local optima. The perturbation strength

may be viewed as the number of the solution components; appropriate perturbation strength depends on the instance size. For example, in the partial optimisation, the strength is the number or percentage of the variable fixing.

2. the acceptance criteria – this can be adjusted empirically without knowing anything about the problem being optimised. An acceptance criteria decides on the move, whether to be taken or not, and can be used to control the balances between the intensification and diversification of the search process. An intensification of the search process is simply to accept a better solution while a diversification applies perturbation to the most recently visited local optima.

2.5 Other metaheuristics based on local search

2.5.1 Guided local search

Guided local search (GLS) is a method of escaping the local optimum by modification of the objective function to the problem. GLS is an intelligent search scheme for CO problems where the main feature of the approach is the iterative use of LS (Voudouris and Tsang, 1995). This method gathers information from various sources to guide the LS to promising parts of the search space.

A set of $\boldsymbol{m}$ features[4] $\boldsymbol{ft_i}$ $(\boldsymbol{i = 1, \dots, m})$ of a solution are first defined. GLS associates a cost $\boldsymbol{c_i}$ and a penalty $\boldsymbol{p_i}$ with each for the problem (Gendrau and Potvin, 2010). The objective function $\boldsymbol{f(x)}$ associated with the solution $\boldsymbol{x}$ is penalised if trapped by a local optimum as (Talbi, 2009):

$$\boldsymbol{f'(x) = f(x) + \lambda\alpha \sum_{i=1}^{m} I_i(x) p_i} \qquad (\mathbf{2.6})$$

where $\boldsymbol{\lambda}$ is the regularisation parameter and $\boldsymbol{I_i(x)}$ is an indicator function that determine whether a feature is present in the solution $\boldsymbol{x}$; 1 if $\boldsymbol{ft_i \in x}$, 0 otherwise. The regularisation parameter $\boldsymbol{\lambda}$ represents the relative importance of penalties with respect to the solution cost and is significantly important because it provides the means to control the influence of the information on the search process (Voudouris and Tsang, 1995). The coefficient $\boldsymbol{\alpha}$ is problem specific where it is used to balance the penalty in the objective function with the changes in the objective function.

GLS balances the intensification and diversification of the search process by focusing the search in the promising regions defined by lower costs of the features and avoids the generated local optima by penalising the features to diversify the search (Talbi, 2009).

[4]A feature of the problem defines a given characteristic of a solution. Example of a feature is whether the candidate tour travels immediately from City A to City B (Gendrau and Potvin, 2010).

The selection of features and the way to penalise them play an important role. Given a local optimum $\boldsymbol{x}^*$, utility $\boldsymbol{u_i}$ associated with each feature $\boldsymbol{i}$ is calculated as (Talbi, 2009):

$$\boldsymbol{u_i}(\boldsymbol{x}^*) = \begin{cases} \boldsymbol{I_i}(\boldsymbol{x}^*)\dfrac{\boldsymbol{c_i}}{\boldsymbol{1+p_i}}; \text{if a given feature i is present in the local optimum } \boldsymbol{x}^* \\ 0 \text{ otherwise} \end{cases}$$

Algorithm 2.4 gives the template of the GLS.

Input: S-metaheuristic LS, $\boldsymbol{\lambda}$, Features $\boldsymbol{I}$, Costs $\boldsymbol{c}$.

$\boldsymbol{x} = \boldsymbol{x_0}$; /* Generation of the initial solution */

$\boldsymbol{p_i} = \boldsymbol{0}$; /* Penalties initialisation */

Repeat

Apply S-metaheuristic *LS*; /* Let $\boldsymbol{x}^*$ be the final solution obtained */

For each feature $\boldsymbol{i}$ of $\boldsymbol{x}^*$ **Do**

$\boldsymbol{u_i} = \frac{\boldsymbol{c_i}}{\boldsymbol{1+p_i}}$; /* Compute its utility */

$\boldsymbol{u_j} = \boldsymbol{max_{i=1,\ldots,m}}(\boldsymbol{u_i})$; /* Compute the maximum utilities */

$\boldsymbol{p_j} = \boldsymbol{p_j} + \boldsymbol{1}$; /* Change the objective function by penalising the feature $\boldsymbol{j}$ */

Until Stopping criteria

Output: Best solution found.

Algorithm 2.4: Template of the iterated local search algorithm (Talbi, 2009)

2.5.2 GRASP

The greedy randomised adaptive search procedure or more commonly known as GRASP was first introduced by Feo and Resende in 1989. GRASP is a set of solutions, generated from adding the best elements on the list ranked by the greedy function.

GRASP is a multistart (or iterative) metaheuristic, which for each of its iterations, consists of two phases (Gendrau and Potvin, 2010):

1. Construction: in this phase, an initial solution is constructed and a repair procedure is performed if the solution is not feasible.
2. Local search: Using the obtained feasible solution, the neighbourhood is investigated until a local minimum is found in this phase.

2.5.4 Variable neighbourhood search (VNS)

VNS was first introduced in 1997 by Mladenović and Hansen and since has undergone various developments and has been applied in countless fields. VNS relies heavily upon the following observations (Hansen et al, 2008):

Fact 1: *A local minimum with respect to one neighbourhood structure is not necessarily a local minimum for another neighbourhood structure.*

Fact 2: *A global minimum is a local minimum with respect to all possible neighbourhood structures.*

Fact 3: *For many problems local minima with respect to one or several neighbourhoods are relatively close to each other.*

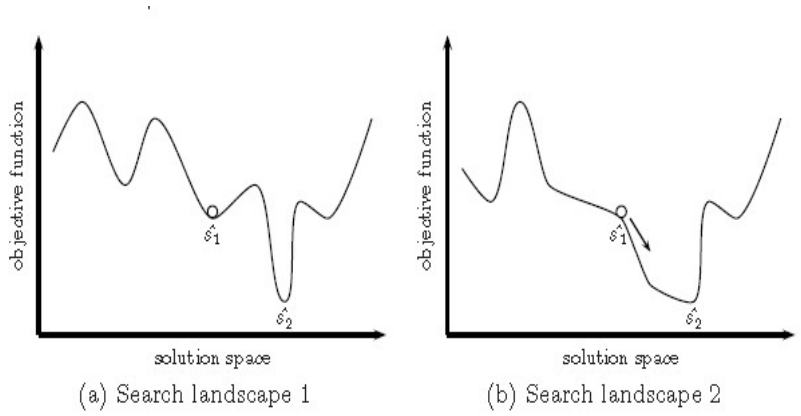

Figure 2.3: Two search landscapes defined by two different neighbourhood structures. On the landscape that is shown in (a), by the best-improvement local search stops at $\hat{s_1}$, while it proceeds until a better local minimum $\hat{s_2}$ on the landscape that is shown in (b). (Blum et al., 2008)

Figure 2.3 illustrates **Fact 1** in **Section 2.3** where a local minimum for neighbourhood structure $\boldsymbol{N_1}$ (search landscape 1) is not necessarily a local minimum for neighbourhood $\boldsymbol{N_2}$ (search landscape 2). Blum et al. (2008) outlined that the main idea of VNS is to define several neighbourhood structures and strategically swap between the different neighbourhood structures during the search process (diversification of search).

VNS exploits the systematic changes of neighbourhoods both in

(1) descent phase, to find a local minimum,

(2) perturbation phase to emerge from the corresponding valley (Hansen et al, 2008).

The basic schemes of VNS and its extensions are simple. VNS does not follow a trajectory but explores increasingly distant neighbourhoods of the current incumbent solution, jumping to a new solution if and only if there is an improvement (Hansen and Mladenović, 2001). Let $\boldsymbol{N_l}$, for $\boldsymbol{l} = \boldsymbol{1}, \dots, \boldsymbol{l_{max}}$ denote the set of neighbourhood structures selected for the heuristic search and $\boldsymbol{N_l(x)}$ be the set of solutions in the $\boldsymbol{l^{th}}$ neighbourhood of x.

There are two main components in VNS; Variable Neighbourhood Descent (VND) and Reduced Variable Neighbourhood Search (RVNS).

2.5.4.1 Variable neighbourhood descent (VND)

VND can be translated as the deterministic component of VNS because the changes within the neighbourhood are performed in a deterministic way. To understand VND, it is crucial to understand the steepest descent heuristic. A steepest descent heuristic starts by choosing an initial solution $\boldsymbol{x}$ within a neighbourhood $\boldsymbol{N(x)}$ and travels in a steepest descending movement to the minimum of $\boldsymbol{f(x)}$ within the neighbourhood $\boldsymbol{N(x)}$. The heuristic stops if there is no descent movement available, otherwise it is iterated.

VND relies on **Fact 1** in **Section 2.3**; a local optimum x within neighbourhood $\boldsymbol{N_l(x)}$ does not mean that it is a local optimum within neighbourhood $\boldsymbol{N_{l+1}(x)}$. Therefore, for all $\boldsymbol{x \in X}$, a neighbourhood $\boldsymbol{N(x)}$ is defined, the heuristic search VND is the combination of the descent heuristic of the neighbourhoods. The algorithm of VND is given by **Algorithm 2.5**.

Initialisation.
Select the set of neighbourhood structures $\boldsymbol{N_l}$, for $\boldsymbol{l = 1, \dots, l_{max}}$, that will be used in the descent; find an initial solution x (or apply the rules to a given $\boldsymbol{x}$);
Repeat the following sequence until no improvement is obtained;

(1) Set $\boldsymbol{l \leftarrow 1}$;
(2) Repeat the following steps until $\boldsymbol{l = l_{max}}$;
    (a) *Exploration of neighbourhood*.
        Find the best neighbour $\boldsymbol{x'}$ of $\boldsymbol{x}$ ($\boldsymbol{x' \in N_l(x)}$);
    (b) *Move or not*.
        If the solution $\boldsymbol{x'}$ thus obtained is better than $\boldsymbol{x}$, set $\boldsymbol{x \leftarrow x'}$ and $\boldsymbol{l \leftarrow 1}$;
        otherwise, set $\boldsymbol{l \leftarrow l + 1}$;

Algorithm 2.5: Steps of the basic VND (Hansen et al, 2008).

2.5.4.2 Reduced variable neigbourhood search (RVNS)

Opposite to VND, RVNS can be viewed as the stochastic component of VNS. RVNS allows deeper exploration of those neighbourhoods close to the current local minimum $\boldsymbol{x}$, providing an approach to exploit **Fact 2** in **Section 2.3**. The initial point of this heuristic search is randomly selected in the $\boldsymbol{l^{th}}$ neighbourhood. This random selection is to avoid cycling which might occur in a deterministic environment.

After selecting the set of neighbourhoods $\boldsymbol{N_1(x), N_2(x), \dots, N_{l_{max}}(x)}$ that are centred around the current local minimum $\boldsymbol{x}$, an initial solution $\boldsymbol{x'}$ is picked at random from the first neighbourhood $\boldsymbol{N_1}$ and a stopping criterion is defined. If $\boldsymbol{f(x') < f(x)}$, the search is then re-centred to be around $\boldsymbol{x'}$ $(\boldsymbol{x \leftarrow x'})$. Otherwise, the search proceeds to the next neighbourhood. The search process continues, and after exploring all the neighbourhoods, it starts over with the first neighbourhoods until the stopping criterion is met. **Algorithm 2.6** demonstrates the step by step algorithm of RVNS.

Initialisation.
Select the set of neighbourhood structures $\boldsymbol{N_l}$, for $\boldsymbol{l = 1, \dots, l_{max}}$, that will be used in the search; find an initial solution $\boldsymbol{x}$; choose a stopping condition;
Repeat the following sequence until the stopping condition is met:

(1) Set $\boldsymbol{l \leftarrow 1}$;
(2) *Repeat* the following steps until $\boldsymbol{l = l_{max}}$;
   (a) *Shaking*. Generate a point $\boldsymbol{x'}$ at random from the $\boldsymbol{l}^{\text{th}}$ neighbourhood of $\boldsymbol{x}$ $(\boldsymbol{x' \in N_l(x)})$;
   (b) *Move or not*. If this point is better than the incumbent, move there $(\boldsymbol{x \leftarrow x'})$, and continue the search with $\boldsymbol{N_1(l \leftarrow 1)}$; otherwise, set $\boldsymbol{l \leftarrow l + 1}$;

Algorithm 2.6: RVNS algorithm (Hansen et al, 2008).

The differences between the VND and RVNS are summarised in the following **Table 2.0**.

	VND	RVNS
Changes in neighbourhood	In a deterministic behaviour.	Randomly selected point from $\boldsymbol{N_l(x)}$.
Iterations	Repetition is made until no improvement in $\boldsymbol{f}$.	Repetition is made until the stopping criterion is met.
Parameter	Sole parameter usually the maximum number of neighbourhood $\boldsymbol{l_{max}}$	Have two or more parameters; number of neighbourhood; $\boldsymbol{l_{max}}$ stopping criterion (e.g.: time limit); $\boldsymbol{t_{max}}$
Problem size	Applicable to all sizes of instances.	Useful for large instances where local search can be costly/expensive.

Table 2.0: Differences between VND and RVNS

2.5.4.3 Basic variable neighbourhood search (BVNS)

Basic variable neighbourhood search (BVNS) explores the neighbourhoods both in a deterministic and a stochastic way. The pre-selected neighbourhood structures are centred around the selected initial solution point $\boldsymbol{x} \in \boldsymbol{X}$. A random point $\boldsymbol{x'}$ is generated and a local search method is applied to find the local optimum $\boldsymbol{x''}$. If there is an improvement in the objective function $\boldsymbol{f}; \boldsymbol{f}(\boldsymbol{x''}) \leq \boldsymbol{f}(\boldsymbol{x})$, then a move is made; $\boldsymbol{x} \leftarrow \boldsymbol{x''}$. Otherwise, the search process continues with the next neighbourhood. **Algorithm 2.6** illustrates the steps of the BVNS.

The characteristic of the deterministic component in BVNS allows an intensive search procedure within the neighbourhood, by using a local search method to increase the possibility of finding an improved solution from the randomly selected candidate solution $\boldsymbol{x'}$.

Initialisation. Select the set of neighbourhood structures $\boldsymbol{N_l}$, for $\boldsymbol{l = 1}$, … , $\boldsymbol{l_{max}}$, that will be used in the search; find an initial solution $\boldsymbol{x}$ and improve it by using RVNS; choose a stopping condition;
Repeat the following sequence until the stopping condition is met:

(1) Set $\boldsymbol{l \leftarrow 1}$;
(2) *Repeat* the following steps until $\boldsymbol{l = l_{max}}$;
   (a) *Shaking*. Generate a point $\boldsymbol{x'}$ at random from the lth neighbourhood of $\boldsymbol{x}$ ($\boldsymbol{x' \in N_l(x)}$) ;
   (b) *Local search*. Apply some local search method with $\boldsymbol{x'}$ as initial solution; denote with $\boldsymbol{x''}$ the obtained local optimum;
   (c) *Move or not*. If the local optimum $\boldsymbol{x''}$ is better than the incumbent $\boldsymbol{x}$, move there ($\boldsymbol{x \leftarrow x''}$), and continue the search with $\boldsymbol{N_l}$ ($\boldsymbol{l \leftarrow 1}$); otherwise, set $\boldsymbol{l \leftarrow l + 1}$;

Algorithm 2.7: Steps of the BVNS (Hansen et al, 2008).

One important step in VNS is *shaking*. To demonstrate the shaking process, consider a 4-cardinality tree problem[5] in **Figure 2.4**. The aim is to find 4 edges that give minimum weight. **Figure 2.5** illustrates how BVNS solves a 4-cardinality tree problem. The figure at the bottom right in each square represents the total weight.

In **Step 0**, the total weight given by a local search (LS) is 40. Shaking in **Step 1**, by removing and adding an edge at random, give the total weight of 60. After a LS, again the total weight is given as 40. Another shaking process is performed in **Step 3** which led to the total weight of 39. This process is repeated to improve the solution obtained. Note that shaking in steps **1, 3, 5** and **7** helps the search process to escape from being stuck in the local minimum, leading to a better solution in step **8**.

[5] $\boldsymbol{k}$-cardinality tree problem is a CO problem of finding a subtree, in a given graph, of minimal weight with a fixed number of minimal $\boldsymbol{k}$ edges. This illustration is taken from Jörnsten and Lokketangen (1997).

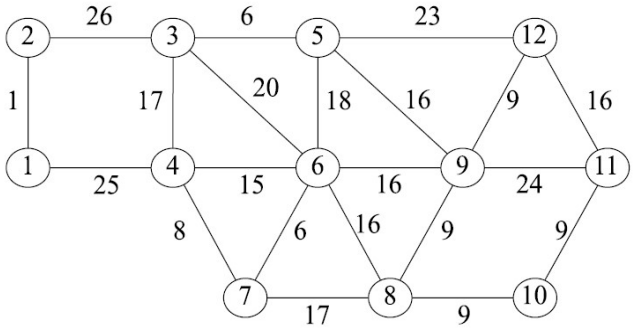

Figure 2.4: 4-cardinality tree problem (Hansen et al., 2009).

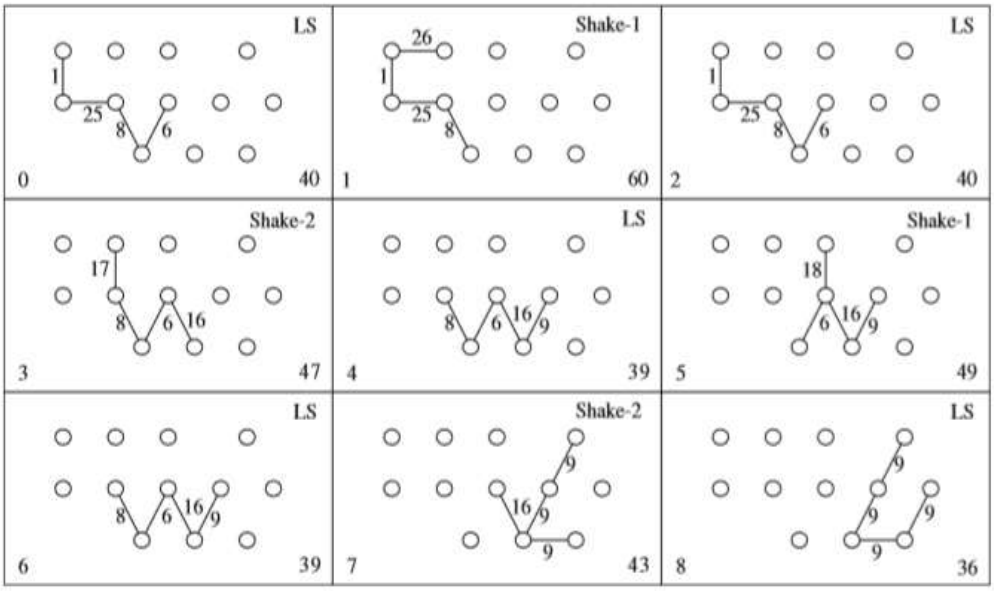

Figure 2.5: Steps of the BVNS for solving 4-cardinality tree problem (Hansen et al., 2009).

2.5.4.4 General variable neighbourhood search (GVNS)

When LS in Step 2(b) in **Algorithm 2.7** is replaced with VND, general variable neighbourhood search (GVNS) is obtained. **Algorithm 2.8** gives the details of GVNS.

There are other extensions of VNS that are not covered in this research such as Skewed VNS and Variable neighbourhood decomposition search. For interested readers, please refer to Hansen et al. (2010) for details.

Initialisation. Select the set of neighbourhood structures $\boldsymbol{N_l}$, for $\boldsymbol{l = 1, \dots, l_{max}}$, that will be used in the shaking phase, and the set of neighbourhood structures $\boldsymbol{N'_s}$ for $\boldsymbol{s = 1, \dots, s_{max}}$ that will be used in the local search; find an initial solution $\boldsymbol{x}$ and improve it by using RVNS; choose a stopping condition;
Repeat the following sequence until the stopping condition is met:

(1) Set $\boldsymbol{k \leftarrow 1}$;
(2) *Repeat* the following steps until $\boldsymbol{l = l_{max}}$;
  (a) *Shaking*. Generate a point $\boldsymbol{x'}$ at random from the $\boldsymbol{l}^{\text{th}}$ neighbourhood $\boldsymbol{N_l(x)}$ of $\boldsymbol{x}$;
  (b) *Local search by VND*.
    (b1) *Set* $\boldsymbol{\leftarrow 1}$;
    (b2) Repeat the following steps until $\boldsymbol{= l_{max}}$;

. *Exploration of neighbourhood*. Find the best neighbour $\boldsymbol{x''}$ of $\boldsymbol{x'}$ in $\boldsymbol{N'_s(x')}$.
. *Move or not*. If $\boldsymbol{f(x'') < f(x')}$ set $\boldsymbol{x' \leftarrow x''}$ and $\boldsymbol{\leftarrow 1}$; otherwise set $\boldsymbol{s \leftarrow s + 1}$;
  (a) *Move or not*. If this local optimum is better than the incumbent, move there ($\boldsymbol{x' \leftarrow x''}$), and continue the search with $\boldsymbol{N_1}$ ($\boldsymbol{l \leftarrow 1}$) ; otherwise, set $\boldsymbol{l \leftarrow l + 1}$;

Algorithm 2.8: Steps of the GVNS (Hansen et al, 2008).

There are occasions when the basic VNS does not provide desirable results and the following ways may aid in improving the VNS performance (Hansen et al., 2010):

a) First vs. best improvement.

   Experimentally compare the first and best improvement strategies within a local search. Based on previous experience; use the first improvement rule if the initial solution is chosen at random but if some constructive heuristic is used, use the best improvement rule.

b) Reduce the neighbourhood.

Unnecessary visits may lead to bad behaviour of the local search. Try to identify "promising" regions in the neighbourhood and focus visits to those regions; ideally, find a rule that automatically removes solutions with worse objective values than the current solution from the neighbourhood solutions.

c) Intensified shaking.

The trade-off between intensification and diversification is balanced in the *shaking* procedure. A completely random jump in the $\boldsymbol{k}^{\text{th}}$ neighbourhood may be too diversified for some problem instances. Therefore, developing a more efficient VNS sometimes requires intensive effort in checking how sensitive the objective function is to small changes (shaking) of the solution.

d) VND.

Develop a VND to substitute the local search routine (changing BVNS to GVNS) to analyse several possible neighbourhood structures, estimate their sizes, order them, try them out and keep the most efficient one.

e) Experiments with parameter settings.

The single parameter of VNS is $\boldsymbol{l_{max}}$, which should be tuned experimentally but the procedure often is not too sensitive to $\boldsymbol{l_{max}}$. One can fix the parameter value to be the value of some input parameter to obtain a parameter-free VNS. For example, set $\boldsymbol{l_{max}} = \boldsymbol{p}$ for the $\boldsymbol{p}$-median[6] problem.

[6] The $\boldsymbol{p}$-median problem concerns with locating $\boldsymbol{p}$ facilities to minimise average distance between the demand nodes and the nearest of the selected facilities (Daskin and Maass, 2015).

2.6 Summary

IP methods guarantee optimality of solutions. But to solely rely on IP methods may be too expensive for large scale CO problems, in terms of memory usage and computational time. This is because IP methods conduct search in an exhaustive manner by enumerating every possible solution for the CO problems.

LS methods on the other hand require very minimum usage of memory. They offer simple and easy implementation. Most local search methods use randomisation, i.e. to start with randomly or heuristically generated candidate solution and iteratively improve this candidate solution. This is to ensure the search process does not stagnate with unsatisfactory candidate solution (Hoos and Tsang, 2006). However, to randomly pick an initial point may lead to low quality of solutions and the likeliness of being stuck in local minimum may as well produce unsatisfactory solutions.

Both methods, exact algorithms and approximate approaches, have their strengths and weaknesses. In **Section 2.1** of this chapter, the advantages and disadvantages of IP and LS methods have been discussed. The strength and weakness of both methods can be seen as complementary and this has led to another branch of metaheuristics techniques called hybrid metaheuristics. This technique has received a lot of interest in recent years and has become a more popular choice among modellers in solving CO problems.

In the next chapter, the concept of hybrid metaheuristics will be introduced and discussed. The hybridisation of metaheuristics in this research is the collaborative combination of the LP relaxation (exact method) with VNS (metaheuristics technique). The next chapter will discuss how the strengths and weakness of the exact method and the metaheuristics technique become complementary to each other.

Chapter 3.0: Hybrid metaheuristics

It is noticeable that research in metaheuristics for CO problems is now focused more on the problem instead of being algorithm-oriented, sparking a lot of interest in the hybridisation of metaheuristics with other techniques for optimisation. The general idea behind hybrid metaheuristics is to combine different search techniques.

3.1 Introduction

Blum et al. (2011) described hybrid metaheuristics as algorithms that do not purely follow the paradigm of a single traditional metaheuristic where these approaches combine various algorithmic components, often originating from algorithms of other research areas of optimisation. Benefiting from synergy, hybrid techniques exploit the complementary character of different optimisation strategies. The following items **(i)** and **(ii)** are the definition of hybrid taken from the Merriam Webster dictionary and the items **(iii)** and **(iv)** are the definitions given by Wiktionary (Raidl, 2006):

i. Something heterogeneous in origin or composition
ii. Something (such as a power plant, vehicle, or electronic circuit) that has two different types of components performing essentially the same function
iii. Offspring resulting from cross-breeding different entities, e.g. different species
iv. Something of mixed origin or composition

There are several publications in the literature that provide taxonomies on hybrid metaheuristics or its subcategories (Cotta, 1998; Talbi, 2002; Blum et al., 2005; Cotta et al., 2005; El-Abd and Kamel, 2005; Puchinger and Raidl, 2005). **Figure 3.0** exhibits the classifications of hybrid metaheuristics summarised by Raidl (2006).

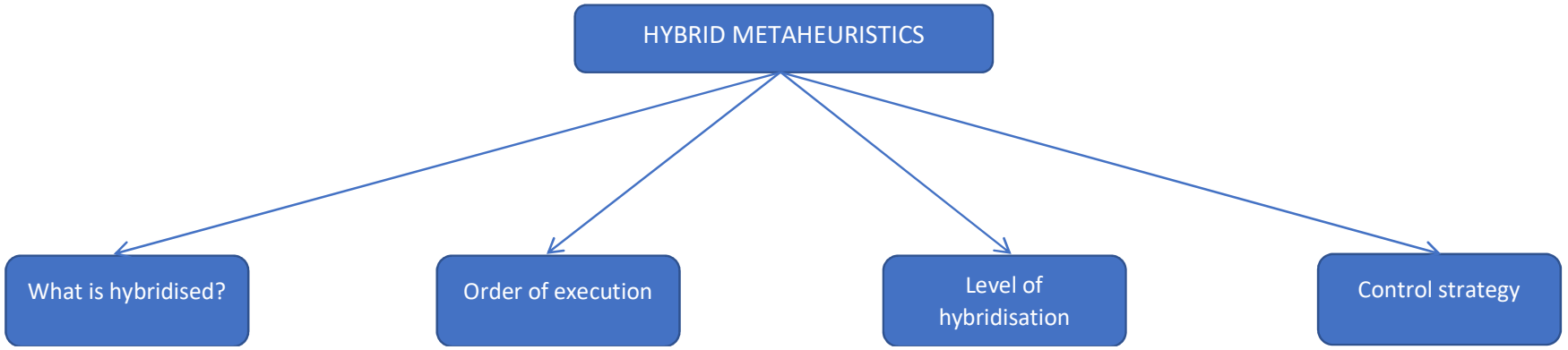

Figure 3.0: A summarised classification of hybrid metaheuristics (Raidl, 2006).

Talbi (2009) categorised the hybridisation of metaheuristics into 4 types of combinations of metaheuristics with:

a) Complementary metaheuristics.
b) Exact methods from mathematical programming approaches.
c) Constraint programming approaches.
d) Machine learning and data mining techniques.

3.1.1 Combining metaheuristics and ILP techniques for CO

As far as this research is concerned, only the combinations of the metaheuristics with the exact techniques will be discussed, precisely the sequential collaborative combinations approach. Blum et al. (2008) categorised the existing techniques of combining exact and metaheuristic algorithms for CO problems into two main categories:

1. Collaborative combinations: The algorithms only exchange information and are not part of each other. The two algorithms may be sequentially, intertwined, or be parallelly executed.
2. Integrative combinations: One technique is a subordinate embedded component of another technique and there is a distinguished master algorithm which can either be exact or an approximate algorithm with at least one technique as an integrated slave.

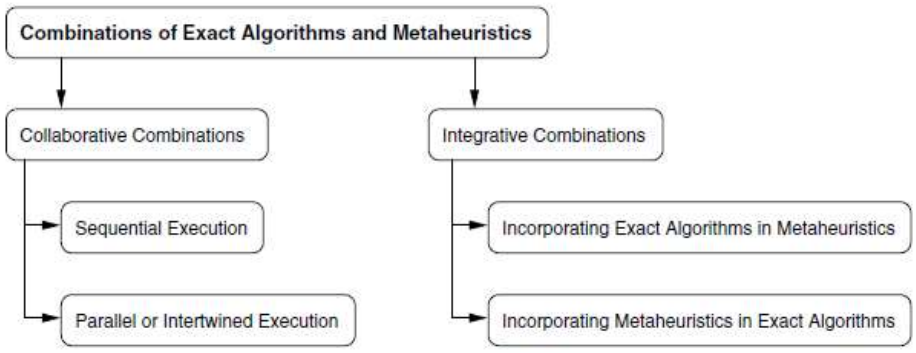

Figure 3.1: Combinations of Exact Algorithms and Metaheuristics (Blum et al, 2008).

Integer linear programming (ILP) is a promising combination for metaheuristic hybridisation in many ways including (Puchinger and Raidl, 2005; Raidl, 2006):

- ILP can be used to search large neighbourhoods.
- LP relaxations often generate valuable information that can be exploited to heuristically guide neighbourhood search, recombination, mutation, repair and/or local improvement.
- ILP can serve as merging solutions; subspaces defined by the merged attributes of two or more solutions can be searched by ILP techniques.
- A good lower and upper bound is important for techniques based on tree search. For a minimisation problem, relaxation methods often give the lower bound and heuristics or metaheuristics can be used to generate the upper bound.
- The identification of inequalities that have been violated by the current solution to the LP relaxation (but valid for the integer optimum) is often hard but can be approached using metaheuristics. The addition of these inequalities to the system helps to improve the bounds when the related LP is resolved.
- The identification of variables that are part of the model whose insertion improved the current solution but are currently not included when solving the problem is hard. Metaheuristics however manage to successfully do so when Puchinger and Raidl (2005) approached the multidimensional knapsack problem through the cooperation between the memetic algorithm and the branch-and-cut algorithm.
- Some promising approaches that tried to bring the spirit of metaheuristics through the idea of local search based metaheuristics into linear programming based branch and bound; for example local branching.

3.1.2 Previous works on collaborative combinations of the metaheuristics hybridisation

Applegate et al. (1998) create a restricted graph as the search space by merging the edge-sets of the solutions from deriving a set of diverse solutions by multiple runs of an iterated local search algorithm for the Traveling Salesman Problem (TSP). They solve the TSP to optimality and the solutions generated outperformed the best solution of the iterated local search.

Joslin and Clements (1999) introduced a column generation approach that uses squeaky wheel optimisation (SWO) to generate feasible solutions for the production-line scheduling problem. This initial step helps to identify elements of that solution that work well or poorly which is useful in determining the search direction. The combined heuristic and exact algorithms allow many good and not so good schedules in the recombination process for a higher quality schedule. The authors took advantage of the randomness of the SWO heuristic to escape from local optima by making large coherent moves when the local search algorithm searches for good schedules. IP techniques provide a kind of global optimisation that has no counterpart in local search which allows the best parts of different solutions to be combined. The computational results show that this approach dominates a tabu search algorithm, providing better quality solutions with faster run time.

In their work, Plateau et al. (2002) first generated a population of different feasible candidate solutions using an interior point method (IPM) with early termination. This set of solutions then serve as the initial population for a scatter search algorithm. Refinements concerning the use of improved directions given by several points of the population were introduced into the interior code process. The computational works on the 0-1 multiconstraint knapsack problems show that using IPM for IP allows the computation of the diversified fractional interior points in different parts of the constraint polyhedron. At the time of publication, the proposed method did provide attractive results where they compared their results to the best solution computed by Chu and Beasley (1998). The proposed hybrid method recorded an average execution time of 14.62

seconds compared to Chu and Beasley's genetic algorithm results that have an average execution time of 1267.4 seconds. The authors believe that the presented algorithm is a promising research direction.

Klau et al. (2004) combined the memetic algorithms (MA) with ILP to solve the prize-collecting Steiner tree problem. The objective function is to minimise the total cost of the edges in the subtree plus the total profit of all vertices outside of the subtree. Similar to the approach used by Applegate et al., the authors tightened the ILP relaxations by introducing cut constraints. They reduced the graph without changing the structure of the optimal solution by using the MA to provide the solution population as a starting point to solve the problem at hand. The modified ILP model is then solved by the branch-and-cut approach. The technique proposed by the authors managed to solve all of the benchmark instances from the literature to optimality, again proving that metaheuristic hybridisation is indeed a powerful tool.

Raidl and Feltl (2004) extended the work of Chu and Beasley by replacing the pure random initialisation with the constraint-ratio heuristic and LP relaxation solutions to the generalised assignment problem (GAP) under consideration to create a candidate population. Due to capacity constraint violations, the population created are likely to be infeasible and hence randomised repair and improvement operations are applied to have more meaningful and often feasible initial solutions for the genetic algorithms (GA). They also use a more intelligent heuristic operator instead of the standard position-wise mutation that has smaller probability to make a feasible solution infeasible or to worsen the capacity excess of an infeasible solution. Empirical results indicate that the initial solutions provided by the constraint-ratio and LP heuristics are more promising and led to the speeding up of the GA, generating better solutions compared to the original work by Chu and Beasley.

For the 0-1 multidimensional knapsack problem (MKP01)[7], Vasquez and Hao (2001) used LP relaxation to first obtain 'promising' continuous optima that served as the search space. Then, using tabu search, the binary areas within the designated search space are carefully and efficiently explored. They heuristically solved the MKP01 by introducing several additional constraints that fix the total number of items to be packed which resulted in the reduction and partition of the search space. Again, using the Chu and Beasley GA results as the benchmark, empirical work showed improved solutions for most of the instances. The authors suggest that their research is a good starting point in developing more improved algorithms for the MKP01 and the basic idea of the proposed algorithm could be explored to tackle other difficult CO problems.

Lin et al. (2004) use the combination of a genetic algorithm and a mixed integer program (GAMIP) to construct the minimal set of affine functions which describe the value function of the finite horizon partially observed Markov decision process (POMDP). POMDP is a generalisation of a Markov decision process that allows for noise-corrupted and costly observations of the underlying system states. A set of points is first generated and then the redundant points are eliminated using a component-wise domination procedure. The problem is solved as a MIP, generating the missing points of the value function. Empirical works indicated that GAMIP used 60 percent less time to construct the minimal set compared to the most efficient LP based exact solution method in the literature.

Heragu et al. (1994) developed several variants of the simulated annealing (SA) algorithm to solve the order picking problem. Some of the variants used optimisation techniques (convex hull algorithm and B&B) to guide the SA's search. The convex hull algorithm is used to develop the initial solution and B&B is used to limit the neighbourhood search in the SA algorithm. The authors compared the developed algorithms with the convex hull algorithm (known to be the most

[7] MKP01 consists in selecting a subset of n given objects in such a way that the total profit of the selected objects is minimised while a set of m knapsack constraints are satisfied (Vasquez and Hao, 2001). The classical knapsack problem 0-1 is the special case of the MKP01 with $m = 1$ i.e. the capacity of the knapsack.

efficient to solve the order picking problem), 2-opt algorithm (known to provide reasonably good solutions) and pure SA. The results obtained show that the variants that use optimisation techniques as a guidance for the SA's search do perform better than the convex hull or pure SA algorithms for most cases.

For a two-machine flowshop scheduling problem, Nagar et al. (1995) introduced a branch-and-bound (B&B) algorithm to guide the GA in its search for optimal solutions. The authors stated that if the machine's processing times have a small gap between the upper-bound and lower-bound, the problem can easily be solved using the B&B algorithm but if the gap is large, more computational time is required making the problem difficult to be solved. The presented B&B algorithm eliminates the schedules that are known to be sub-optimal, providing an initial population consisting of schedules that are likely to lead to optimal or near optimal solutions for the GA procedure. The elimination process also contributes to a faster convergence of the algorithm, reducing the run time of the problem. Computational works proved that the proposed algorithm is significantly superior to the pure approach of the B&B algorithm or the GA.

3.2 Design

Metaheuristics are reflected as a pre-processing or post-processing step for IP. Collaborative combinations of the hybrid metaheuristics is considered as a top level technique because the interactions between the different algorithms are strictly to exchange information from one algorithm to another, aiming to provide better solutions and no algorithm is contained or embedded within another algorithm (Puchinger and Raidl, 2006).

Instances of the information that is provided by the metaheuristics includes upper bounds, incomplete solutions, and subproblems while for the IP, the information provided can be partial solutions, optimal solutions for relaxed problems and many more (**Figure 3.2**)[8].

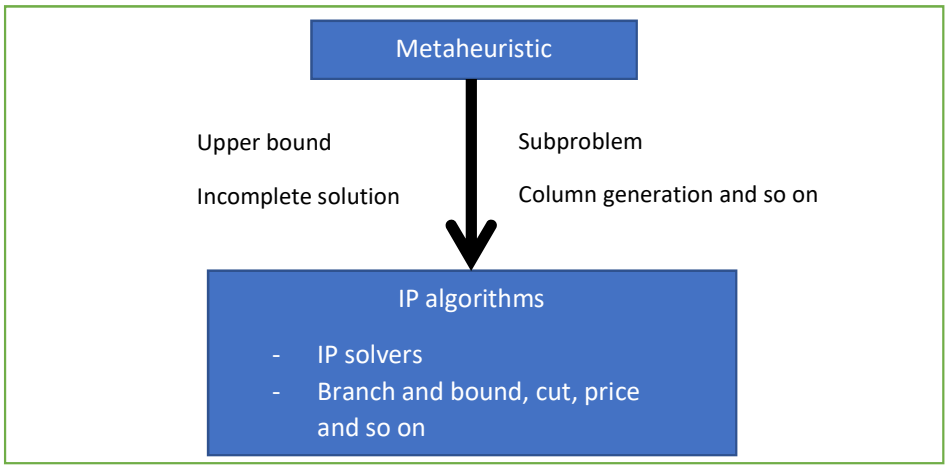

Figure 3.2: Information provided by metaheuristics to MP algorithms (Talbi, 2009).

Talbi (2009) outlined the information provided by the exact algorithms (IP) and metaheuristics:

- **Information provided by exact algorithms:**

i. Partial solution: to first provide partial solution that is completed by a metaheuristic.

ii. Problem reduction: carry out problem reduction before the metaheuristic solves the problem for example within a simplified objective function.

iii. Relaxed optimal solutions and their duals: the exploitation by metaheuristics on the information of the relaxed formulation optimal solution and its dual.

- **Information provided by metaheuristics:**

i. Finding a good upper bound that will be used by the IP algorithm in the bounding phase.

ii. Reduce the size of the original problem which then can be solved using the reduced problem. Reduction phase may be concerned with:

[8] In the original source, mathematical programming (MP) is used instead of IP. In the context of this research, we focus on the IP and thus use the term IP for consistency and to avoid confusion. IP is a branch of MP.

(a) Partitioning of decision variables: the decision variables are partitioned into $\boldsymbol{X}$ and $\boldsymbol{Y}$ where metaheuristics will fix the variables of the set $\boldsymbol{X}$ and the exact method will optimise the problem over the set $\boldsymbol{Y}$.

(b) Domain reduction: Reduce the domain of values that the decision variables can take where a metaheuristic will perform a domain reduction for the decision variable and then an exact method is used over the reduced domains.

In this chapter, the fundamental concepts of hybrid metaheuristics have been introduced. We discussed one approach of hybrid metaheuristic in particular, the combination of IP with metaheuristics. We also discussed why ILP is a promising technique for combination with metaheuristics.

In the next chapter, the readers will be introduced to two CO problems. The structures of the two different yet similar problems will be discussed and most important of all how the two CO problems can be conveniently expressed as a MBIP problem.

Chapter 4.0: Descriptions of the studied MBIP problems

There are two optimisation problems that are studied in this research. In the following section, we illustrate the characteristics of both problems and how to formulate them as MBIP problems.

4.1 Constrained index tracking problem (CITP)

Portfolio management is an investment activity that manages the asset allocation, aiming to achieve the investment objectives. It involves the activity of purchasing and selling assets, which incur transaction costs.

There are two broad basic strategies that are adopted by fund managers in managing portfolios; active management and passive management. Active management relies heavily on the fund managers' expertise and judgement to sell/purchase high performing assets, whilst passive management limits the fund managers' flexibility to assets' selection. Passive management focuses on the fund managers' role to conform to a closely defined set of criteria, usually to have the fund achieve a target return, and index tracking (IT) is a popular form of passive management (Beasley et al., 2003).

There are two types of index tracking model: full replication and partial replication. The investor will purchase all assets available in the index in the full replication whereas in the partial replication, only a subset of the assets will be purchased.

Provided that there are no changes to the universe of securities (market), full replication may be the simplest way to track an index, offering the advantages of (Kwiatkwoski, 1992):

- having no further transaction costs incurred
- exact matching is achieved
- easy management

However, this approach is subjected to some disadvantages (Beasley et al., 2003, Canakgoz and Beasley, 2009):

- Certain stocks (assets) may be held in very small quantities and if there are restrictions on the stocks' purchase/availability in the market, the purchase cost may become expensive.
- The index composition may need to be revised at times due to several reasons, for example, the growth of certain company or companies' mergers. When the index revision occurs, the stocks' holdings may need to be changed, incurring possibly higher transaction costs.
- Transaction costs relating to the sale and purchase of a stock need to be considered as they may be excessive as the full replication formulation does not have limits on the transaction costs.

Partial replication on the other hand offers more flexibility on the selection of assets to be included in the tracking portfolio but is subjected to loss of accuracy in tracking the intended index and may require frequent assets rebalancing. Nevertheless, this approach allows (Kwiatkwoski, 1992):

- easier adjustment of new issuance of assets
- smaller initial transaction costs compared to the full replication approach
- if the fund generates net income, it is possible to avoid unnecessary transaction costs in investing the income during the assets rebalancing

Canakgoz and Beasley (2009) described the constrained index tracking problem (CITP) as a decision problem of picking the subset of stocks (assets) to mirror/reproduce the performance of the index over time. Adopting the regression-based view, the index tracking problem can be formulated as a mixed binary integer linear program.

Beasley et al. (2003) presented an evolutionary heuristic technique to identify the best stocks to be placed in the portfolio. The authors made use of a population heuristic method to select the desirable solution for the index tracking problem. Their findings showed that holding 250 assets is a much more desirable proposition rather than replicating the full portfolio, holding approximately 7000 stocks.

4.1.2 Approach to CITP

The model represented in this research aims at solving the MIP rather than focusing on the problem's formulation. Suppose that over time $\boldsymbol{t = 0, 1, 2, \dots, T}$, the value of assets $\boldsymbol{i = 1, 2, \dots, N}$ are observed, as well as the value of the index that we wish to track. Adopting partial replication, the aim is to choose the best subset of assets to be included and related appropriate weights of the assets in the portfolio i.e. $\boldsymbol{K < N}$ where $\boldsymbol{K}$ is the limit on the unique assets and $\boldsymbol{N}$ are the assets available. Looking back at the index historical performance, we want to know what would be the best set of assets and their appropriate weights that would best track the index over the time period $[\boldsymbol{1, T}]$, assuming that the past is a guide to the future (Canakgoz and Beasley, 2009).

4.1.2.1 Definition of notation

4.1.2.1.1 Sets

$\boldsymbol{i = 1..N}$ be the universe of total assets available for investment

$\boldsymbol{t = 1..T}$ be the time period of the observed historical values for assets and index.

4.1.2.1.2 Data

$\boldsymbol{l_i}$ be the minimum proportion of the total investment held in asset $\boldsymbol{i}$ if investment is made on asset $\boldsymbol{i}$

$\boldsymbol{u_i}$ be the maximum proportion of the total investment held in asset $\boldsymbol{i}$ if investment is made on asset $\boldsymbol{i}$

$\boldsymbol{R_t}$ be the single period continuous time return for the index at time $\boldsymbol{t}$

$\boldsymbol{r_{t,i}}$ be the single period continuous time return for the asset $\boldsymbol{i}$ at time $\boldsymbol{t}$

$\boldsymbol{K}$ be the maximum number of the total assets to be included in the tracking portfolio

4.1.2.2 Decision variables

$\boldsymbol{x_i \geq 0}$ the proportion of asset $\boldsymbol{i}$ to be held in the tracking portfolio if investment is made on asset $\boldsymbol{i}$

$\boldsymbol{y_i = 0, 1}$ the binary decision variable indicating whether asset $\boldsymbol{i}$ is held; 1 if asset $\boldsymbol{i}$ is held, 0 otherwise

4.1.2.3 Objective function

One way of calculating the objective for the index tracking problem is to use the tracking error (**TE**). **TE** is the absolute difference between the return of a portfolio with the returns of the index that it was supposed to replicate. In this research, the **TE** is obtained by subtracting the index's returns, $\boldsymbol{R_t}$ with the portfolio's return, $\sum_i \boldsymbol{x_i r_{t,i}}$;

$$\sum_t \left| \left(\sum_i \boldsymbol{x_i r_{t,i}} - \boldsymbol{R_t} \right) \right| \qquad (\mathbf{4.0})$$

where $\boldsymbol{x_i}$ is the weight (proportion) of asset $\boldsymbol{i}$ in the portfolio. To best track the index, the aim is to minimise the cumulative difference between the two returns. We introduced the following variables into our formulation:

1. $\boldsymbol{O_t}$: indicates the difference between the two returns when the portfolio's return, $\sum_i \boldsymbol{x_i r_{t,i}}$ outperformed the index's return, $\boldsymbol{R_t}$; $\sum_i \boldsymbol{x_i r_{t,i}} > \boldsymbol{R_t}$
2. $\boldsymbol{U_t}$: indicates the difference between the two returns when the portfolio's return, $\sum_i \boldsymbol{x_i r_{t,i}}$ underperformed the index's return, $\boldsymbol{R_t}$; $\sum_i \boldsymbol{x_i r_{t,i}} < \boldsymbol{R_t}$

The objective function to the CITP:

$$\boldsymbol{TE} = \mathbf{min} \sum_{\boldsymbol{t}} (\boldsymbol{O_t} + \boldsymbol{U_t}) \qquad (\mathbf{4.1})$$

4.1.2.4 Constraints

$$\sum_{i=1}^{N} \boldsymbol{r_{t,i}\, x_i} = \boldsymbol{R_t} + \boldsymbol{o_t} - \boldsymbol{u_t} \quad \forall\, \boldsymbol{t} \qquad (\mathbf{4.2})$$

$$\boldsymbol{l_i y_i} \leq \boldsymbol{x_i} \leq \boldsymbol{u_i y_i} \quad \forall\, \boldsymbol{i} \qquad (\mathbf{4.3})$$

$$\sum_i \boldsymbol{x_i} = \mathbf{1} \qquad (\mathbf{4.4})$$

$$\sum_{\boldsymbol{i}} \boldsymbol{y_i} \leq \boldsymbol{K} \qquad (\mathbf{4.5})$$

Equation 4.2 gives the definition of the **TE**.

Equation 4.3 serves as a dual function. One is to ensure that the weight of any asset $\boldsymbol{i}$ that is not chosen to be in the portfolio is 0; $\boldsymbol{x_i = 0}$. The other function is that for any asset $\boldsymbol{i}$ selected, the weight of the asset $\boldsymbol{i}$ is between the minimum and maximum requirements of the asset proportion.

Equation 4.4 defines that the portfolio is fully invested.

The selection of assets should not exceed the maximum number $\boldsymbol{K}$ that we wish to have in the new portfolio and **Equation 4.5** ensures this.

4.2 Gas supply problem

The gas supply problem is a CO problem and is computationally challenging in nature. Gas usage is a function of weather which is uncertain. Due to the uncertainty inherent in the input parameters, careful considerations are needed to model the gas supply problem to ensure these uncertainties are accounted for.

4.2.1 Approach to the gas supply problem

The focus in solving this problem will be on the strategic decisions of buying and storing gas as well as the setup of the storage facilities across the locations that will minimise the total cost in supplying gas. The formulation of the model is a large stochastic mixed binary integer programming problem, which aims to select potential locations to open the gas storage facilities.

The formulation given is a simplified version of Koberstein et al (2011) formulation. The purpose of this model is to get an insight idea in developing an approach to solving the complex model of the combinatorial problem. Although simplified, the model does incorporate realistic features of:

1. The multi distribution system from facilities to customers
2. The capacity capabilities of the facilities
3. Transportation costs

The resulting formulation is a large stochastic mixed binary integer programming problem, involving a huge number of decision variables and constraints which are difficult to manage.

4.2.1.1 Definition of notation

4.2.1.1.1 Sets

$\boldsymbol{t = 1..T}$ be the time periods

$\boldsymbol{l = 1..L}$ be the locations

$\boldsymbol{f = 1..S}$ be the forecasted scenarios of the weather

$\boldsymbol{K_l}$ the subset of locations that can supply location $\boldsymbol{l}$ $\forall\, \boldsymbol{l = 1..L}$

4.2.1.1.2 Data

$\boldsymbol{D'_{t,l,s}}$ the amount of the forecasted gas demand at time $\boldsymbol{t}$ under forecasted scenario $\boldsymbol{s}$ at location $\boldsymbol{l}$ $\forall\, \boldsymbol{t, l, s}$

$\boldsymbol{P'_{t,l,s}}$ the forecasted gas price at time $\boldsymbol{t}$ under forecasted scenario $\boldsymbol{s}$ at location $\boldsymbol{l}$ $\forall\, \boldsymbol{t, l, s}$

$\boldsymbol{d_{l,l1}}$ distance between location $\boldsymbol{l}$ and $\boldsymbol{l1}$ $\forall\, \boldsymbol{l, l1} \in \boldsymbol{K_l}$

$\boldsymbol{C}$ the cost per unit gas to store gas

$\boldsymbol{F}$ the cost to set up a storage facility

$\boldsymbol{SC_{l,l1}}$ the cost per unit gas to ship gas from location $\boldsymbol{l}$ to location $\boldsymbol{l1}$ $\forall\, \boldsymbol{l1} \in \boldsymbol{K_l}, \forall \boldsymbol{l}$

$\boldsymbol{\pi_s}$ the probability of forecasted scenario $\boldsymbol{s}$ occurring

$\boldsymbol{M}$ the maximum storage allowed during any time period $\boldsymbol{t}$

4.2.1.1.3 Decision variables

$x_{t,l,s} \geq 0$ the amount of gas purchased at time t under forecasted scenario s at location $l \quad \forall t, l, s$

$z_{t,l,s} \geq 0$ the amount of gas stored at time t under forecasted scenario s at location $l \quad \forall t, l, s$

$i_{t,l1,l,s} \geq 0$ the amount of gas shipped from location $l1$ to l under forecasted scenario s at time t (given that it is possible to ship from location $l1$ to l) $\forall t, s, l1 \in K_l, l$ and $l \neq l1$

$y_l = 0, 1$ the binary decision variable indicating whether to install a facility at location l; 1 if facility is installed at l, 0 otherwise

4.2.1.1.3 Objective function

Minimise

$$\sum_{t=1}^{T}\sum_{l=1}^{L}\sum_{s=1}^{S} x_{t,l,s} \times P'_{t,l,s} \times \pi_s + \sum_{t=1}^{T}\sum_{l=1}^{L}\sum_{s=1}^{S} z_{t,l,s} \times C \times \pi_s + \sum_{t=1}^{T}\sum_{l=1}^{L}\sum_{l1 \in K_l}\sum_{s=1}^{S} i_{t,l1,l,s} \times d_{l,l1} \times SC_{l,l1} \times \pi_s + \sum_{l=1}^{L} y_l \times F \quad (4.6)$$

4.2.1.3 Constraints

$$D'_{1,l,s} + z_{1,l,s} + \sum_{l1 \in K_l} i_{1,l1,l,s} = x_{1,l,s} + \sum_{l1 \in K_l} i_{1,l,l1,s} \quad \forall l, s \quad (4.7)$$

$$D'_{t,l,s} + z_{t,l,s} + \sum_{l1 \in K_l} i_{1,l1,l,s} = x_{t,l,s} + z_{t-1,l,s} + \sum_{l1 \in K_l} i_{1,l,l1,s} \quad \forall t \geq 2, l, s \quad (4.8)$$

$$z_{t,l,s} \leq M \times y_l \quad \forall t, l, s \quad (4.9)$$

$$x_{1,l,1} = x_{1,l,s} \quad \forall l, s \quad (4.10)$$

Equation 4.6 represents the objective function of the model, where we represent the as total cost being the cost of buying the gas units, cost of storing the gas units, cost of transporting the gas units from location $\boldsymbol{l1}$ to location $\boldsymbol{l}$ and the cost of setting up the storage facility at location $\boldsymbol{l}$.

Equations 4.7 and **4.8** ensure that all the buying, storing and transporting gas units meet the demands at each location $\boldsymbol{l}$.

Equation 4.9 is the constraint on the maximum units of gas that can be stored at time $\boldsymbol{t}$ which can be translated as the capacity of the storage and determines whether a gas storage facility is installed.

Equation 4.10 involves the non-anticipativity constraints where under the Here and Now theory, all the decisions made today must be the same for each forecasted scenario $\boldsymbol{s}$.

4.2.2 Representation of uncertainty

The decisions on gas purchases and the storing of gas as well as the gas storage facility set up depend heavily on the gas price and gas demand. These two parameters however are uncertain and thus we rely heavily on the future predictions based on the historical data. In order to capture these uncertain elements in a proper manner, we create a model to forecast the future prices and demands. We have created 100 possible scenarios for corresponding gas prices and gas demands.

The gas supply model proposed is divided into two stages. Stage 1 deals with the activities that are deterministic where the gas demand and gas price are known beforehand. Also, the decision to set up the storage facilities is made at this stage. Stage 2 takes into consideration the stochastic part of the gas supply problem where the activities are decided upon using the scenarios generated of the gas demand and gas price.

4.2.3 Motivation for the reduction of shipping variables

The shipping variable denoted by $\boldsymbol{i_{t,l1,l,s}}$ represents the amount of gas that can be injected to location $\boldsymbol{l}$ when the gas supply at location $\boldsymbol{l}$ is not sufficient. However, this activity would only be considered when the cost of shipping the gas from location $\boldsymbol{l1}$ is cheaper than purchasing the gas at location $\boldsymbol{l}$ itself.

Due to the complexity issues, the shipping variable $\boldsymbol{i_{t,l1,l,s}}$ is reduced in dimension by allowing only the $\boldsymbol{n}$ nearest locations to ship gas to location $\boldsymbol{l}$. The reason for limiting only the $\boldsymbol{n}$ nearest locations $\boldsymbol{l1}$ to supply gas to the location $\boldsymbol{l}$ is that the cost of supplying gas does depend on the distance between location $\boldsymbol{l}$ and $\boldsymbol{l1}$. The further the distance between the two locations, the higher the transportation cost will be and this will cause the cost of supplying gas to be more expensive, making that location as a poor or less desirable option. For each location $\boldsymbol{l}$, $\boldsymbol{K_l}$ is the set of potential connections and let $\boldsymbol{d_{l,k}}$ be the $\boldsymbol{n^k}$ farthest distance from location $\boldsymbol{l}$;

$$\boldsymbol{K_l' = \{j \in L | d_{l,l1} \leq d_{l,k}\}} \qquad \forall\, \boldsymbol{l} \qquad (\mathbf{4.11})$$

Another reason for introducing this constraint is that due to the availability of at each location, it is not possible for the cheapest location to be the sole supplier for all of the other locations. Therefore for practical purposes, only those locations $\boldsymbol{l1}$ that are near to the location $\boldsymbol{l}$ should be considered.

Before a constraint is implemented on the shipping amount variable $\boldsymbol{i_{t,l1,l,s}}$, the dimension of this variable is:

$$\boldsymbol{i_{t,l1,l,s} = 20 \times 50 \times 50 \times 100 = 5\,000\,000}$$

After we allowed only the 15 nearest locations to supply gas to the location $\boldsymbol{l}$, the dimension of this shipping variable $\boldsymbol{i_{t,l1,l,s}}$ is reduced to 70 percent of its original size:

$$\boldsymbol{i_{t,l1,l,s} = 20 \times 15 \times 50 \times 100 = 1\,500\,000}$$

4.2.4 Assumptions and limitations

Time 1 purchase decisions

At time 1, the gas price and gas demand are known with certainty. The deterministic decision at this stage is a single purchase decision at each location that minimises the total cost.

Storage facility set up decisions

The decision variable on setting up the storage facility is deterministic as well. This is a large capital investment decision and is hence a strategic decision. In an earlier version of the model, we considered this to be time dependent. All of our investigations showed that the decision it made in the first time period remained constant throughout later time periods. Hence, the model has been simplified by considering this set up decision only in the first time period.

Ad hoc purchase decisions

The ad hoc purchase depends on the forecasted spot price. This activity only happens given that the spot price is relatively cheaper compared to other available alternatives which are injections from other locations.

Storage amount decisions

The amount of the gas stored depends on the forecasted future gas price and the capacity of the storage facility.

Shipping amount decisions

This activity only occurs given that other alternatives are more expensive.

Due to the absence of real data of the gas price and gas demand, the data used in the model is generated using a Uniform distribution. The scenario generation for the gas price and gas demand are given in the appendix.

The consideration of deliverability is ignored where it is assumed that gas is easily extracted from storage.

This chapter detailed the two CO problems used in this research. Both problems can be modelled as MBIP problems and the next chapter illustrates how the proposed method is designed to solve the MBIP problems.

Chapter 5.0: The proposed hybrid technique

The idea of the proposed method is to suggest a simple implementation to the computationally hard MBIP problems that are too expensive to solve for exact methods, requiring high computational time and worse, failing to solve due to high memory usage. The method proposed in this research is a hybrid metaheuristic, a collaborative combination of the following:

i. IP method: LP relaxation

ii. Metaheuristics: VNS

The proposed method uses the relaxation of the MBIP at each iteration, providing flexibility to reselect the binary variables that need to be fixed. This technique does not only consider any possibility of discovering potential variables but also keeps the priority of the variables that have high impact on the objective function.

VNS possesses, to a greater extent, all the properties listed in **Section 1.5** and the interest in VNS is rapidly growing with the increasing number of published papers (Hansen et al., 2008). Driven by this fact, VNS is intended to serve as the search agent to explore the search space of the problem at hand.

5.1 LP relaxation in this research

From **(1.7)**, the basic MBIP model is formulated as follows:

$$\textbf{Min } \boldsymbol{f(x,y) = c^T x + d^T y} \qquad (\mathbf{5.0})$$

subject to

$$\boldsymbol{Ax + Gy \leq b} \tag{5.1}$$

$$\boldsymbol{x \geq 0} \tag{5.2}$$

$$\boldsymbol{y \in [0, 1]} \tag{5.3}$$

where $\boldsymbol{f(x, y)}$ is the objective function to be minimised, $\boldsymbol{x}$ is a vector of the continuous decision variables, $\boldsymbol{y}$ is the vector of binary decision variables, $\boldsymbol{A}$ and $\boldsymbol{G}$ are matrices and $\boldsymbol{c^T}$, $\boldsymbol{d^T}$ and $\boldsymbol{b}$ are the constant vectors of coefficients.

In this research, the relaxation of IP is exploited in a sense to:

1. gather information for the initial solution construction.
2. to guide search in promising directions of the search area.

The important idea is to reduce the computational time by preventing searching in the areas that are not producing good solutions. In CO problems studied, the decision variable $\boldsymbol{y_i}$ is the binary variable that will either take the value 0 or 1. By allowing a weaker condition, this binary decision variable $\boldsymbol{y_i}$ can take values ranging from 0 to 1. The relaxed LP problem is then solved to optimality using the CPLEX solver. The information gained from the optimal solutions of the LP relaxation is then observed. This information will not only be used in generating the partial solution (1) but also to allow early pruning of the sub-optimal solutions to the studied CO problem (2).

The selection on the partial fixing of variable $\boldsymbol{y_i}$ is made based on the observation of the optimal solutions of the decision variable that have a direct impact towards the objective function $\boldsymbol{f}$. We denote $\boldsymbol{q_i}$ to be the observed continuous decision variable of interest. It is important to note that there must be a link between the binary variable $\boldsymbol{y_i}$ and the chosen decision variable $\boldsymbol{q_i}$ that is being observed. The higher the value of $\boldsymbol{q_i}$ indicates that the object $\boldsymbol{i}$ has more impact towards the objective value $\boldsymbol{f}$.

For the CITP problem, $\boldsymbol{q_{i=}x_i}$ since the portion of asset $\boldsymbol{i}$ does impact the return on the portfolio. While for the gas supply problem, $\boldsymbol{q_{i=}y_i}$, the binary variable of opening the facility at location $\boldsymbol{l}$. The cost of opening the facility does impacts the total cost.

5.1.1 Defining the neighbourhood

The first step is to construct the partial solution using a similar approach to variable fixing, discussed by Atamtürk and Savelsbergh (2005) in "*Integer-programming software system*". In the MBIP problems considered, the binary variable $\boldsymbol{y_i}$ is used to answer the yes/no question; whether to include an asset $\boldsymbol{i}$ in the portfolio for the CITP problem or to install a facility at a potential location $\boldsymbol{l}$ for the gas supply problem. The number of the binary variables $\boldsymbol{y_i}$ to be chosen i.e. to be fixed to 1, is constrained to be $\boldsymbol{k}$ out of $\boldsymbol{n}$ object $\boldsymbol{i}$.

To generate the initial solutions, we assign $\boldsymbol{\alpha k}$ fixed values of the decision variable $\boldsymbol{y_i}$ where $\boldsymbol{\alpha}$ is the percentage of the total $\boldsymbol{k}$ objects to be chosen. The remaining $\boldsymbol{(1-\alpha)k}$ decision variables $\boldsymbol{y_i}$ are left unfixed and will be optimised in the subsequent stage, along with the remaining $\boldsymbol{(n-k)y_i}$. We denote $\boldsymbol{F}$ to be the set of fixed $\boldsymbol{\alpha k}$ of the variable $\boldsymbol{y_i}$.

Applying the relaxation to decision variable $\boldsymbol{y_i}$

$$\boldsymbol{0 \leq y_i \leq 1 \quad \forall i} \qquad (\boldsymbol{5.4})$$

gives the relaxed LP problem to the original problem. Solving exactly using the CPLEX solver will yield the information on the optimal solutions $\boldsymbol{q_i^*}$ of the decision variable $\boldsymbol{q_i}$.

Steps to generate the initial solutions:

1. Apply integrality relaxations on the binary decision $\boldsymbol{y_i}$ and solve the related LP problem. The optimal solutions of $\boldsymbol{q_i^*}$ are analysed as mentioned earlier.
2. The binary variables $\boldsymbol{y}$ are then sorted in a descending manner with respect to the corresponding relaxed variable $\boldsymbol{q_i^*}$. We denote this ordered set of object $\boldsymbol{i}$ as $\boldsymbol{Q}$.
3. Select the $\boldsymbol{\alpha k}$ highest $\boldsymbol{y_i}$ and fixed $\boldsymbol{y_i = 1}$ leaving the others unfixed. We denote $\boldsymbol{F}$ as the set of the variables to be fixed to 1.
4. The integrality is then forced back on the decision variable $\boldsymbol{y_i}$.
5. The revised problem $\boldsymbol{P'}$ is optimised using the CPLEX solver.

5.2 VNS in this research

VNS offers great potential as a search technique to explore the search space in finding better solutions. By dynamically changing the neighbourhood, we reselect the partial fixing. One important characteristic is *shaking*. The initial solution generated will be used as the starting point and the information gathered from the LP relaxation will be used to guide the search at this stage.

From the obtained revised problem $\boldsymbol{P'}$, we select $\boldsymbol{(1-\alpha)k}$ of the unfixed $\boldsymbol{y_i}$ variables and fix them to 1. This is the set of the binary variable $\boldsymbol{y_i}$ that is found by the solver. We then fix this $\boldsymbol{y_i = 1}$ and optimise the $\boldsymbol{P'}$. The newly obtained problem is denoted as $\boldsymbol{P''}$.

Steps to improve the solutions:

1. From $\boldsymbol{P'}$, set $\boldsymbol{F} := \{\boldsymbol{i} \in \boldsymbol{I} - \boldsymbol{F} | \boldsymbol{y_i = 1}\}$ and fix these values. Let the remaining variables be unfixed.
2. Optimise the problem using the solver to obtained $\boldsymbol{P''}$.

5.2.1 Searching the neighbourhood

There are two stopping criterion used in the search process;

1. Time limit/CPU time ($\boldsymbol{t_{max}}$) and solution limit

By definition (AMPL webpage), the solution limit is the limit on the feasible solutions found before the search terminates and if there is no specification made on the solution limit, the optimiser will search for an optimal solution. Time limit is the CPU time spent before terminating the search and by default there is no time limit imposed. Balancing these two elements is crucial. Although we want to avoid spending too much time on solving the problem (intensification) but at the same time we have to make sure that a diversified search is conducted (have enough feasible solutions).

2. The upper bound of the objective function $\boldsymbol{f}$

Each time the problem is solved, the upper bound of the objective function will be updated to be equal to the value of the objective function of the current incumbent solution. The search process will continue if the objective value of the current solution is smaller than the upper bound otherwise it terminates once the current solution yields a higher objective function value.

5.3 Enhancing the model performance

5.3.1 Intensified shaking

One function that enables the search process to escape from being trapped in the local minimum is *shaking*. We use $\boldsymbol{\beta}$, the combination of several values of the fixing percentage.

Neighbourhoods shaking: At this point, we set $\boldsymbol{\alpha} = \boldsymbol{\beta}$. At each iteration, the LP relaxation is performed and a new $\boldsymbol{\beta k}$ of binary variables that need to be fixed to 1 is selected. However, any binary solution previously determined by the solver with the value of 1 is kept and added to the new set of the partial solution fixing.

We set $\boldsymbol{\beta} = \{\mathbf{0.75}, \mathbf{0.8}, \mathbf{0.95}\}$. At the first iteration the $\boldsymbol{\alpha}$ value will be replaced by $\boldsymbol{\beta} = \mathbf{0.75}$ and for the second iteration $\boldsymbol{\beta} = \mathbf{0.8}$. For iteration 3 onwards, the value of $\boldsymbol{\beta} = \mathbf{0.95}$.

5.3.2 Bound tightening

LP relaxation provides information of $\boldsymbol{l_i^*}$ and $\boldsymbol{u_i^*}$ of the variable $\boldsymbol{q_i^*}$. Using the information provided, we wish to tighten the bounds value on the variable $\boldsymbol{q_i}$ by determining the new values on $\boldsymbol{u_i}$. Bound tightening in this research aims not only to improve the objective function, but also to reduce the computational time.

5.4 The general framework

Figure 5.1 shows the steps of the proposed approach and **Figure 5.2** illustrates the general framework of the proposed approach.

Function (t_{max}, q_i^*, $\alpha = 0.5$, $\beta = 0.75, 0.8, 0.95$)

1 Solve LP

If CITP

Let $q = x$

Else

Let $q = y$

End if

2 Let $Q := \mathbf{sorted}(q^*)$

3 Let $F := \{\}$

4 **For** $\{i \in Q || F| \leq f$ **and** $q_i^* > 0\}$ /*apply VNS to improve the solution*/

Let $F := F \cup \{ i \}$

End for

5 Solve MIP

6 While $t < t_{max}$ **do** /*enhancing the solution with intensified shaking*/

7 **For** $\{j$ **in** $1..3\}$

If CITP

Let $q = x$

Else

Let $q = y$

End if

If $j = 1$

Let $f = 0.75$

Else if $j = 2$

Let $f = 0.8$

Else

Let $f = 0.9$

End if

Figure 5.1: The algorithms of the proposed approach

8 Let $\boldsymbol{Q} := \mathbf{sorted}(\boldsymbol{q}^*)$

9 Let $\boldsymbol{F} := \{\boldsymbol{i} \in \boldsymbol{I} - \boldsymbol{F} | \boldsymbol{y_i} = \mathbf{1}\}$

10 **For** $\{\boldsymbol{i} \in \boldsymbol{Q} || \boldsymbol{F}| \leq \boldsymbol{f}$ **and** $\boldsymbol{q_i} > \mathbf{0}\}$

Let $\boldsymbol{F} := \boldsymbol{F} \cup \{\mathbf{i}\}$

End for

11 Solve MIP

End for

12 **if** $\boldsymbol{f} < \boldsymbol{f}^*$ **then**

13 $\boldsymbol{f}^* \leftarrow \boldsymbol{f}$, go to line 7

until $\boldsymbol{f} > \boldsymbol{f}^*$

14 **End while**

Figure 5.1: The algorithms of the proposed approach (cont)

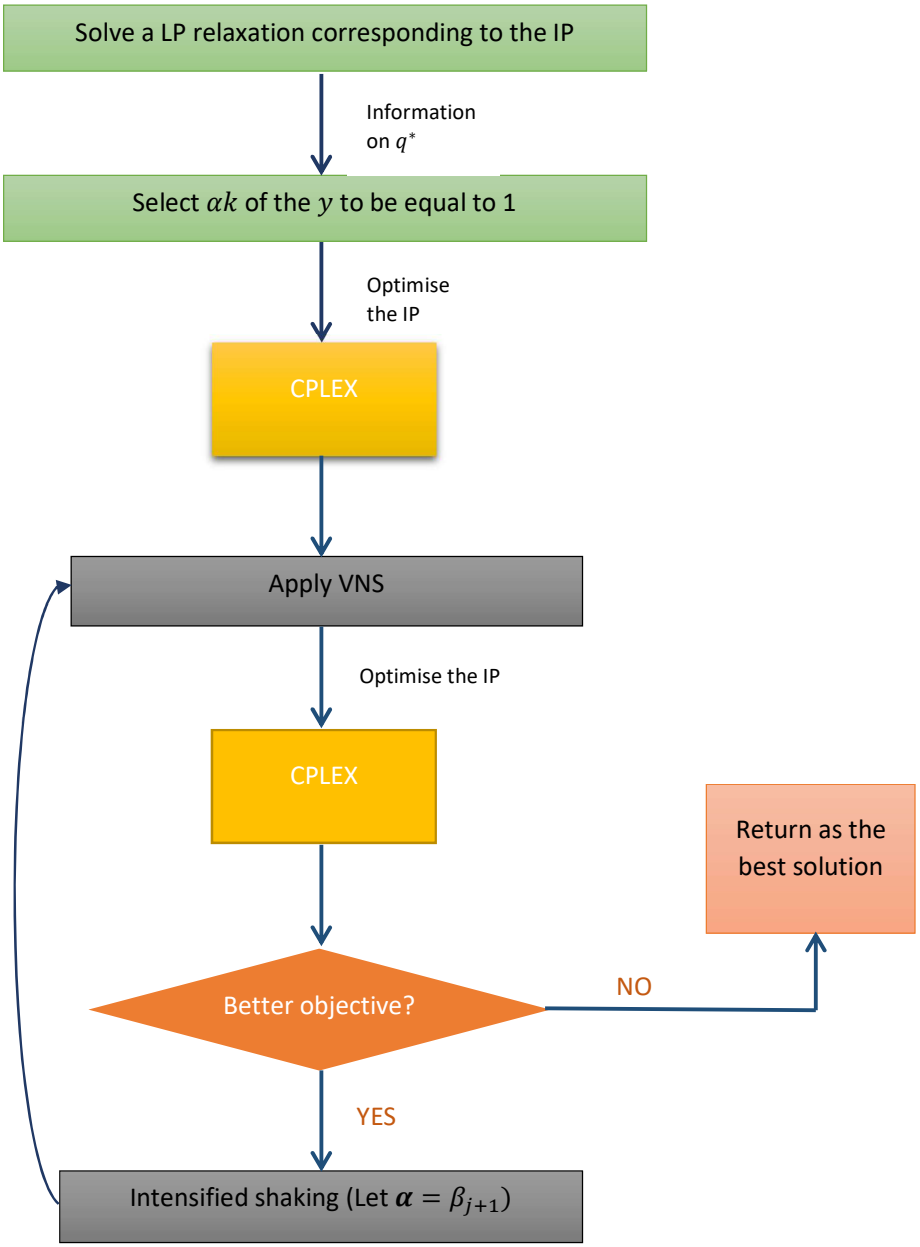

Figure 5.2: The general framework of the proposed approach

5.5 Hybrid approach to CITP

In this section, we give details of the proposed hybrid approach and the modifications made to solve the CITP.

5.5.1 Partial fixing of y_i

For the CITP problem, the decision variable that was observed is the weight of the assets held, $\boldsymbol{x_i}$ in selecting the binary variable $\boldsymbol{y_i}$ for the asset $\boldsymbol{i}$ to be included in the new portfolio. This is because the decision variable $\boldsymbol{x_i}$ influences the objective function, **TE** (since the amount of the asset $\boldsymbol{i}$ held contributes to the portfolio's return).

We use $\boldsymbol{\alpha}$ to partially fix the binary variables $\boldsymbol{y_i}$. The value of $\boldsymbol{\alpha}$ is defined by the user, ranging from 0 to 1. The impact of this value will be closely observed by testing several initial $\boldsymbol{\alpha}$ values (**0.25**, **0.5**, and **0.75**) to determine the value that yields the minimum value of the objective function **TE**.

5.5.2 Stopping criteria

The stopping criteria that used for the search process are:

1. The maximum time allowed for the search process $\boldsymbol{t_{max}}$
2. The upper bound of the tracking error, **TE**.

5.5.3 Enhancing the model performance

To improve the solutions obtained, the following parameters will be experimented with:

1. The upper bound $\boldsymbol{u_i}$ of the variable $\boldsymbol{x_i}$.

   We performed bound tightening on $\boldsymbol{u_i}$ to enhance the performance of the proposed technique. By setting different values of the upper threshold $\boldsymbol{u_i}$, the performance of the model is analysed to determine the best value of $\boldsymbol{u_i}$ that yields the best **TE**.

2. Intensified shaking

   At this stage, we replace the $\boldsymbol{\alpha}$ value with the $\boldsymbol{\beta}$. We allow a higher percentage of the partial fixing of the binary variable $\boldsymbol{y_i}$.

5.5.4 Variants of shaking

Intensified shaking, **Step 10** in **Figure 5.1** was performed by forcing some assets into the partial solution. These assets were picked based on their weight in the related LP relaxed solution. We developed two ways of performing the intensified shaking procedure in this proposed hybrid technique. The following subsections explain how the shaking procedure works under each variant.

5.5.4.1 Shaking variant 1

From **Section 5.1.1**, we denote $\boldsymbol{Q} := \boldsymbol{sorted}(\boldsymbol{x}^*)$ where $\boldsymbol{x}^*$ is the related LP relaxed value of $\boldsymbol{x}$ and from **Section 5.2**, we denote $\boldsymbol{I} \coloneqq$ **assets** $\boldsymbol{i}$ **that were chosen by the solver**. This set $\boldsymbol{I}$ is updated each time the solver solves the MBIP (**Step 5** and **Step 11** in **Figure 5.1**).

The following **Figure 5.3** exhibits how the modification of the general algorithm in **Figure 5.1** was done under **Shaking Variant 1** in order to solve the CITP.

8 Let $\boldsymbol{Q} := \mathbf{sorted}(\boldsymbol{x_i^*})$

9 Let $\boldsymbol{F} := \{\boldsymbol{i} \in \boldsymbol{I} - \boldsymbol{F} | \boldsymbol{y_i} = \mathbf{1}\}$

10 **For** $\{\boldsymbol{i} \in \boldsymbol{Q} || \boldsymbol{F}| \leq \boldsymbol{f}$ **and** $\boldsymbol{x_i^*} > \mathbf{0}\}$

Let $\boldsymbol{F} := \boldsymbol{F} \cup \{\mathbf{i}\}$

End for

11 Solve MIP

End for

Figure 5.3: Shaking Variant 1

5.5.4.2 Shaking variant 2

The difference between **Shaking Variant 1** and **Shaking Variant 2** is that in **Shaking Variant 2**, the assets were reordered each time the problem was solved. By fixing the assets in set $\boldsymbol{I}$ equal to 1, the integrality of the binary assets is relaxed and after the problem is solved, a new set $\boldsymbol{Q}$ is obtained. The reselection of assets on the next run is based on this set $\boldsymbol{Q}$.

The following **Figure 5.4** shows how the shaking process works under **Shaking Variant 2**.

8 Let $\boldsymbol{Q} := \mathbf{sorted}(\boldsymbol{x}_i^*)$

9 Let $\boldsymbol{F} := \{\boldsymbol{i} \in \boldsymbol{I} - \boldsymbol{F} | \boldsymbol{y}_i = \mathbf{1}\}$

10 **For** $\{\boldsymbol{i} \in \boldsymbol{Q} || \boldsymbol{F}| \leq \boldsymbol{f} \textbf{ and } \boldsymbol{x}_i^* > \mathbf{0}\}$

Let $\boldsymbol{F} := \boldsymbol{F} \cup \{\boldsymbol{i}\}$

End for

11 Solve the MIP

End for

12 Let $\boldsymbol{Q} := \{\}$

Fix $\boldsymbol{y}_i = \mathbf{1}$ for $\boldsymbol{i} \in \boldsymbol{F}$

Allow weaker integrality on $\boldsymbol{y}_i$ for $\boldsymbol{i} \notin \boldsymbol{F}$

Solve MIP

Let $\boldsymbol{Q} := \mathbf{sorted}(\boldsymbol{x}^*)$ for $\boldsymbol{i} \in (\boldsymbol{I} - \boldsymbol{F})$

Force the integrality back into the problem

Go to line 10

13 **if** $\boldsymbol{f} < \boldsymbol{f}^*$ **then**

14 $\boldsymbol{f}^* \leftarrow \boldsymbol{f}$, go to line 10

until $\boldsymbol{f} > \boldsymbol{f}^*$

15 **End while**

Figure 5.4: Shaking Variant 2

5.6 Hybrid approach to the gas supply problem

5.6.1 Initial partial fixing of y_l

In the gas supply problem, the binary decision variable is the decision on opening the storage facility at location $\boldsymbol{l}$, denoted by $\boldsymbol{y_l}$. Since the cost to open the facility is included in the objective function, the binary variable $\boldsymbol{y_l}$ does impact the objective function $\boldsymbol{f}$.

To partially fix the binary variables $\boldsymbol{y_l}$ to 1, we use the information on the relaxed solutions of the variable $\boldsymbol{y_l^*}$. Since the value of $\boldsymbol{k}$ is not in the formulation, we set the $\boldsymbol{k}$ value to be:

$$\boldsymbol{k} = \sum_{l} \boldsymbol{y_l^*} \qquad (\mathbf{5.5})$$

Partial fixing of $\boldsymbol{y_l}$ is done similar to the partial fixing of $\boldsymbol{y_i}$ in the CITP problem, using $\boldsymbol{\alpha}$ values (**0.25**, **0.5**, and **0.75**) to determine the value that yields the minimum value of the total cost.

5.6.2 Stopping criteria

The stopping criteria:

1. The maximum time allowed for the search process $\boldsymbol{t_{max}}$
2. The upper bound of the cost function $\boldsymbol{f}$.

5.6.3 Enhancing the model performance

The parameters that will be used to enhance the model performance:

1. The upper value $\boldsymbol{m_l}$ of the maximum storage $\boldsymbol{M}$.

   We performed bound tightening on $\boldsymbol{m_l}$ to enhance the performance of the proposed technique. With the information obtained on $\boldsymbol{M}^*$, we analyse the value of $\boldsymbol{M}$ that will improve the cost function $\boldsymbol{f}$.

2. Intensified shaking

   At this stage, we replace the $\boldsymbol{\alpha}$ value with the $\boldsymbol{\beta}$. We allow a higher percentage of the partial fixing of the binary variable $\boldsymbol{z_l}$.

5.6.4 Variants of initial partial fixing

In **Section 5.6.1**, we discussed how the initial partial fixing is done. In this section, we explain an alternative way to perform the initial partial fixing of the binary variables. Instead of fixing the binary variables to be 1, we now choose which binary variables that will be fixed to 0. We determine the value of $\boldsymbol{k}$ the same way we did in the partial fixing explained in **Section 5.6.1** and fix the lowest $\boldsymbol{k-1}$ of $\boldsymbol{y_l^*}$ to 0. In the CITP, we have two different variants of shaking. In the gas supply chain problem, we have two ways of the initial partial fixing.

For both variants, the partial fixing for the shaking process was done similar to the CITP, where binary variables with higher relaxed value of $\boldsymbol{y_l^*}$ are forced into the partial solutions.

Chapter 6.0: Experimental results

This chapter provides a comprehensive analysis of the proposed technique's performance. One of the most commonly adopted approaches to measure the performance of a proposed technique is to compare the results obtained by the proposed technique with the results available in the literature. However, due to several factors, it is not possible to make a fair comparison of the results generated by the proposed techniques with the results available in the literature. For example, the capacity of the machine used may not be the same or no information on the machine's capacity, and the approach taken to solve the studied problem is different.

6.1 Computer support

The code was written using AMPL running the CPLEX solver (version 12.6.0). The program was run on an Intel® Core ™ i5-2500 at 3.30 GHz with 16 GB of RAM.

6.2 Constrained index tracking problem (CITP)

6.2.1 Data

To test the performance of the proposed method, we used the data set taken from the Beasley OR Library. There were 6 sets of data and for each data set, there were 3 instances tested, making a total of 18 instances used to test the efficiency and effectiveness of the proposed technique. The following **Table 6.0** provides the information of each of the data sets, where the readers can gain some overview.

Data set	Total number of assets	Non-zero assets[9]	Instances (cardinality)
Indtrack3	89	81	30
			35
			40
Indtrack4	98	83	30
			35
			40
Indtrack5	225	159	50
			75
			100
Indtrack6	457	144	50
			75
			100
Indtrack7	1418	229	50
			75
			100
Indtrack8	2151	234	50
			75
			100

Table 6.0: Information of each data set for CITP

[9] These are the assets $\boldsymbol{i}$ with positive weights $\boldsymbol{x_i^* > 0}$ in the relaxed LP.

From the table, we can see that the number of the cardinality picked is about 30 to 50 percent from the total number of non-zero assets, with the exception for 5 instances where the percentage does not fall into this range. We based the cardinality percentage on the non-zero assets because the intelligence that was introduced in the proposed technique was based on the non-zero assets, where we forced some of the assets based on their weight in the related relaxed solution.

Therefore, for ***indtrack8***, if we were to take 30 percent from the total number of assets, about 537 assets should be picked. This is not possible since the number of non-zero assets for ***indtrack8*** is 234.

6.2.2 Empirical results

After running several tests, it is concluded that the best combination of the parameters values that yields the best results were found as follows:

1) Solution limit is 35 for the first run and 25 for the subsequent runs.
2) Time limit is best set at 300 seconds for the first run and 150 seconds for the subsequent runs.
3) The $\boldsymbol{\alpha}$ value is 0.5.
4) The combination of the $\boldsymbol{\beta}$ (intensified shaking) values are 0.75, 0.8, 0.9.
5) The upper value $\boldsymbol{u_i}$ of the variable $\boldsymbol{x_i}$ is $\boldsymbol{u_i = 0.015}$ for $\boldsymbol{x_i^* = 0}$ and $\boldsymbol{u_i = \max(4.3x_i^*, 0.015)}$ for $\boldsymbol{x_i^* > 0}$

Due to the limitations imposed on both the solution limit and time limit, different results were generated each time we solved the problem. Therefore, for each instances, we ran 5 times and the following **Table 6.1** and **Table 6.2** give the best found, mean and mode of the solutions that were generated by the proposed method using **Shaking Variant 1** and **Variant 2**. The full results are provided in **Appendix B** of this thesis.

The mode of the solution were the value of **TE** that appeared more than once when we solved the problem for 5 different times. The mode of the solution gives the likely value of the **TE** that we will obtain when we solved the problem. Best found is the lowest **TE** generated among the 5 times we solved the problem.

Figure 6.0, **Figure 6.1** and **Figure 6.3** illustrate the number of wins and draws (both shaking procedure give the same **TE**) of best found, mean and mode of the solutions between **Shaking Variant 1** and **Shaking Variant 2**. From this point onwards, we denote the proposed hybrid technique using **Shaking Variant 1** and **Shaking Variant 2** as **Hybrid Variant 1** and **Hybrid Variant 2** respectively.

	total assets[10]	Cardinality	Hybrid Variant 1					
			Best		Mean		Mode	
			TE	seconds	TE	seconds	TE	seconds
indtrack3	89	30	0.489247450	415.801	0.489247450	416.133	0.489247450	416.133
		35	0.429655449	764.113	0.429655449	764.123	0.429655449	764.123
		40	0.378027573	744.800	0.378027573	749.130	0.378027573	749.130
indtrack4	98	30	0.440715907	323.701	0.443689184	316.274	0.444432504	321.032
		35	0.375229507	320.572	0.383282648	328.420	0.386013452	330.706
		40	0.344979505	370.247	0.344979505	370.325	0.344979505	370.325
indtrack5	225	50	0.319647027	1134.432	0.322500563	1084.840	0.319647027	1140.867
		75	0.230784396	1371.642	0.230784396	1380.915	0.230784396	1380.915
		100	0.194837657	1055.679	0.195204407	1107.544	0.195187017	1193.793
indtrack6	457	50	0.550234487	1087.177	0.550234487	1088.077	0.550234487	1088.077
		75	0.449535004	998.186	0.449535004	998.101	0.449535004	998.101
		100	0.422325618	840.514	0.422325618	842.361	0.422325618	842.361
indtrack7	1418	50	0.761432001	1503.114	0.768103691	1341.828	0.769771614	1301.506
		75	0.492734947	1144.066	0.508353423	1030.434	0.505182551	901.909
		100	0.359056561	1333.960	0.372651092	1001.381	0.376049725	918.237
indtrack8	2151	50	0.521914474	1484.288	0.545190203	1019.532	0.551009136	903.343
		75	0.365493892	903.214	0.365493892	903.296	0.365493892	903.296
		100	0.267110930	1587.021	0.267110930	1749.182	0.267110930	1749.182

Table 6.1: The best found, mean and mode of the solutions generated by **Hybrid Variant 1**

[10] The total number of assets available in the portfolio

	total assets	Cardinality	Hybrid Variant 2					
			Best		Mean		Mode	
			TE	seconds	TE	seconds	TE	seconds
indtrack3	89	30	0.48924745	600.982	0.48924745	601.7504	0.48924745	601.7504
		35	0.431925095	715.127	0.431925095	715.38	0.431925095	715.38
		40	0.380424531	661.247	0.380424531	667.7252	0.380424531	667.7252
indtrack4	98	30	0.43270049	528.986	0.43270049	533.1558	0.43270049	533.1558
		35	0.375229507	613.476	0.382855382	632.9588	0.38476185	637.8295
		40	0.344979505	670.527	0.344979505	670.6842	0.344979505	670.6842
indtrack5	225	50	0.327314345	900.703	0.326234847	907.393	0.327314345	900.76
		75	0.231418232	1078.839	0.231418232	1080.1916	0.231418232	1080.1916
		100	0.191327665	1843.689	0.194112834	1169.1264	Do not exist	
indtrack6	457	50	0.551165494	886.689	0.551165494	887.5808	0.551165494	887.5808
		75	0.449535004	1006.345	0.449535004	1009.0958	0.449535004	1009.0958
		100	0.422325618	826.364	0.422325618	838.773	0.422325618	838.773
indtrack7	1418	50	0.725226933	1582.103	0.771262735	1118.1138	0.786002718	901.937
		75	0.505182551	901.939	0.50964396	901.9404	0.505182551	901.97733
		100	0.381857299	1361.567	0.400532569	1074.9484	0.406079358	903.854
							0.404323415	1102.7335
indtrack8	2151	50	0.52586196	1502.932	0.526898491	1344.2708	0.527157624	1304.6055
		75	0.365493892	903.595	0.365493892	903.6934	0.365493892	903.6934
		100	0.295978968	903.412	0.295978968	903.5018	0.295978968	903.5018

Table 6.2: The best found, mean and mode of the solutions generated by **Hybrid Variant 2**

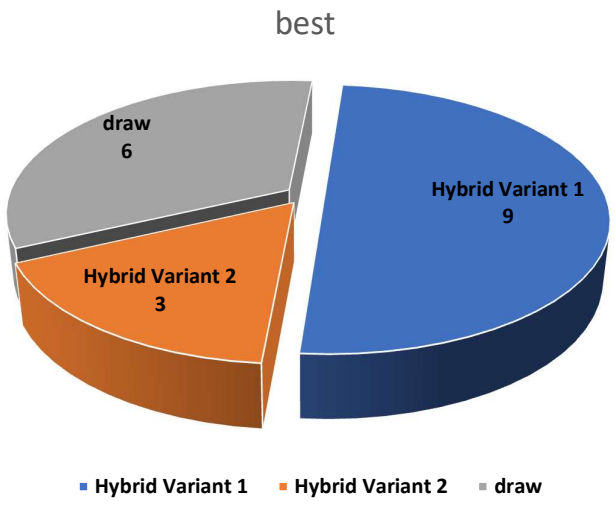

Figure 6.0: Number of wins and draws (best found) between **Hybrid Variant 1** and **Hybrid Variant 2**

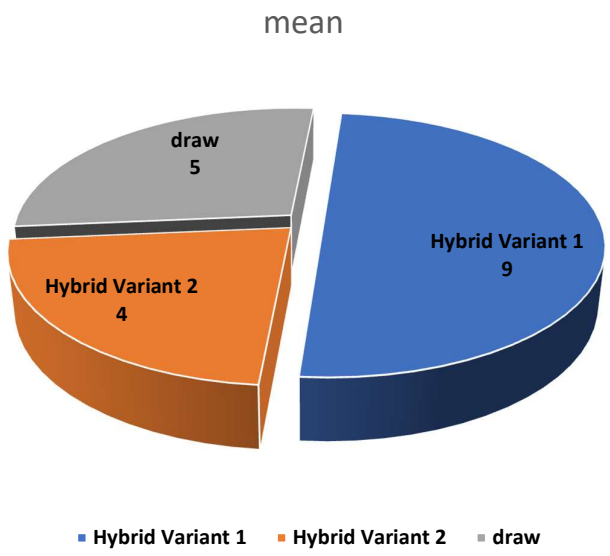

Figure 6.1: Number of wins and draws (mean) between **Hybrid Variant 1** and **Hybrid Variant 2**

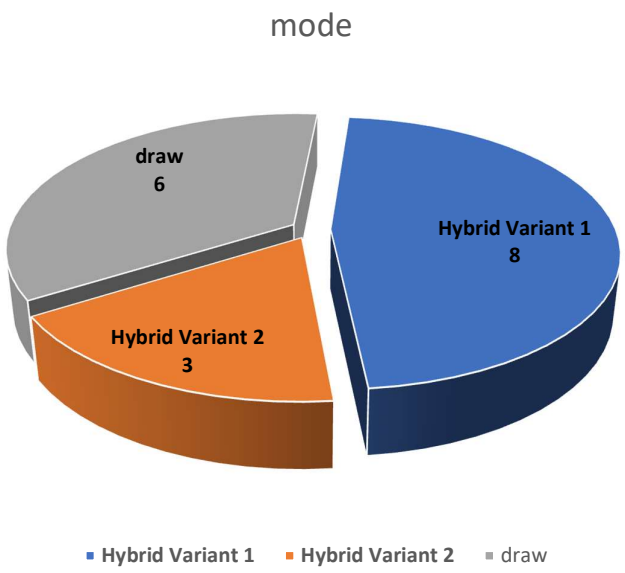

Figure 6.2: Number of wins and draws (mode) between **Hybrid Variant 1** and **Hybrid Variant 2**

Comparing the results obtained by both **Hybrid Variant 1** and **Hybrid Variant 2**, **Hybrid Variant 1** has the most number of wins in all of the three comparisons. On average (mean), **Hybrid Variant 1** outperformed **Hybrid Variant 2** in 9 instances (out of 18 instances) and obtained the same result for 5 instances.

The following **Table 6.3** compares the performance of the mean solutions of **Hybrid Variant 2** to **Hybrid Variant 1**. We calculated the relative error and the changes in the computational time using the following formulations:

$$\textbf{relative error} = \frac{TE_{V2} - TE_{V1}}{TE_{V1}} \qquad (6.0)$$

$$\textbf{changes in time} = \frac{seconds_{V2} - seconds_{V1}}{seconds_{V1}} \qquad (6.1)$$

TE_{V1} and TE_{V2} are the **TE** generated using **Hybrid Variant 1** and **Hybrid Variant 2** respectively. $seconds_{V1}$ is the computational time recorded by **Hybrid Variant 1** and $seconds_{V2}$ is the computational time recorded by **Hybrid Variant 2**.

	total assets	cardinality	Mean $\frac{TE_{V2}-TE_{V1}}{TE_{V1}}$ (%)	Mean $\frac{seconds_{V2}-seconds_{V1}}{seconds_{V1}}$ (%)
indtrack3	89	30	0.00	44.61
		35	0.53	(6.38)
		40	0.63	(10.87)
indtrack4	98	30	(2.48)	68.57
		35	(0.11)	92.73
		40	0.00	81.11
indtrack5	225	50	1.16	(16.36)
		75	0.27	(21.78)
		100	(0.56)	5.56%
indtrack6	457	50	0.17	(18.43)
		75	0.00	1.10
		100	0.00	(0.43)
indtrack7	1418	50	0.41	(16.67)
		75	0.25	(12.47)
		100	7.48	7.35
indtrack8	2151	50	(3.36)	31.85
		75	0.00	0.04
		100	10.81	(48.35)

Table 6.3: Relative error of **Hybrid Variant 2** to **Hybrid Variant 1**

From **Table 6.3**, we can see that, in the case where both **Hybrid Variant 1** and **Hybrid Variant 2** generated the same **TE**, **Hybrid Variant 2** required a significantly higher running time compared to **Hybrid Variant 1** for 2 out of the 5 instances. In the remaining 3 instances, the difference of the running time is relatively small and in 1 of the instances, the reduction in the running time in **Hybrid Variant 1** is only 0.43 percent.

We concluded that **Hybrid Variant 1** performed better than **Hybrid Variant 1**. The shaking under **Hybrid Variant 2** may have redirected the search farther than the neighbourhood with good solutions, hence generating less desirable solutions.

6.2.3 Comparison of the proposed hybrid technique to other methods

To evaluate the performance of the proposed technique, the results obtained are compared to:

1) an exact method (CPLEX) with some limitations imposed
2) a heuristic technique

6.2.3.1 Hybrid VS exact method (brute force)

We first compare the generated results with the results produced using the exact method. One exact method available is the CPLEX solver. However, the CPLEX solver used in the experiments was unable to solve the problem to optimality due to the massive computational efforts required. Therefore, the brute force method with the following limitations were imposed:

1) Solution limit: The solution limit in the first run was set to 20 solutions and this limit was increased by 10 solutions in each subsequent run whenever the current solutions were better than the previous solution and the programme terminated when there was no improvement in the current solution.
2) Time limit: The time limit was fixed at 300 seconds for each run.

In the first run, the solution limit was 20 and the time limit was 300 seconds. Now, in the second run, the solution limit is 30 and the time limit is still 300 seconds. If there is an improvement in the solution generated, the progamme will continue to run for the third time with solution limit 40 and time limit 300 seconds. **Table 6.4** gives the best found, mean and mode of the solution generated by the CPLEX solver. The running time given in the table is the accumulated time for all of the runs.

	total assets	cardinality	Solver					
			Best		Mean		Mode	
			TE	seconds	TE	seconds	TE	seconds
indtrack3	89	30	0.481754067	679.126	0.481754067	679.8148	0.481754067	679.8148
		35	0.427148057	608.289	0.428688435	904.2884	0.427148057	754.1335
		40	0.378027573	655.099	0.37994514	654.7428	0.380424531	654.6538
indtrack4	98	30	0.429386982	639.348	0.433192267	639.4724	0.433717321	639.236
		35	0.387476607	658.922	0.387476607	659.695	0.387476607	659.695
		40	0.351930283	1248.879	0.352012285	888.5724	0.352066953	648.2723
indtrack5	225	50	0.336470332	1099.05	0.337833567	1099.4732	0.336470332	1099.311
		75	0.233221442	1636.469	0.237321315	1276.2558	0.239218337	1036.001
		100	0.192830675	1924.459	0.194152056	1143.641	0.194960159	873.334
indtrack6	457	50	0.561663663	1740.073	0.561663663	1740.5826	0.561663663	1740.583
		75	0.447960169	2291.379	0.449802306	1571.0194	0.45026284	1390.93
		100	0.424167101	480.108	0.424167101	480.4504	0.424167101	480.4504
indtrack7	1418	50	1.054647903	900.717	1.054647903	900.7666	1.054647903	900.7666
		75	0.686114743	1200.932	0.735953341	960.7924	0.748412991	900.7575
		100	0.420452453	900.51	0.420452453	900.746	0.420452453	900.746
indtrack8	2151	50	0.731449872	900.847	0.731449872	900.8882	0.731449872	900.8882
		75	0.428562345	600.695	0.428562345	600.842	0.428562345	600.842
		100	0.308858348	600.582	0.308858348	600.6108	0.308858348	600.6108

Table 6.4: Best found, mean and mode of the **TE** generated by the CPLEX solver

Figure 6.3, **Figure 6.4** and **Figure 6.5** illustrate the number of wins of best found, mean and mode of the solutions between **Hybrid Variant 1**, **Hybrid Variant 2** and the CPLEX solver.

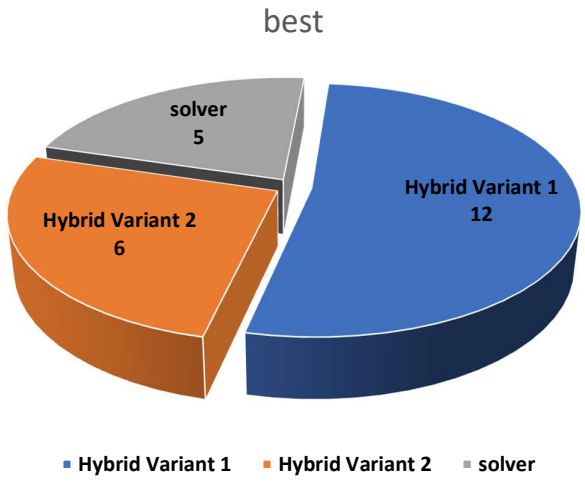

Figure 6.3: Number of wins and draws (best found) between **Hybrid Variant 1**, **Hybrid Variant 2** and CPLEX solver

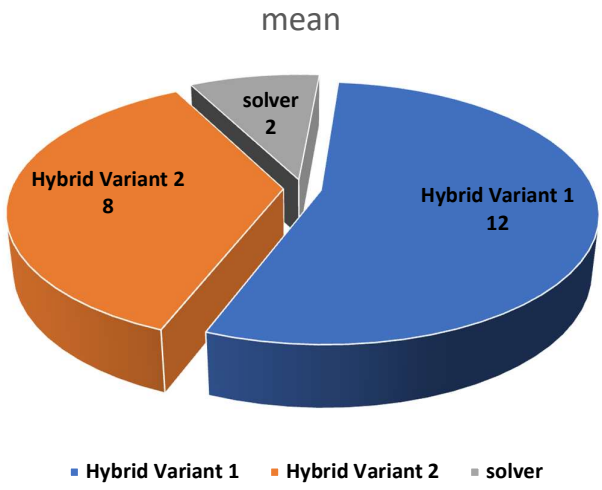

Figure 6.4: Number of wins and draws (mean) between **Hybrid Variant 1**, **Hybrid Variant 2** and CPLEX solver

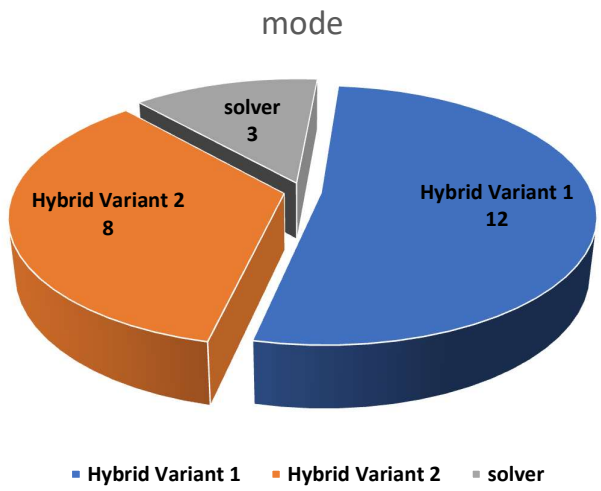

Figure 6.5: Number of wins and draws (mode) between **Hybrid Variant 1**, **Hybrid Variant 2** and solver

No draws in the **TE** were recorded between the proposed hybrid technique (both variants) and the CPLEX solver. There was only 1 instance (***indtrack3*** with cardinality 40) where **Hybrid Variant 1** generated the same with the CPLEX solver under the best found solutions comparison.

We can see that **Hybrid Variant 1** outperformed **Hybrid Variant 2** and the CPLEX solver in all of the three comparisons with a significant number of wins. Although **Hybrid Variant 2** did not perform as well as **Hybrid Variant 1**, it still outperformed the results given by the CPLEX solver.

Table 6.5 analysed how well on average (mean) **Hybrid Variant 1** and **Hybrid Variant 2** performed compared to the CPLEX solver while **Table 6.6** shows how the three approaches performed when being compared to the best (mean) solutions among the three. The error and speed were calculated using the formulations given in **(6.0)** and **(6.1)**.

	total assets	cardinality	Hybrid Variant 1		Hybrid Variant 2	
			$\frac{TE_{V1}-TE_{solver}}{TE_{solver}}$ (%)	$\frac{seconds_{V1}-seconds_{solver}}{seconds_{solver}}$ (%)	$\frac{TE_{V2}-TE_{solver}}{TE_{solver}}$ (%)	$\frac{seconds_{V2}-seconds_{solver}}{seconds_{solver}}$ (%)
indtrack3	89	30	1.56	(38.79)	1.56	(11.48)
		35	0.23	(15.50)	0.76	(20.89)
		40	(0.50)	14.42	0.13	1.98
indtrack4	98	30	2.42	(50.54)	(0.11)	(16.63)
		35	(1.08)	(50.22)	(1.19)	(4.05)
		40	(2.00)	(58.32)	(2.00)	(24.52)
indtrack5	225	50	(4.54)	(1.33)	(3.43)	(17.47)
		75	(2.75)	8.20	(2.49)	(15.36)
		100	0.54	(3.16)	(0.02)	2.23
indtrack6	457	50	(2.03)	(37.49)	(1.87)	(49.01)
		75	(0.06)	(36.47)	(0.06)	(35.77)
		100	(0.43)	75.33	(0.43)	74.58
indtrack7	1418	50	(27.17)	48.97	(26.87)	24.13
		75	(30.93)	7.25	(30.75)	(6.13)
		100	(11.37)	11.17	(4.74)	19.34
indtrack8	2151	50	(25.46)	13.17	(27.97)	49.22
		75	(14.72)	50.34	(14.72)	50.40
		100	(13.52)	191.23	(4.17)	50.43

Table 6.5: Relative error of the proposed techniques to the CPLEX solver

	total assets	cardinality	Hybrid Variant 1		Hybrid Variant 2		solver	
			$\frac{TE_{V1}-TE_{best}}{TE_{best}}$ (%)	$\frac{seconds_{V1}-seconds_{best}}{seconds_{best}}$ (%)	$\frac{TE_{V2}-TE_{best}}{TE_{best}}$ (%)	$\frac{seconds_{V2}-seconds_{best}}{seconds_{best}}$ (%)	$\frac{TE_{solver}-TE_{best}}{TE_{best}}$ (%)	$\frac{seconds_{solver}-seconds_{best}}{seconds_{best}}$ (%)
indtrack3	89	30	1.56	(38.79)	1.56	(11.48)	0.00	0.00
		35	0.23	(15.50)	0.76	(20.89)	0.00	0.00
		40	0.00	0.00	0.63	(10.87)	0.51	(12.60)
indtrack4	98	30	2.54	(40.68)	0.00	0.00	0.11	19.94
		35	0.11	(48.11)	0.00	0.00	1.21	4.22
		40	0.00	0.00	0.00	81.11	2.04	139.94
indtrack5	225	50	0.00	0.00	1.16	(16.36)	4.75	1.35
		75	0.00	0.00	0.27	(21.78)	2.83	(7.58)
		100	0.56	(5.27)	0.00	0.00	0.02	(2.18)
indtrack6	457	50	0.00	0.00	0.17	(18.43)	2.08	59.97
		75	0.00	0.00	0.00	1.10	0.06	57.40
		100	0.00	0.43	0.00	0.00	0.44	(42.72)
indtrack7	1418	50	0.00	0.00	0.41	(16.67)	37.31	(32.87)
		75	0.00	0.00	0.25	(12.47)	44.77	(6.76)
		100	0.00	0.00	7.48	7.35	12.83	(10.05)
indtrack8	2151	50	3.47	(24.16)	0.00	0.00	38.82	(32.98)
		75	0.00	0.00	0.00	0.04	17.26	(33.48)
		100	0.00	0.00	10.81	(48.35)	15.63	(65.66)

Table 6.6: Relative error of **Hybrid Variant 1**, **Hybrid Variant 2** and the CPLEX solver compared to the best solutions (mean) found among the three approaches

[11] TE_{best} is the **TE** given by the best solutions obtained among the three approaches

[12] $seconds_{best}$ is the computational time of the best solutions obtained among the three approaches

From **Table 6.5**, we can see that, except for 3 to 4 instances, the **TE** generated using the proposed hybrid technique showed a decrease in the value, especially for larger data sets. For ***indtrack7*** and ***indtrack8***, all of the instances showed a better **TE** compared to the ones generated by the CPLEX solver with a decrease of more than 10 percent for most of the instances. The required running time to achieve these results showed a moderate increase (about 10 to 50 percent) except for 1 instance where the running time recorded an increase of 191.23 percent.

In Table 6.6, the **TE** generated by the solver increased about 12 to 45 percent for all of the instances in both of the larger data sets, ***indtrack7*** and ***indtrack8***. The decrease in the running time did not show an immense change since the largest percentage of decrease recorded was 65.66 percent. Both **Table 6.5** and **Table 6.6** indicate that a better **TE** was achieved using the proposed hybrid technique without requiring a large change of running time.

We then increased the time limit imposed on the solver to 2000 seconds to see if the solver can perform better and provide more competitive results to the results generated by the proposed hybrid technique. **Table 6.7** gives the mean results generated by the CPLEX solver when the time limit is increased and **Table 6.8** compares the average results generated by the CPLEX solver with 2000 seconds of time limit with the one with 300 seconds time limit.

	total assets	cardinality	Solver					
			Best		Mean		Mode	
			TE	seconds	TE	seconds	TE	seconds
indtrack3	89	30	0.481754067	4081.7	0.481754067	4083.0522	0.481754067	4083.0522
		35	0.427148057	4010.429	0.428828461	4372.5212	0.427148057	4677.658
		40	0.378027573	4057.55	0.37994514	4057.2168	0.380424531	4057.1335
indtrack4	98	30	0.429386982	4041.764	0.433842715	4041.9346	do not exist	
		35	0.387476607	4061.598	0.387476607	4063.2372	0.387476607	4063.2372
		40	0.351930283	6052.248	0.351957617	6453.2968	0.351930283	7054.0135
indtrack5	225	50	0.3262242	6037.928	0.329979584	6562.5776	0.330172034	6190.7205
		75	0.230377212	6018.324	0.235501797	5670.2916	do not exist	
		100	0.191397623	6052.919	0.192852076	6423.068	do not exist	
indtrack6	457	50	0.553554599	14250.481	0.561186166	8552.7564	do not exist	
		75	0.448097106	8198.638	0.450452564	7797.4722	do not exist	
		100	0.424167101	2181.703	0.424167101	2182.1914	0.424167101	2182.1914
indtrack7	1418	50	0.788738237	21828.03	0.836067921	14206.7174	do not exist	
		75	0.492241966	31836.885	0.504439822	24204.046	do not exist	
		100	0.362909001	22418.574	0.364239725	22006.1488	do not exist	
indtrack8	2151	50	0.509472133	28016.323	0.53144998	22413.7172	do not exist	
		75	0.325592283	20994.889	0.343899392	11386.815	0.34847617	8984.7965
		100	0.252870911	20587.671	0.259625893	11771.5206	0.258746597 0.2638826793	10568.2815 8566.6845

Table 6.7: Best found, mean and mode of the TE generated using the CPLEX solver with 2000 time limit

	total assets	cardinality	Mean	
			$\frac{TE_{solver2}[13]-TE_{solver}}{TE_{solver}}$ (%)	$\frac{seconds_{solver2}[14]-seconds_{solver}}{seconds_{solver}}$ (%)
indtrack3	89	30	0.00	500.61
		35	0.03	383.53
		40	0.00	519.67
indtrack4	98	30	0.15	532.07
		35	0.00	515.93
		40	(0.02)	626.25
indtrack5	225	50	(2.32)	496.88
		75	(0.77)	344.29
		100	(0.67)	461.63
indtrack6	457	50	(0.09)	391.37
		75	0.14	396.33
		100	0.00	354.20
indtrack7	1418	50	(20.73)	1477.18
		75	(31.46)	2419.18
		100	(13.37)	2343.10
indtrack8	2151	50	(27.34)	2387.96
		75	(19.76)	1795.14
		100	(15.94)	1859.92

Table 6.8: Relative error of the **TE** (mean) using the CPLEX solver with 2000 time limit to the **TE** generated using the CPLEX solver with 300 time limit

The results generated by the CPLEX solver with the time limit of 2000 seconds showed a decrease in the **TE** for most of the instances and performed significantly better in the large data set, ***indtrack7*** and ***indtrack8***, showing a reduction of about 13 to 32 percent in the **TE** for all of the 6 instances. However, an enormous increase in the running time, about 1400 to 2500 times than the time of the CPLEX solver with 300 seconds time limit is required to obtain these significantly better **TE**.

Because the mode for most of the instances did not exist, we compare only the best found solutions and the solutions mean of the CPLEX solver with 2000 seconds time limit with the results generated by the proposed hybrid technique using **Hybrid Variant 1** and **Hybrid Variant 2**.

[13] $TE_{solver2}$ is the TE obtained using the CPLEX solver with 2000 time limit
[14] $seconds_{solver2}$ is the computational time using the CPLEX solver with 2000 time limit

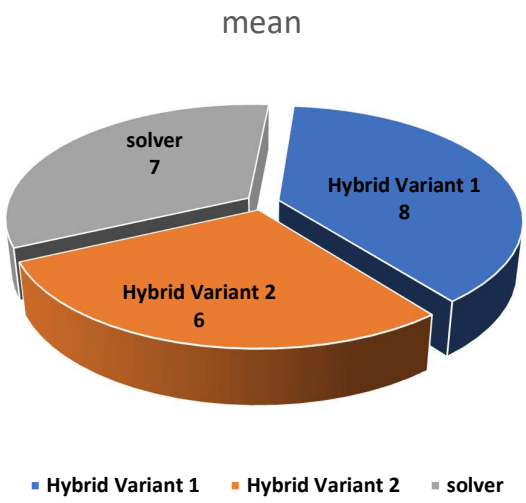

Figure 6.5: Number of wins (mode) between **Hybrid Variant 1**, **Hybrid Variant 2** and CPLEX solver using 2000 time limit

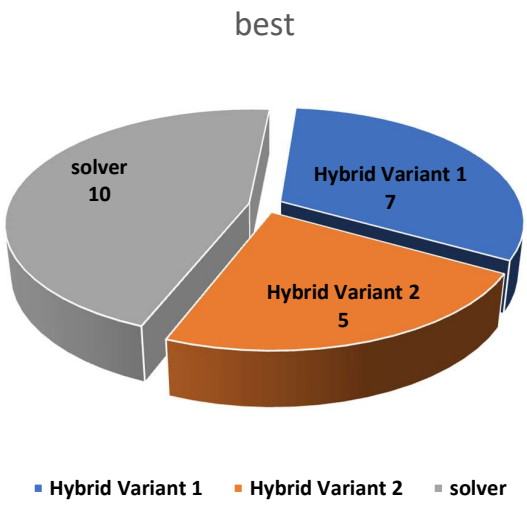

Figure 6.6: Number of wins (best found) between **Hybrid Variant 1**, **Hybrid Variant 2** and CPLEX solver using 2000 time limit

From **Figure 6.6**, we can see that on average, **Hybrid Variant 1** managed to outperform the CPLEX solver with the time limit of 2000 seconds. **Hybrid Variant 1** has the best TE for 8 of the instances compared to the solver that has 7.

However, the solver has 10 wins for the best solutions found while **Hybrid Variant 1** only has 7 wins. We may have better **TE** if we increased the time limit used for the proposed hybrid technique. But as we can see from **Table 6.4**, the running time required might increase immensely to achieve a better **TE**.

6.2.3.2 Hybrid VS metaheuristic technique (genetic algorithm)

In **Chapter 5** of this thesis, we discussed how the hybridisation of the metaheuristic intended to improve the solutions generated by a single metaheuristic technique. Therefore, in this section, we compare the results generated by the proposed technique with the results generated using a metaheuristic technique. The metaheuristic technique chosen is a genetic algorithm (GA).

6.2.3.2.1 Genetic algorithms (GA)

GA is a metaheuristic technique that imitates the biological phenomena by producing better offspring, through a mutation process of two parents where it maintains the best traits from the two parents. We created 100 parents and chose the best 10 percent that have the best fitness value (smallest **TE** values) for the mutation process. The solution limit and the time limit were set to 35 solutions and 300 seconds for each mutation process between Parent A and Parent B.

Table 6.9 gives the best found, mean and mode of the **TE** values obtained for all of the instances using the GA technique. The following **Table 6.10** compares the performance of the solutions (mean) obtained using the GA approach with **Hybrid Variant 1** and **Hybrid Variant 2**.

	total assets	cardinality	GA					
			Best		Mean		Mode	
			TE	seconds	TE	seconds	TE	seconds
indtrack3	89	30	0.481754067	2754.086	0.484804798	2508.5464	0.484766765	2357.9415
		35	0.429886484	1862.885	0.429886484	1866.0688	0.429886484	1866.0688
		40	0.38692187	2673.548	0.387316979	2780.5576	0.387415756	2807.31
indtrack4	98	30	0.430703819	2016.925	0.430703819	2051.24	0.430703819	2051.24
		35	0.373900849	2331.39	0.373900849	2347.869	0.373900849	2347.869
		40	0.344656988	3722.255	0.344915609	3056.0724	0.345057156	2947.586
indtrack5	225	50	0.340397742	6774.716	0.340397742	6779.3784	0.340397742	6779.3784
		75	0.235488117	11472.411	0.236500593	11796.7578	do not exist	
		100	0.19215004	11242.348	0.193017908	11073.6412	do not exist	
indtrack6	457	50	0.595205771	6026.084	0.595205771	6032.3394	0.595205771	6032.3394
		75	0.451718519	8749.587	0.453791761	8776.6182	0.454464484	8785.559667
		100	0.422014697	8724.308	0.422014697	8733.4948	0.422014697	8733.4948
indtrack7	1418	50	0.899214481	13511.784	0.899214481	13512.8946	0.899214481	13512.8946
		75	0.694055155	13512.061	0.734802815	13513.6536	0.754136812	13513.419
		100	0.460963271	13511.154	0.460963271	13511.5564	0.460963271	13511.5564
indtrack8	2151	50	0.731449789	13520.666	0.731449789	13521.3644	0.731449789	13521.3644
		75	0.428562345	13522.5	0.428562345	13522.7598	0.428562345	13522.7598
		100	0.308146556	13516.216	0.308146556	13517.6894	0.308146556	13517.6894

Table 6.9: Best found, mean and mode of the TE generated using the GA approach

	total assets	cardinality	GA compared to Hybrid Variant 1		GA compared to Hybrid Variant 2	
			$\frac{TE_{GA}[15]-TE_{V1}}{TE_{V1}}$ (%)	$\frac{seconds_{GA}[16]-seconds_{V1}}{seconds_{V1}}$ (%)	$\frac{TE_{GA}-TE_{V2}}{TE_{V2}}$ (%)	$\frac{seconds_{GA}-seconds_{V2}}{seconds_{V2}}$ (%)
indtrack3	89	30	(0.91)	502.82	(0.91)	316.87
		35	0.05	144.21	(0.47)	160.85
		40	2.46	271.17	1.81	316.42
indtrack4	98	30	(2.93)	548.56	(0.46)	284.74
		35	(2.45)	614.90	(2.34)	270.94
		40	(0.02)	725.24	(0.02)	355.66
indtrack5	225	50	5.55	524.92	4.34	647.13
		75	2.48	754.27	2.20	992.10
		100	(1.12)	899.84	(0.56)	847.17
indtrack6	457	50	8.17	454.40	7.99	579.64
		75	0.95	779.33	0.95	769.75
		100	(0.07)	936.79	(0.07)	941.22
indtrack7	1418	50	17.07	907.05	16.59	1108.54
		75	44.55	1211.45	44.18	1398.29
		100	23.70	1249.29	15.09	1156.95
indtrack8	2151	50	34.16	1226.23	38.82	905.85
		75	17.26	1397.05	17.26	1396.39
		100	15.36	672.80	4.11	1396.14

Table 6.10: Relative error of solutions (mean) obtained by the GA approach to the solutions (mean) obtained using **Hybrid Variant 1** and **Hybrid Variant 2**

[15] TE_{GA} is the TE obtained using the GA technique

[16] $seconds_{GA}$ is the computational time using the GA technique

From **Table 6.9**, we can see that the solutions generated using the GA approach required large computational time compared to the computational time under the proposed hybrid approach. For the small data sets, ***indtrack3*** and ***indtrack4***, the GA approach recorded around 2000 to 3000 seconds for most of the instances. In the case of larger data set, ***indtrack7*** and ***indtrack8***, the computational time was up to 13500 seconds for all of the instances. These big number of computational time were due to the fact that under the GA approach, the model will have to complete all of the mutation processes in order to find the best solution. Even if the best solution has actually been found in the early stage of the mutation processes, there is no stopping criteria incorporated into the model to exit the process unlike the hybrid approach where it terminates the search process once there is no improvement in the current solution.

We then compare the solutions generated using the GA technique with the solutions obtained using the proposed hybrid technique. **Table 6.10** gives the relative error and the speed of the solutions generated using the GA approach to the solutions obtained under **Hybrid Variant 1** and **Hybrid Variant 2**. It can be observed that the quality for most of the solutions obtained using the GA approach were outperformed by the solutions generated by **Hybrid Variant 1** and **Hybrid Variant 2**, especially for all of the instances in the larger data sets. For ***indtrack7*** and ***indtrack8***, the difference in the **TE** recorded about 15 to 45 percent of increase with noticeably huge gap in the computational time, about 1100 to 1400 times more compared to the computational time under **Hybrid Variant 1** and **Hybrid Variant 2** for most of the instances.

In the following **Figure 6.7, Figure 6.8** and **Figure 6.9**, comparisons are made on the best found, mean and mode of the solutions generated using the proposed hybrid technique, the CPLEX solver and the GA approach.

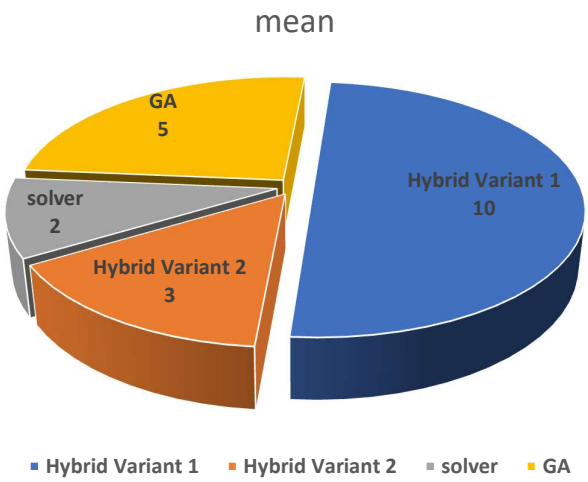

Figure 6.7: Number of wins (mean) between **Hybrid Variant 1**, **Hybrid Variant 2**, the GA technique and the CPLEX solver

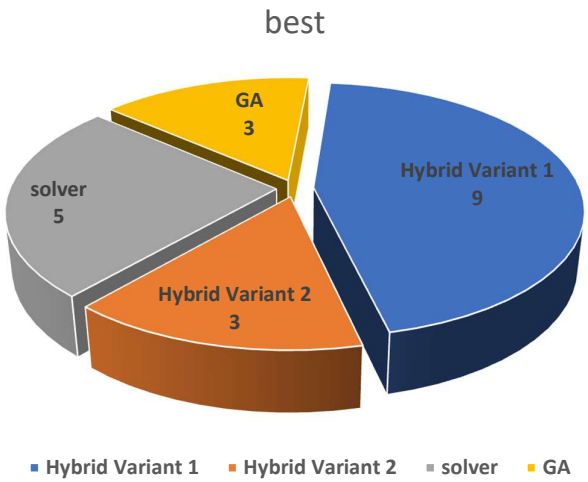

Figure 6.8: Number of wins (best found) between **Hybrid Variant 1**, **Hybrid Variant 2**, the GA technique and the CPLEX solver

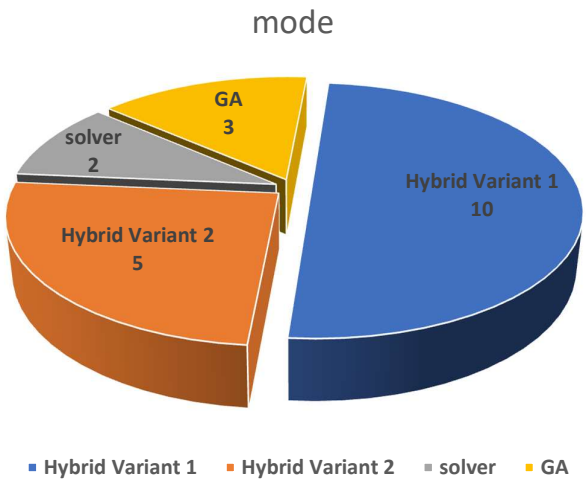

Figure 6.9: Number of wins (mode) between **Hybrid Variant 1**, **Hybrid Variant 2**, the GA technique and the CPLEX solver

We can see that **Hybrid Variant 1** outperformed the other approaches with significant number of wins in all of the 3 comparisons. Although **Hybrid Variant 2** did not record much number of wins, the quality of the solutions generated using this approach were actually better than the solutions generated using the CPLEX solver and the GA approach. The low number of wins was actually due to the better quality provided by the solutions generated using **Hybrid Variant 1**. If we refer to **Table 6.5** and **Table 6.10**, **Hybrid Variant 2** outperformed 15 out of 18 instances of the solutions (mean) obtained using the CPLEX solver and 11 out of 18 instances of the solutions obtained under the GA approach.

The quality of the solutions provided by the GA technique can be enhanced by increasing the number of the parents set to undergo the mutation process or by increasing the time limit. This however will record a much larger computational time. As we mentioned earlier, the larger computational time recorded by the GA technique was due to the mutation processes that need to be completed between the parents. Therefore, by increasing the number of parents set will require more mutation processes, leading to larger computational time.

6.2.4 Comparison to the relaxed LP solutions

Another way to measure the quality of the solutions generated is to see how far they are from the relaxed LP solutions. The following **Table 6.11** compares the solutions (mean) of the proposed hybrid technique, the CPLEX solver and the GA approach with the solutions obtained from the LP relaxation.

For ***indtrack3***, ***indtrack4***, ***indtrack5*** and ***indtrack6***, all of the approaches recorded similar differences to the relaxed LP solution for all of the instances tested. For the biggest cardinality for each of the data sets, 40 for ***indtrack3*** and ***indtrack4*** and 100 for ***indtrack5*** and ***indtrack6***, the **TE** generated compared to the relaxed LP solutions showed tolerable differences. For ***indtrack3*** and ***indtrack4***, the differences ranges from 28 to 36 percent and the difference in the **TE** for ***indtrack5*** is around 15 percent. ***indtrack6*** showed incredibly small differences to the **TE** in the relaxed LP solutions, about 7 percent increase on average under each approach.

In the larger data sets, ***indtrack7*** and ***indtrack8***, the gap between the **TE** generated by all of the approaches to the **TE** of the relaxed LP was quite noticeable. The differences range from 80 to 500 percent. However, it is noticeable that the differences showed by the proposed hybrid technique were almost half from the differences recorded by the CPLEX solver and the GA approach for most of the instances.

Readers should note that the **TE** generated by the approaches discussed in this research were obtained using only smaller percentage of the total assets, especially for the larger data sets. From **Table 6.0**, the relaxed LP solutions used 229 and 234 assets for ***indtrack7*** and ***indtrack8***, respectively. The largest number of the cardinality picked was 100 which is less than half of the number of assets in the relaxed LP solutions. There are some other elements that were not included in the models such as the transaction fees which could affect the return of the index since more number of assets requires higher transaction fees.

	total assets	relaxed LP solution	cardinality	Hybrid Variant 1	Hybrid Variant 2	CPLEX solver	GA
				$\frac{TE_{V1}-TE_{LP}}{TE_{LP}}$ (%)	$\frac{TE_{V2}-TE_{LP}}{TE_{LP}}$ (%)	$\frac{TE_{solver}-TE_{LP}}{TE_{LP}}$ (%)	$\frac{TE_{GA}-TE_{LP}}{TE_{LP}}$ (%)
indtrack3	89	0.285511633	30	71.36	71.36	68.73	69.80
			35	50.49	51.28	50.15	50.57
			40	32.40	33.24	33.08	35.66
indtrack4	98	0.268240316	30	65.41	61.31	61.49	60.57
			35	42.89	42.73	44.45	39.39
			40	28.61	28.61	31.23	28.58
indtrack5	225	0.168588219	50	91.29	93.51	100.39	101.91
			75	36.89	37.27	40.77	40.28
			100	15.79	15.14	15.16	14.49
indtrack6	457	0.394932939	50	39.32	39.56	42.22	50.71
			75	13.83	13.83	13.89	14.90
			100	6.94	6.94	7.40	6.86
indtrack7	1418	0.203944621	50	276.62	278.17	417.12	340.91
			75	149.26	149.89	260.86	260.30
			100	82.72	96.39	106.16	126.02
indtrack8	2151	0.124590627	50	337.59	322.90	487.08	487.08
			75	193.36	193.36	243.98	243.98
			100	114.39	137.56	147.90	147.33

Table 6.11: Relative error of solutions (mean) obtained under **Hybrid Variant 1**, **Hybrid Variant 2**, the GA approach and the CPLEX solver to the relaxed LP solution

[17] TE_{LP} is the TE of the relaxed LP solutions

Looking at the solutions generated using the CPLEX solver and the GA approach, the proposed hybrid technique did extremely well in larger data sets, showing significant differences in the instances tested with huge gap in computational time. For other data sets, the proposed hybrid technique produce better **TE** than the other approaches for most of the instances, although the differences in the **TE** were small. The solutions obtained under the proposed hybrid technique were also the closest to the relaxed LP solutions for most of the instances.

From the comparisons made, it can be concluded that the proposed hybrid technique was indeed capable of providing good solutions in a relatively short amount of computational time, especially for large size problems.

6.3 Gas supply chain problem

6.3.1 Data

Due to the absence of the real data, we generated our own set of data where the details of how the data is generated are provided in the **Appendix A** of this thesis.

The gas supply chain problem is more complex since it has more decision variables compared to the CITP. There were two sets of decision variables involved in the CITP. For the gas supply chain problem, there were four sets of decision variables, making the matrix of this problem much larger than the CITP.

6.3.2 Empirical results

6.3.2.1 The proposed hybrid technique

6.3.2.1.1 Initial partial fixing variant 1

We tested 3 different $\boldsymbol{\alpha}$ values to determine which fixing yields the best result i.e. the minimum total cost. The results are given in the following **Table 6.12**.

Partial fixing	Initial solution (LP)	Initial shaking	Number of facility opened
$\boldsymbol{\alpha = 0.25}$	335635.3705	335458.1823	34
$\boldsymbol{\alpha = 0.5}$	335860.7784	335419.0757	33
$\boldsymbol{\alpha = 0.75}$	336110.2965	335562.2318	34

Table 6.12: The total cost generated using different values of $\boldsymbol{\alpha}$ under initial partial fixing variant 1

Partial fixing	Improvement in the total cost	% of decrease in total cost
$\boldsymbol{\alpha = 0.25}$	177.19	0.05
$\boldsymbol{\alpha = 0.5}$	441.70	0.13
$\boldsymbol{\alpha = 0.75}$	548.07	0.16

Table 6.13: The decrease in the total cost using different values of $\boldsymbol{\alpha}$ under initial partial fixing variant 1

From **Table 6.12**, it shows that the $\boldsymbol{\alpha}$ value that yields the best result is 0.5. It is also interesting to note that, the hybridisation of the LP relaxation and the VNS did show improvement in the objective function, total cost. In the first stage where we use the LP relaxation to partially fix the value of the binary variable, the total cost was 335860.78. When we apply the VNS technique by changing the neighbourhood of the current solution, the total cost showed a decrease of 441.70 to 335419.08, about 0.13 percent difference. The improvement in the total cost also can be seen in the other two cases where each case recorded a decrease of 0.05 percent and 0.16 percent.

Although $\boldsymbol{\alpha = 0.75}$ showed the biggest improvement among the three cases as shown in **Table 6.13**, it still gave a higher total cost than $\boldsymbol{\alpha = 0.5}$, a difference of 143.15. Furthermore, it suggest to open storage facilities at 34 locations, compared to $\boldsymbol{\alpha = 0.5}$ that suggest to open the facilities at 33 locations. The cost of maintaining the storage was not considered in this model and thus it might not be beneficial in the long run. The time period considered in the model was based on the 4 seasons in a year. With time period $\boldsymbol{t = 20}$, the amount calculated was to cover the operational cost for 5 years. The maintenance cost may show a significant difference in the total cost in a longer time period.

6.3.2.1.2 Initial partial fixing variant 2

	Initial solution (LP)	Initial shaking	Number of facility opened	Locations
Initial partial fixing variant 1	335860.78	335419.08	33	3,4,5,6,7,9,10,11, 12,14,15,16,20,21,22,23, 24,25,26,28,31,32,35,36, 37,38,41,42,43,45,48,49,50
Initial partial fixing variant 2	335400.05	335399.74	33	2,4,5,6,7,8,9,10,11, 12,14,15,16,20,21,22,23, 25,26,28,31,32,35,36,37, 38,41,42,43,45,48,49,50

Table 6.14: The comparison of the total cost generated by initial partial fixing variant 1 and initial partial fixing variant 2

From **Table 6.14**, the initial partial fixing variant 2 performed better than variant 1 with a decrease in total cost of 19.34. Although the difference in the total cost is small, about 0.0058 percent, it might bring a significant impact in the long run.

Both variants suggest the same number of storage facilities should be open. However, some of the suggested locations were different. Under the initial partial fixing variant 1, the model suggests locations 3 and 24 but not under initial partial fixing variant 2 where it suggests locations 2 and 8 instead. Further analysis showed that on average, locations 2 and 8 did offer a cheaper gas price than locations 3 and 24. This most likely contributed to the lower amount of total cost under initial partial fixing variant 2 and may highly impacts the total cost in the long run.

6.3.2.2 CPLEX solver and brute force method

The gas supply chain problem has more decision variables compared to the CITP. Like the CITP, the CPLEX solver also failed to solve the gas supply chain problem due to the excessive usage of

memory. Hence, we took the same approach, the brute force method by implementing some limitations on the time limit and solution limit. We used the same time limit that we imposed under initial partial fixing variant 2, 10 000 seconds. For the solution limit, we used 20 and increased the limit by 10 for the next run each time the CPLEX solver managed to find an improved solution in the current run. The programme terminated once the current solution is the same or worse than the one in the previous run.

We then compared the quality of the solutions generated by the CPLEX solver with the solutions found by the proposed hybrid technique using initial partial fixing variant 2. The computational time given is the cumulative of all of the runs.

	Total costs	seconds	Number of facility opened	Locations
Initial partial fixing variant 2	335399.74	15980.40	33	2,4,5,6,7,8,9,10,11,12,14, 15,16,20,21,22,23,25,26,28,31,32, 35,36,37,38,41,42,43,45,48,49,50
CPLEX solver	335399.74	19987.27	33	2,4,5,6,7,8,9,10,11,12,14, 15,16,20,21,22,23,25,26,28,31,32, 35,36,37,38,41,42,43,45,48,49,50

Table 6.15: The comparison of the total cost generated by initial partial fixing variant 2 and the CPLEX solver

The proposed hybrid technique using initial partial fixing variant 2 managed to replicate the solutions found by the CPLEX solver in relatively shorter computational time, about 20 percent less time required by the CPLEX solver.

The results obtained in **Table 6.15** were obtained when we allowed a maximum of 15 locations of $\boldsymbol{l1}$ to supply the gas to location $\boldsymbol{l}$. We then reduce this number to 10 and **Table 6.16** shows the solutions generated by initial partial fixing variant 2 and the CPLEX solver.

	Total costs	seconds	Number of facility opened	Locations
Initial partial fixing variant 2	335418.22	12379.00	33	2,4,5,6,7,9,10,11,12,13,14,15, 16,20,21,23,24,25,26,28,31,32,35, 36,37,38,41,42,43,45,48,49,50
CPLEX solver	335825.83	20090.00	34	3,4,5,6,7,9,10,11,12,15,16,20, 21,22,23,24,25,26,28,31,32,34,35, 36,37,38,41,42,43,44,45,48,49,50

Table 6.16: The total cost generated by initial partial fixing variant 2 and the CPLEX solver when the number of locations allowed to supply gas was reduced to 10

The proposed hybrid technique provides better solutions than the CPLEX solver, giving lower total costs with a difference of 407.61 or about 0.12 percent decrease in the total costs. The computational time required by the proposed method also was a lot shorter compared to the time recorded by the CPLEX solver.

Furthermore, the solutions suggested by the CPLEX solver requires 34 locations of storage facilities should be open but for only 33 locations were suggested under the proposed hybrid technique.

6.3.3 Enhancing the solutions

6.3.3.1 Bound tightening

One way to enhance the solutions is to tighten the bounds. We set a new upper value $\boldsymbol{m_l}$ to the maximum value of storage, $\boldsymbol{M}$ and observed the solutions generated. The following **Table 6.15** shows the solutions obtained using initial partial fixing variant 1 under different values of upper value $\boldsymbol{m_l}$.

	Total cost	Number of storage facility opened
$\boldsymbol{u_l = 1.5m_l^*}$	338424.58	37
$\boldsymbol{u_l = 2.0m_l^*}$	336868.39	37
$\boldsymbol{u_l = 2.5m_l^*}$	336071.60	36
$\boldsymbol{u_l = 3.0m_l^*}$	335873.15	36
$\boldsymbol{u_l = 4.3m_l^*}$	336078.29	35

Table 6.15: Comparison of the solutions using different $\boldsymbol{m_l}$

From the solutions obtained, it is clear that tightening the bound did not help in improving the solutions for the gas supply problem. This may imply that the original value of the maximum value of storage, $\boldsymbol{M}$ is optimum and reducing the capacity of the storage facility will force the model to opt for opening more facilities, as reflected in **Table 6.15**. This led to the increase in the total costs.

6.3.3.1 Intensified shaking

The following **Table 6.16** shows the comparison in the solutions before and after the shaking process. The $\boldsymbol{\beta}$ values were set at $\boldsymbol{\beta = 0.6, 0.7, 0.8}$.

Hybrid	Initial solution (LP)	Initial shaking	Intensified shaking
Initial partial fixing variant 1	335860.78	335419.08	335419.08
Initial partial fixing variant 2	335400.05	335399.74	335399.74

Table 6.16: Comparison of the solutions before and after the intensified shaking process

Applying intensified shaking also did not help in improving the solutions obtained. Both variants did not show any improvement in the total cost.

In conclusion, the proposed method was successfully implemented to solve the gas supply problem and did generate satisfactory solutions in a considerable amount of computational time. There was no improvement made when bound tightening and intensified shaking were performed. The absence of the real data may be the contributing factor to the solutions obtained.

Chapter 7.0: Conclusions and findings

7.1 Conclusions

The research presented in this thesis focuses on the development of a hybrid metaheuristic between an exact method and a metaheuristic technique for CO problems.

In **Chapter 1**, we introduced IP, a form of CO problems. We discussed about the types of IP with more focus on the MBIP. Then, we introduced the concept of heuristics and metaheuristics, a framework of search methods that have become increasingly popular in solving computationally hard IP problems. The contributions of this research are also discussed at the end of this chapter.

Dumitrescu and Stützle (2003) concluded that IP methods and local search methods have the most significant success in solving CO problems. In **Chapter 2**, we presented a survey on some of the most commonly used IP methods and local search methods, as well as its extensions. A brief introduction to the techniques enables the readers to gain some basic idea on the methods.

Hybrid metaheuristics are relatively new but are enjoying much popularity among researchers due to their ability to exploit the complementary character of different optimisation techniques. In **Chapter 3**, the concept of hybrid metaheuristics was presented, focusing on the collaborative combinations between metaheuristics and ILP techniques. Readers are able to gain some insight about this technique as well as the existing literature available.

To demonstrate the flexibility of our proposed method, two similar yet different MBIP problems were studied. In **Chapter 4**, the structures of the CITP and the gas supply chain problem were discussed where readers could see the characteristics of the two problems. The two problems were formulated as MBIP problems.

In **Chapter 5**, the proposed solution method, the hybrid of LP relaxations and VNS, was presented. The complementary combinations of the two techniques were described. Readers were

able to see how the LP relaxation was used to generate an initial point. VNS was then used to redirect the search into promising directions of the search space where better optimum may be found.

The experimental results of both studied problems were presented in **Chapter 6** of this thesis. Comparisons were made with other techniques to analyse the performance of the solutions generated using the proposed hybrid technique.

7.2 Findings

The aim of this research was to provide an alternative method to solve the computationally hard MBIP problems. A hybrid between the LP relaxation and the VNS was proposed. The proposed hybrid method used LP relaxation of the MBIP to generate initial solutions and guide the search process. VNS was then used to improve the solutions guided by the information provided by the LP relaxation.

For the CITP, the results provided and the comparisons made to the other methods showed that our proposed method did provide satisfactory solutions. In most instances, our proposed method dominates the results obtained by the CPLEX solver and the GA approach with significantly shorter amount of time. It is difficult to compare our results with other authors' works since few authors use the same data set, the approach taken also is different. Therefore, the results from the literature are not comparable to the results obtained in this research.

In the gas supply problem, since the real data is absent, we generated our own set of data by incorporating some of the important features of the gas supply chain problem such as the gas price, gas demand, and the cost of opening a facility. Therefore, apart from comparing our results with the performance of the CPLEX solver, we cannot compare the results with other works in the literature. From the results provided in **Section 6.3**, it is clear that the proposed method did provide better solutions compared to the CPLEX solver.

The main motivation of this research was to build a general framework of a solution approach that can be easily implemented for different classes of CO problems, particularly in this research, to MBIP problems. Our aim was to demonstrate the flexibility of the proposed method, focusing more on solving the MBIP at hand, with satisfactory solutions, rather than finding the best solutions to the problem.

VNS offers great potential as a search agent and compared to other LS based metaheuristics methods, VNS is easier to implement and more user-friendly. To overcome the randomness, we used LP relaxation as a guide to search in more promising areas that may lead to better solutions. We want to maintain the simplicity of the VNS characteristics but at the same time to provide satisfactory solutions.

There are several characteristics of a good metaheuristic maintained in our proposed method:

1. Simplicity and easy implementation.
2. Generality that demonstrates the flexibility/ability of implementing the technique to different classes of CO problems by minor adjustments.
3. Precision where the steps of our proposed method can be formulated in precise mathematical terms.
4. Effectiveness where the proposed method was able to produce satisfactory solutions in moderate computing time.
5. User-friendliness where the proposed method is clearly expressed, easy to understand and, easy to use.
6. Interactivity which means that the proposed method allows the user to incorporate this knowledge to improve the resolution process.

From the results obtained, we managed to demonstrate the flexibility of our proposed method to be implemented for different types of CO problems. With minor adjustments, the proposed method yields satisfactory solutions for both problems. The only adjustment needs was the

decision variables that were used to guide the search process; the weight of asset $\boldsymbol{i}$, $\boldsymbol{x_i}$ for the CITP problem and the relaxed binary decision variable of opening a facility at location $\boldsymbol{l}$, $\boldsymbol{z_l}$ for the gas supply problem.

Chapter 8.0: Future research

8.1 Effects of the heuristic controls

This research focuses on proposing a method that can solve a variety of MBIP problems rather than finding the best solution for a specific problem. The integer element in the decision variables of the MBIP problems often caused these problems become computationally challenging and too expensive for the exact method to solve. More often than not, MIP solvers such as CPLEX ran out of memory while enumerating the combinations of the possible solutions to the MBIP problem being solved.

As mentioned under **Section 1.5.2**, Fischetti and Lodi (2003) stated that many commercial MIP solvers provide some flexibility to have certain heuristic controls over some parameters that affect:

1) the exploration of the branching rules
2) the frequency of application of the internal heuristics
3) the fact of emphasizing the solution integrality rather than its optimality

and many more. However, the authors also stressed that the flexibility given may not be adequate for some cases which led to many opted for specialised heuristics.

In this research, a hybrid technique between a metaheuristics and an exact method was proposed. The hybrid technique is a collaborative combination of the VNS (a metaheuristics technique) with LP relaxation (an exact method). It uses the information provided by the LP relaxation to guide the search for solutions in the neighbourhood by VNS. This specialised heuristic uses the flexibility provided by the MIP solver to exploit item (1) above. It is interesting to see if the hybridisation of the LP relaxation and the VNS introduced in this research will be able to exploit item (2) and (3) in solving the MBIP problems.

8.2 Implementations to the other MBIP problems

The general framework of the proposed hybrid technique that was designed to give the flexibility to the technique being implemented to different types of the MBIP problems. The proposed hybrid technique was successfully implemented on both of the problems studied in this research. This proved that the motivation of this research to propose a technique that focuses on solving the MBIP in general rather than achieving the best solutions to a specific studied problem was accomplished.

There are many other MBIP problems that are widely studied by many researchers in various fields of researches. One direction that this research may take is to implement the proposed hybrid technique as an alternative technique to the other MBIP problems, especially for researchers that is looking for a technique that requires a short amount of computational time.

8.3 Consistency of the solutions generated (CITP)

The proposed hybrid technique provides good solutions to both problems in a relatively shorter computational time, as discussed in **Chapter 6** of this thesis. From the empirical results and comparisons made to the other techniques, it showed that the proposed hybrid technique does provide a promising potential as a solution method to the CO problems, in particular to the MBIP problems.

However, due to the limitations imposed both on the solution limit and time limit, the model showed variations in the solutions each time we solved the problem (for the CITP). Although the variations recorded an average of 0.42 and 0.41 of differences between the solutions generated for **Hybrid Variant 1** and **Hybrid Variant 2** respectively, it is more desirable if the model is able to produce the same **TE** value each time we solve the problem. Interested readers or researchers can focus on making the model more robust by finding the method to fix this issue.

8.4 Partial solution construction

One important features that was introduced in the proposed method was the way of selecting and forcing some of the variables into the partial solution. This feature showed a promising potential of aiding the search process for better solutions to the studied problems as well as reducing the computational time needed. Future researches may consider this feature to be applied in other techniques and observed whether the same results obtained in this research can be achieved.

8.5 Element of risk

One of the limitations in the models used to solve the CITP and the gas supply chain problem in this research is that none of them incorporate the element of risk into the models. The work of this research can be expanded to study the incorporation of risk into the models. One of the risk measure that may be considered is the conditional value of risk, CVaR.

Bibliography

Aarts, E.H.L., and Lenstra, J.K. (1997). *Local search in combinatorial optimization*. Chichester: Wiley.

Aickelin, U., and Clark, A. (2011). Heuristic optimisation. *Journal of the Operational Research Society*, *62 (2)*, 251-252.

Applegate, D. L., Bixby, R. E., Chvátal, V., and Cook, W. J. (1998). On the solution of the travelling salesman problem. *Documenta Mathematica Extra Volume ICM III*, 645-656.

Atamtürk, A., and Savelsbergh, M. W. P. (2005). Integer-programming software systems. Annals of Operations Research, 140(1), 67-124. doi: https://doi.org/10.1007/s10479-005-3968-2

Beasley, J. E. (1990). OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, *41(11)*, 1069-1072, doi: 10.1057/jors.1990.166

Beasley, J. E., Meade, N., and Chang, T., -J. (2003). *An evolutionary heuristic for the index tracking problem. European Journal of Operational Research*, *148(3)*, 621-643. Retrieved from https://doi.org/10.1016/S0377-2217(02)00425-3

Bennell, J. (2015). *Part 1. Combinatorial optimization problems* [Class handout]. Southampton, United Kingdom. University of Southampton, Combinatorial Optimization.

Blum, C., Blesa Aguilera, M.J., Roli, A., and Samples, M. (2008). Hybrid metaheuristics - An emerging approach to optimization. *Volume 114 of Studies in Computational Intelligence*. Berlin, Germany: Springer Verlag.

Blum, C., Puchinger, J., Raidl, G., and Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, *11 (6)*, 4135-4151.

Blum, C., Roli, A., and Alba, E. (2005). An introduction to metaheuristic techniques. In: *Parallel Metaheuristics: A New Class of Algorithms*, *47*, 3-42. New York, USA: John Wiley & Sons. doi: 10.1002/0471739383.ch1

Brilliant.org (2019). Combinatorial optimization. Retrieved from https://brilliant.org/wiki/combinatorial-optimization/

Canakgoz, N. A., and Beasley, J. E. (2009). Mixed-integer programming approaches for index tracking and enhanced indexation. *European Journal of Operational Research*, *196(1)*, 384-399. Retrieved from https://doi.org/10.1016/j.ejor.2008.03.015

Christofides, N., Mingozzi, A., Toth, P., and Sandi, C. (1979). *Combinatorial optimization*. Bath, Great Britain: John Wiley & Sons.

Chu, P., and Beasley, J. (1998). A genetic algorithm for the multidimensional knapsack problem. Journal of Heuristics, 4(1), 63-86. doi: https://doi.org/10.1023/A:1009642405419

Consoli, S., and Darby-Dowman, K. (2007). Combinatorial optimization and metaheuristics. *Annals of Operations Research*, 140(1), 189-213.

Cotta, C. (1998). A study of hybridisation techniques and their application to the design of evolutionary algorithms. *AI Communications* ,11, 223-224.

Cotta, C., Talbi, E-G., and Alba, E. (2005). *Parallel metaheuristics – A new class of algorithms*. Hoboken, New Jersey: Wiley & Sons.

Dantzig, G.B. (1963). *Linear programming and extensions*. Princeton, New Jersey: Princeton University Press.

Daskin, M. S., and Maass, K., L. (2015). The *p*-median problem. In: Laporte, G., Nickel, S., Saldanha da Gama, F. (Eds) *Location Science*, 21-45. Springer International Publishing. doi: https://doi.org/10.1007/978-3-319-13111-5_2

Dumitrescu, I., and Stützle, T. (2003). *Combinations of local search and exact algorithms*. In: Raidl, G. R. et al. (Eds.), Applications of Evolutionary Computing, Vol. 2611 of Lecture Notes in Computer Science. Berlin, Germany: Springer.

El-Abd, M., and Kamel, M. (2005). A taxonomy of cooperative search algorithms. In: Blesa, M. J., Blum, C., Roli, A., and Samples, M. (Eds), *HM 2005: Hybrid Metaheuristics, Lecture Notes in Computer Science*, *3636*, 32-41. Berlin : Springer.

Feo, T. A., and Resende, M. G. C. (1989). A probabilistic heuristic for computationally difficult set covering problem. *Operations Research Letters*, *8*, 67-71.

Festa, P. (2014, July). *A Brief Introduction to Exact, Approximation, and Heuristic Algorithms for Solving Hard Combinatorial Optimization Problems*. Paper presented at 2014 16th International Conference on Transparent Optical Networks (ICTON): IEEE. doi: 10.1109/ICTON.2014.6876285

Fischetti, M., and Lodi, A. (2003). Local branching. *Mathematical Programming Series B*, 98, 23-47.

Frontline Solvers. (2018). Retrieved from https://www.solver.com/integer-constraint-programming

Galli, L. (2018). *Algorithms for integer programming* [Class handout]. Retrieved from http://www.di.unipi.it/optimize/Courses/RO2IG/aa1617/ROII1617.html

Gendreau, M., and Potvin, J, -Y. (2005). Metaheuristics in combinatorial optimization. *Annals of Operations Research*, *140*, 189-213.

Gendreau, M., and Potvin, J, -Y (Eds). (2010). *Handbook of metaheuristics*. New York, USA: Springer.

Gill, P.E., Murray W., and Wright, M.H. (1982). *Practical optimization*. London, United Kingdom: Academic Press.

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

Glover, F., and Martí, R. (2006). Tabu search. In: Alba, E., and Martí, R. (Eds), *Metaheuristic Procedure for Training Neutral Networks, Operations Research/Computer Science Interfaces Series, Volume 36*. Boston, Massachusetts: Springer.

Gomory, R. E. (1963). An algorithm for integer solutions to linear programs. In Graves, R. L., and Wolfe, P. (Eds), *Recent Advances in Mathematical Programming* (pp 269-302). New York: McGraw-Hill.

Gustavsson, E. (2015). Topics in convex and mixed binary linear optimization. (Doctoral dissertation). Retrieved from http://hdl.handle.net/2077/38634

Hansen, P., and Mladenović, N. (2001). Variable neighbourhood search: Principles and applications. *European Journal of Operational Research*, *130*, 449-467.

Hansen, P., Mladenović, N., and Moreno Pérez, J. A. (2008). Variable neighbourhood search: Methods and applications. *4OR*, *6(4)*, 319-360. doi: https://doi.org/10.1007/s10288-008-0089-1

Hansen, P., Mladenović, N., and Moreno Pérez, J. A. (2010). Variable neighbourhood search: Algorithms and apllications. *Annals of Operations Research*, *175*, 367-407.

Heragu, S. S., Mazacioglu, B., and Fuerst, K. D. (1994). Meta-heuristic algorithms for the order picking problem. International Journal in Industrial Engineering, 1(1), 67-76.

Hoos, H., and Stützle, T. (2004). *Stochastic local search: Foundations and applications*. San Francisco, California: Morgan Kaufmann Publishers Inc.

Joslin, D. E., and Clements, D. P. (1999). Squeaky wheel optimization. *Journal of Artificial Intelligence Research*, *10*, 353-373.

Jörnsten, K., and Lokketangen, A. (1997). Tabu search for weighted *k*-cardinality trees. *Asia-Pacific Journal of Operations Research*, *14(2)*, 9-26.

Kirkpatrick, S., Gelatt, C., and Vecchi, M. (1983). Optimization by simulated annealing. Science, 220(4598), 671-680. Retrieved from http://www.jstor.org/stable/1690046

Klau, G., Ljubić, I., Moser, A., Mutzel, P., Neuner, P., Pferschy, U., Raidl, G., and Weiskircher, R. (2004). Combining a memetic algorithm with integer programming to solve the prize-collecting Steiner tree problem. In: Deb, K. (Eds), Genetic and Evolutionary Computation – GECCO 2004. Lecture Notes in Computer Science, 3102, 1304-1315. Berlin: Springer, Heidelberg. doi: https://doi.org/10.1007/978-3-540-24854-5_125

Koberstein, A., Lucas, C., Wolf, C., and König, D. (2011). Modeling and optimizing risk in the strategic gas-purchase planning problem of local distribution companies. Journal of Energy Markets, 4(3), 47-68. doi: 10.21314/JEM.2011.061

Kwiatowski, J.W. (1992). Algorithm for index tracking. *Journal of Mathematics Applied in Business and Industry*, *4*, 279-299.

Land, A. H., and Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, *28(3)*, 497-520. doi: 10.2307/1910129

Lin, A. Z. –Z., Bean, J., and White, I. C. C. (2004). A hybrid genetic/optimization algorithm for finite horizon partially observed markov decision process. Journal on Computing, 16(1), 27-38.

Lin, B.W.Y., and Rardin, R. L. (1977). Development of parametric generating procedure for integer programming test problems. *Journal of the ACM*, *24(3)*, 465-472.

Little, J.D. C., Murty, K. G., Sweeney, D. W., and Karel, C. (1963). An algorithm for the traveling salesman problem. *Operations Research*, *11(6),* 972-989. doi: 10.1287/opre.11.6.972

McCarl, B. A., and Spreen, T. H. (1997). *Applied mathematical programming using algebraic systems*. Retrieved from http://agecon2.tamu.edu/people/faculty/mccarl-bruce/books.htm

Metaheuristic Networks. (2000). Retrieved from http://www.metaheuristics.org/index.php%3Fmain=1.html

MIT (2015). *Introduction to LP formulations* [Power point slides]. Retrieved from https://ocw.mit.edu/courses/sloan-school-of-management/15-053-optimization-methods-in-management-science-spring-2013/tutorials/MIT15_053S13_tut01.pdf

Mitchell, J. E. (2002). Branch-and-cut algorithms for combinatorial optimization. In: *Handbook of Applied Optimization* (pp 65-77). Oxford, United Kingdom: Oxford University Press.

Mladenović, N., and Hansen, P. (1997). Variable neighbourhood search. *Computers & Operations Research*, *24*, 1097-1100.

Nagar, A., Heragu, S. S., and Haddock, J. (1995) A meta-heuristic algorithm for a bi-criteria scheduling problem. *Annals of Operations Research*, *63*, 397-414.

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

Pan, L. (2015). Cutting plane method [Power Point slides]. Retrieved from https://www.math.cuhk.edu.hk/course_builder/1415/math3220/L5.pdf

Pirlot, M. (1996). General local search methods. European Journal of Operational Research, 9(23), 493-511.

Plateau, A., Tachat, D., and Tolla, P. (2002). A hybrid search combining interior point methods and metaheuristics for 0-1 programming. *International Transactions in Operations Research*, *9*, 731-746.

Puchinger, J., and Raidl, G. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In: *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, 41-53. Berlin, Germany: Springer.

Raidl, G. R. (2006). A unified view on hybrid metaheuristics. In: Almeida, F., Blesa Aguilera, M. J., Blum, C., Moreno Vega, J. M., Pérez, M. P., Roli, A., and Samples, M. (Eds), *Proceedings of HM 2006 – Third International Workshop on Hybrid Metaheuristics, Vol. 4030 of Lecture Notes in Computer Science* (pp 1–12). Berlin, Germany: Springer Verlag.

Raidl, G. R., and Feltl, H. (2004). An improved hybrid genetic algorithm for the generalized assignment problem. In: Haddadd, H. M., Omicini, A., Wainwright, R. L., and Liebrock, L. M. (Eds.), *Proceedings of the 2004 ACM Symposium on Applied Computing* (pp 990-995). Nicosia: Cyprus: ACM New York.

Rajab, R.S. (2012). *Some applications of continuous variable neighbourhood search metaheuristic (mathematical modelling)* (Doctoral dissertation). Retrieved from Brunel University Research
Archive.

Rouse, M. (2018, February). *Definition soft computing*. Retrieved from https://whatis.techtarget.com/definition/soft-computing

Sierksma, G., and Zwols, Y. (2015). *Linear and integer optimization: Theory and practice* (3rd ed.). Boca Raton, Florida: CRC Press.

Stützle, T. (1999). *Local search algorithms for combinatorial problems: Analysis, improvements, and new applications, Volume 220 of DISKI*. Sankt Augustin, Germany: Infix.

Talbi, E.G. (2002). A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, *8*, 541-564.

Talbi, E.G. (2009). *Metaheuristics: From design to implementation*. Hoboken, New Jersey: John Wiley & Sons.

Toth, P., and Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, *15*, 333-346.

Vasquez, M., and Hao, J. K. (2001). A hybrid approach for the 0-1 multidimensional knapsack problem. In: *Proceedings of the International Joint Conferences on Artificial Intelligence Organization*, 1, 328-333. Seattle.

Voss, S., Martello, S., Osman, I. H., and Roucairol, C. (1999). *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization.* Boston, Massachusetts: Kluwer Academic Publishers.

Voudouris, C., and Tsang, E.P.K. (1995). *Guided local search* (Technical Report CSM-247).

Winston, W.L. (2004). *Operations research: applications and algorithms* (4th ed.). Belmont, California: Duxbury Press.

Appendix A

DEMAND AND PRICE SCENARIOS GENERATION OF THE GAS SUPPLY PROBLEM

Prices for natural gas are mainly influenced by the weather, demographics, economic growth, fuel competition, storage and exports. However, in this problem, the projection of the gas prices and demands are only affected by the weather.

The demand and price of gas are the random variables that bring the uncertainty factor into the gas supply modelling problem. These two elements have to be properly considered in the decision-making model. With a known probability distribution (from statistical data), the future values of the two random variables are predicted by generating scenarios. A probabilistic model or simulation generates a batch of scenarios to represent how the demand and price will unfold in the future. 100 scenarios of gas price and demand are forecasted based on the three major weather outcomes; normal, cold and very cold.

DISTRIBUTION OF THE GAS DEMAND AND GAS PRICE

Under this model, we assume that the price and demand of gas follow a Uniform distribution based on the outcomes of weather.

THE SCENARIO GENERATION ALGORITHM

We define

i. $\boldsymbol{f = 1..100}$ are scenarios

ii. $\boldsymbol{s = 1..3}$ are the major outcomes of winter

iii. $\boldsymbol{P_{t,l,s}}$ is the expected value of price for major outcome $\boldsymbol{s}$ at location $\boldsymbol{l}$ at time $\boldsymbol{t}$

iv. $\boldsymbol{D_{t,l,s}}$ is the expected value of demand for major outcome $\boldsymbol{s}$ at location $\boldsymbol{l}$ at time $\boldsymbol{t}$

v. $\boldsymbol{P'_{t,l,f}}$ is the generated price value of scenario $\boldsymbol{f}$ at location $\boldsymbol{l}$ at time $\boldsymbol{t}$

vi. $\boldsymbol{D'_{t,l,f}}$ is the generated demand value of scenario $\boldsymbol{f}$ at location $\boldsymbol{l}$ at time $\boldsymbol{t}$

vii. $\boldsymbol{P_s}$ is the minimum value of price for major outcome $\boldsymbol{s}$

viii. $\boldsymbol{D_s}$ is the minimum value of demand for major outcome $\boldsymbol{s}$

Then we define $\boldsymbol{a = A, b = B}$ and let $\boldsymbol{k = f - |\frac{f}{3}| \times 3}$ where $\boldsymbol{k}$ is sampled 100 times; $\boldsymbol{k}$ takes the value of either $0, 1$ or 2 where each represents either warm, cold or very cold forecasted winter scenarios respectively and $\boldsymbol{f}$ is the scenario number. The two boundaries of $\boldsymbol{a}$ and $\boldsymbol{b}$ are the lower and upper value of the uniform distribution.

We let

$$\boldsymbol{P'_{1,l,f} = P_{1,l,cold}} \qquad : \forall\, f, l$$

$$\boldsymbol{D'_{1,l,f} = D_{1,l,cold}} \qquad : \forall\, f, l$$

Given $\boldsymbol{k}$ then for $\boldsymbol{t \geq 2}$

$$\boldsymbol{P'_{t,l,f} = max\left(P_s, U(a,b) \times P_{t,l,k}\right)} \qquad : \forall\, t \geq 2, l, f$$

$$\boldsymbol{D'_{t,l,f} = max\left(D_s, U(a,b) \times D_{t,l,k}\right)} \qquad : \forall\, t \geq 2, l, f$$

Appendix B

Full results of Hybrid Variant 1

					best		error	speed	mean		error	speed
dataset	cardinality	time (n)	TE	seconds	TE	seconds			TE	seconds		
indtrack3	30	1	0.4892474495	415.801	0.4892474495	415.801	0.00%	0.00%	0.4892474495	416.133	0.00%	-0.08%
		2	0.4892474495	415.843			0.00%	0.01%			0.00%	-0.07%
		3	0.4892474495	416.392			0.00%	0.14%			0.00%	0.06%
		4	0.4892474495	416.343			0.00%	0.13%			0.00%	0.05%
		5	0.4892474495	416.286			0.00%	0.12%			0.00%	0.04%
	35	1	0.4296554491	764.278	0.4296554491	764.113	0.00%	0.02%	0.4296554491	764.1232	0.00%	0.02%
		2	0.4296554491	764.268			0.00%	0.02%			0.00%	0.02%
		3	0.4296554491	764.113			0.00%	0.00%			0.00%	0.00%
		4	0.4296554491	764.194			0.00%	0.01%			0.00%	0.01%
		5	0.4296554491	763.763			0.00%	-0.05%			0.00%	-0.05%
	40	1	0.3780275727	747.299	0.3780275727	744.8	0.00%	0.34%	0.3780275727	749.1302	0.00%	-0.24%
		2	0.3780275727	745.845			0.00%	0.14%			0.00%	-0.44%
		3	0.3780275727	747.447			0.00%	0.36%			0.00%	-0.22%
		4	0.3780275727	744.8			0.00%	0.00%			0.00%	-0.58%
		5	0.3780275727	760.26			0.00%	2.08%			0.00%	1.49%
indtrack4	30	1	0.444432504	294.573	0.440715907	323.701	0.84%	-9.00%	0.4436891842	316.274	0.17%	-6.86%
		2	0.440715907	323.701			0.00%	0.00%			-0.67%	2.35%
		3	0.4444325035	321.145			0.84%	-0.79%			0.17%	1.54%
		4	0.4444325035	321.001			0.84%	-0.83%			0.17%	1.49%
		5	0.4444325035	320.95			0.84%	-0.85%			0.17%	1.48%
	35	1	0.3829354839	327.984	0.3752295070	320.572	2.05%	2.31%	0.3832826480	328.4198	-0.09%	-0.13%
		2	0.3860134519	330.717			2.87%	3.16%			0.71%	0.70%
		3	0.3862213451	332.132			2.93%	3.61%			0.77%	1.13%
		4	0.3860134519	330.694			2.87%	3.16%			0.71%	0.69%
		5	0.3752295070	320.572			0.00%	0.00%			-2.10%	-2.39%
	40	1	0.3449795047	370.417	0.3449795047	370.247	0.00%	0.05%	0.3449795047	370.3254	0.00%	0.02%
		2	0.3449795047	370.247			0.00%	0.00%			0.00%	-0.02%
		3	0.3449795047	370.329			0.00%	0.02%			0.00%	0.00%
		4	0.3449795047	370.286			0.00%	0.01%			0.00%	-0.01%
		5	0.3449795047	370.348			0.00%	0.03%			0.00%	0.01%

Full results of Hybrid Variant 1 (cont)

					best		error	speed	mean		error	speed
dataset	cardinality	time (n)	TE	seconds	TE	seconds			TE	seconds		
indtrack5	50	1	0.3196470269	1146.535	0.3196470269	1134.432	0.00%	1.07%	0.3225005629	1084.8396	-0.88%	5.69%
		2	0.3196470269	1141.635			0.00%	0.63%			-0.88%	5.24%
		3	0.3273143445	900.741			2.40%	-20.60%			1.49%	-16.97%
		4	0.3196470269	1134.432			0.00%	0.00%			-0.88%	4.57%
		5	0.3262473894	1100.855			2.06%	-2.96%			1.16%	1.48%
	75	1	0.2307843963	1392.761	0.2307843963	1371.642	0.00%	1.54%	0.2307843963	1380.9148	0.00%	0.86%
		2	0.2307843963	1371.642			0.00%	0.00%			0.00%	-0.67%
		3	0.2307843963	1383.926			0.00%	0.90%			0.00%	0.22%
		4	0.2307843963	1373.54			0.00%	0.14%			0.00%	-0.53%
		5	0.2307843963	1382.705			0.00%	0.81%			0.00%	0.13%
	100	1	0.1951870169	1253.633	0.1948376568	1055.679	0.18%	18.75%	0.1952044074	1107.5438	-0.01%	13.19%
		2	0.1948376568	1055.679			0.00%	0.00%			-0.19%	-4.68%
		3	0.1951870169	1082.744			0.18%	2.56%			-0.01%	-2.24%
		4	0.1951870169	1245.002			0.18%	17.93%			-0.01%	12.41%
		5	0.1956233293	900.661			0.40%	-14.68%			0.21%	-18.68%
indtrack6	50	1	0.5502344866	1087.837	0.5502344866	1087.177	0.00%	0.06%	0.5502344866	1088.0768	0.00%	-0.02%
		2	0.5502344866	1087.177			0.00%	0.00%			0.00%	-0.08%
		3	0.5502344866	1087.578			0.00%	0.04%			0.00%	-0.05%
		4	0.5502344866	1089.122			0.00%	0.18%			0.00%	0.10%
		5	0.5502344866	1088.67			0.00%	0.14%			0.00%	0.05%
	75	1	0.4495350040	999.646	0.4495350040	998.186	0.00%	0.15%	0.4495350040	998.1014	0.00%	0.15%
		2	0.4495350040	998.186			0.00%	0.00%			0.00%	0.01%
		3	0.4495350040	997.346			0.00%	-0.08%			0.00%	-0.08%
		4	0.4495350040	997.631			0.00%	-0.06%			0.00%	-0.05%
		5	0.4495350040	997.698			0.00%	-0.05%			0.00%	-0.04%
	100	1	0.4223256180	841.973	0.4223256180	840.514	0.00%	0.17%	0.4223256180	842.3612	0.00%	-0.05%
		2	0.4223256180	840.756			0.00%	0.03%			0.00%	-0.19%
		3	0.4223256180	840.514			0.00%	0.00%			0.00%	-0.22%
		4	0.4223256180	841.052			0.00%	0.06%			0.00%	-0.16%
		5	0.4223256180	847.511			0.00%	0.83%			0.00%	0.61%

Full results of Hybrid Variant 1 (cont)

					best		error	speed	mean		error	speed
dataset	cardinality	time (n)	TE	seconds	TE	seconds			TE	seconds		
indtrack7	50	1	0.7697716141	1298.58	0.7614320007	1503.114	1.10%	-13.61%	0.7681036914	1341.8278	0.22%	-3.22%
		2	0.7697716141	1302.459			1.10%	-13.35%			0.22%	-2.93%
		3	0.7697716141	1302.566			1.10%	-13.34%			0.22%	-2.93%
		4	0.7614320007	1503.114			0.00%	0.00%			-0.87%	12.02%
		5	0.7697716141	1302.42			1.10%	-13.35%			0.22%	-2.94%
	75	1	0.5051825509	901.966	0.4927349473	1144.066	2.53%	-21.16%	0.5083534229	1030.4344	-0.62%	-12.47%
		2	0.5233793614	1302.512			6.22%	13.85%			2.96%	26.40%
		3	0.5152877038	901.777			4.58%	-21.18%			1.36%	-12.49%
		4	0.5051825509	901.851			2.53%	-21.17%			-0.62%	-12.48%
		5	0.4927349473	1144.066			0.00%	0.00%			-3.07%	11.03%
	100	1	0.3760497253	918.28	0.3590565612	1333.96	4.73%	-31.16%	0.3726510925	1001.3812	0.91%	-8.30%
		2	0.3760497253	918.12			4.73%	-31.17%			0.91%	-8.31%
		3	0.3590565612	1333.96			0.00%	0.00%			-3.65%	33.21%
		4	0.3760497253	918.163			4.73%	-31.17%			0.91%	-8.31%
		5	0.3760497253	918.383			4.73%	-31.15%			0.91%	-8.29%
indtrack8	50	1	0.5510091358	903.172	0.5219144736	1484.288	5.57%	-39.15%	0.5451902034	1019.5316	1.07%	-11.41%
		2	0.5219144736	1484.288			0.00%	0.00%			-4.27%	45.59%
		3	0.5510091358	903.447			5.57%	-39.13%			1.07%	-11.39%
		4	0.5510091358	903.462			5.57%	-39.13%			1.07%	-11.38%
		5	0.5510091358	903.289			5.57%	-39.14%			1.07%	-11.40%
	75	1	0.3654938924	903.214	0.3654938924	903.214	0.00%	0.00%	0.3654938924	903.2956	0.00%	-0.01%
		2	0.3654938924	903.313			0.00%	0.01%			0.00%	0.00%
		3	0.3654938924	903.353			0.00%	0.02%			0.00%	0.01%
		4	0.3654938924	903.323			0.00%	0.01%			0.00%	0.00%
		5	0.3654938924	903.275			0.00%	0.01%			0.00%	0.00%
	100	1	0.2671109300	1989.575	0.2671109300	1587.021	0.00%	25.37%	0.2671109300	1749.182	0.00%	13.74%
		2	0.2671109300	1592.906			0.00%	0.37%			0.00%	-8.93%
		3	0.2671109300	1989.376			0.00%	25.35%			0.00%	13.73%
		4	0.2671109300	1587.021			0.00%	0.00%			0.00%	-9.27%
		5	0.2671109300	1587.032			0.00%	0.00%			0.00%	-9.27%

Full results of Hybrid Variant 2

					best found		error	speed	mean		error	speed
dataset	cardinality	time (n)	TE	seconds	TE	seconds			TE	seconds		
indtrack3	30	1	0.4892474495	602.124	0.4892474495	600.982	0.00%	0.19%	0.4892474495	601.7504	0.00%	0.06%
		2	0.4892474495	602.655			0.00%	0.28%			0.00%	0.15%
		3	0.4892474495	601.885			0.00%	0.15%			0.00%	0.02%
		4	0.4892474495	601.106			0.00%	0.02%			0.00%	-0.11%
		5	0.4892474495	600.982			0.00%	0.00%			0.00%	-0.13%
	35	1	0.4319250954	715.689	0.4319250954	715.127	0.00%	0.08%	0.4319250954	715.38	0.00%	0.04%
		2	0.4319250954	715.282			0.00%	0.02%			0.00%	-0.01%
		3	0.4319250954	715.127			0.00%	0.00%			0.00%	-0.04%
		4	0.4319250954	715.55			0.00%	0.06%			0.00%	0.02%
		5	0.4319250954	715.252			0.00%	0.02%			0.00%	-0.02%
	40	1	0.3804245312	677.647	0.3804245312	661.247	0.00%	2.48%	0.3804245312	667.7252	0.00%	1.49%
		2	0.3804245312	661.247			0.00%	0.00%			0.00%	-0.97%
		3	0.3804245312	661.339			0.00%	0.01%			0.00%	-0.96%
		4	0.3804245312	661.682			0.00%	0.07%			0.00%	-0.91%
		5	0.3804245312	676.711			0.00%	2.34%			0.00%	1.35%
indtrack4	30	1	0.4327004895	528.986	0.4327004895	528.986	0.00%	0.00%	0.4327004895	533.1558	0.00%	-0.78%
		2	0.4327004895	549.028			0.00%	3.79%			0.00%	2.98%
		3	0.4327004895	529.197			0.00%	0.04%			0.00%	-0.74%
		4	0.4327004895	529.154			0.00%	0.03%			0.00%	-0.75%
		5	0.4327004895	529.414			0.00%	0.08%			0.00%	-0.70%
	35	1	0.3752295070	613.476	0.3752295070	613.476	0.00%	0.00%	0.3828553817	632.9588	-1.99%	-3.08%
		2	0.3847618504	637.855			2.54%	3.97%			0.50%	0.77%
		3	0.3847618504	637.815			2.54%	3.97%			0.50%	0.77%
		4	0.3847618504	637.822			2.54%	3.97%			0.50%	0.77%
		5	0.3847618504	637.826			2.54%	3.97%			0.50%	0.77%
	40	1	0.3449795047	670.547	0.3449795047	670.527	0.00%	0.00%	0.3449795047	670.6842	0.00%	-0.02%
		2	0.3449795047	670.987			0.00%	0.07%			0.00%	0.05%
		3	0.3449795047	670.705			0.00%	0.03%			0.00%	0.00%
		4	0.3449795047	670.655			0.00%	0.02%			0.00%	0.00%
		5	0.3449795047	670.527			0.00%	0.00%			0.00%	-0.02%

Full results of Hybrid Variant 2 (cont)

					best found		error	speed	mean		error	speed
data set	cardinality	time (n)	TE	seconds	TE	seconds			TE	seconds		
indtrack5	50	1	0.3273143445	900.824	0.3273143445	900.703	0.00%	0.01%	0.3262348469	907.393	0.33%	-0.72%
		2	0.3273143445	900.703			0.00%	0.00%			0.33%	-0.74%
		3	0.3246156005	917.271			-0.82%	1.84%			-0.50%	1.09%
		4	0.3273143445	900.753			0.00%	0.01%			0.33%	-0.73%
		5	0.3246156005	917.414			-0.82%	1.86%			-0.50%	1.10%
	75	1	0.2314182321	1081.753	0.2314182321	1078.839	0.00%	0.27%	0.2314182321	1080.1916	0.00%	0.14%
		2	0.2314182321	1079.887			0.00%	0.10%			0.00%	-0.03%
		3	0.2314182321	1080.814			0.00%	0.18%			0.00%	0.06%
		4	0.2314182321	1079.665			0.00%	0.08%			0.00%	-0.05%
		5	0.2314182321	1078.839			0.00%	0.00%			0.00%	-0.13%
	100	1	0.1953191901	1097.172	0.1913276648	1843.689	2.09%	-40.49%	0.1941128342	1169.1264	0.62%	-6.15%
		2	0.1953192140	1069.471			2.09%	-41.99%			0.62%	-8.52%
		3	0.1951870155	919.071			2.02%	-50.15%			0.55%	-21.39%
		4	0.1934110865	916.229			1.09%	-50.30%			-0.36%	-21.63%
		5	0.1913276648	1843.689			0.00%	0.00%			-1.43%	57.70%
indtrack6	50	1	0.5511654938	888.071	0.5511654938	886.689	0.00%	0.16%	0.5511654938	887.5808	0.00%	0.06%
		2	0.5511654938	887.062			0.00%	0.04%			0.00%	-0.06%
		3	0.5511654938	887.769			0.00%	0.12%			0.00%	0.02%
		4	0.5511654938	886.689			0.00%	0.00%			0.00%	-0.10%
		5	0.5511654938	888.313			0.00%	0.18%			0.00%	0.08%
	75	1	0.4495350040	1016.309	0.4495350040	1006.345	0.00%	0.99%	0.4495350040	1009.0958	0.00%	0.71%
		2	0.4495350040	1007.456			0.00%	0.11%			0.00%	-0.16%
		3	0.4495350040	1008.256			0.00%	0.19%			0.00%	-0.08%
		4	0.4495350040	1007.113			0.00%	0.08%			0.00%	-0.20%
		5	0.4495350040	1006.345			0.00%	0.00%			0.00%	-0.27%
	100	1	0.4223256180	842.098	0.4223256180	826.364	0.00%	1.90%	0.4223256180	838.773	0.00%	0.40%
		2	0.4223256180	841.556			0.00%	1.84%			0.00%	0.33%
		3	0.4223256180	841.636			0.00%	1.85%			0.00%	0.34%
		4	0.4223256180	842.211			0.00%	1.92%			0.00%	0.41%
		5	0.4223256180	826.364			0.00%	0.00%			0.00%	-1.48%

Full results of Hybrid Variant 2 (cont)

					best found		error	speed	mean		error	speed
data set	cardinality	time (n)	TE	seconds	TE	seconds			TE	seconds		
indtrack7	50	1	0.7730785858	1302.655	0.7252269332	1582.103	6.60%	-17.66%	0.7712627347	1118.1138	0.24%	16.50%
		2	0.7860027182	901.94			8.38%	-42.99%			1.91%	-19.33%
		3	0.7860027182	901.93			8.38%	-42.99%			1.91%	-19.33%
		4	0.7252269332	1582.103			0.00%	0.00%			-5.97%	41.50%
		5	0.7860027182	901.941			8.38%	-42.99%			1.91%	-19.33%
	75	1	0.5051825509	902.05	0.5051825509	901.939	0.00%	0.01%	0.5096439604	901.9404	-0.88%	0.01%
		2	0.5051825509	901.939			0.00%	0.00%			-0.88%	0.00%
		3	0.5173844457	901.665			2.42%	-0.03%			1.52%	-0.03%
		4	0.5152877038	902.105			2.00%	0.02%			1.11%	0.02%
		5	0.5051825509	901.943			0.00%	0.00%			-0.88%	0.00%
	100	1	0.3818572993	1361.567	0.3818572993	1361.567	0.00%	0.00%	0.4005325688	1074.9484	-4.66%	26.66%
		2	0.4060793577	905.708			6.34%	-33.48%			1.38%	-15.74%
		3	0.4043234147	1102.742			5.88%	-19.01%			0.95%	2.59%
		4	0.4060793577	902			6.34%	-33.75%			1.38%	-16.09%
		5	0.4043234147	1102.725			5.88%	-19.01%			0.95%	2.58%
indtrack8	50	1	0.5258619602	1502.932	0.5258619602	1502.932	0.00%	0.00%	0.5268984910	1344.2708	-0.20%	11.80%
		2	0.5271576237	1304.588			0.25%	-13.20%			0.05%	-2.95%
		3	0.5271576237	1304.611			0.25%	-13.20%			0.05%	-2.95%
		4	0.5271576237	1304.625			0.25%	-13.19%			0.05%	-2.95%
		5	0.5271576237	1304.598			0.25%	-13.20%			0.05%	-2.95%
	75	1	0.3654938924	903.595	0.3654938924	903.595	0.00%	0.00%	0.3654938924	903.6934	0.00%	-0.01%
		2	0.3654938924	903.682			0.00%	0.01%			0.00%	0.00%
		3	0.3654938924	903.787			0.00%	0.02%			0.00%	0.01%
		4	0.3654938924	903.693			0.00%	0.01%			0.00%	0.00%
		5	0.3654938924	903.71			0.00%	0.01%			0.00%	0.00%
	100	1	0.2959789679	903.58	0.2959789679	903.412	0.00%	0.02%	0.2959789679	903.5018	0.00%	0.01%
		2	0.2959789679	903.454			0.00%	0.00%			0.00%	-0.01%
		3	0.2959789679	903.572			0.00%	0.02%			0.00%	0.01%
		4	0.2959789679	903.412			0.00%	0.00%			0.00%	-0.01%
		5	0.2959789679	903.491			0.00%	0.01%			0.00%	0.00%

Full results of CPLEX solver

dataset	cardinality	time (n)	TE	seconds	best TE	best seconds	error	speed	mean TE	mean seconds	error	speed
indtrack3	30	1	0.4817540665	679.126	0.4817540665	679.126	0.00%	0.00%	0.4817540665	679.8148	0.00%	-0.10%
		2	0.4817540665	680.534			0.00%	0.21%			0.00%	0.11%
		3	0.4817540665	680.192			0.00%	0.16%			0.00%	0.06%
		4	0.4817540665	679.569			0.00%	0.07%			0.00%	-0.04%
		5	0.4817540665	679.653			0.00%	0.08%			0.00%	-0.02%
	35	1	0.4271480572	899.978	0.4271480572	608.289	0.00%	47.95%	0.4286884345	904.2884	-0.36%	-0.48%
		2	0.4307490080	1208.62			0.84%	98.69%			0.48%	33.65%
		3	0.4271480572	608.289			0.00%	0.00%			-0.36%	-32.73%
		4	0.4299569129	903.472			0.66%	48.53%			0.30%	-0.09%
		5	0.4284401372	901.083			0.30%	48.13%			-0.06%	-0.35%
	40	1	0.3804245312	654.584	0.3780275727	655.099	0.63%	-0.08%	0.3799451395	654.7428	0.13%	-0.02%
		2	0.3804245312	654.486			0.63%	-0.09%			0.13%	-0.04%
		3	0.3780275727	655.099			0.00%	0.00%			-0.50%	0.05%
		4	0.3804245312	654.709			0.63%	-0.06%			0.13%	-0.01%
		5	0.3804245312	654.836			0.63%	-0.04%			0.13%	0.01%
indtrack4	30	1	0.4384358902	639.758	0.4293869816	639.348	2.11%	0.06%	0.4331922668	639.4724	1.21%	0.04%
		2	0.4337173213	639.222			1.01%	-0.02%			0.12%	-0.04%
		3	0.4293869816	639.348			0.00%	0.00%			-0.88%	-0.02%
		4	0.4337173213	639.25			1.01%	-0.02%			0.12%	-0.03%
		5	0.4307038194	639.784			0.31%	0.07%			-0.57%	0.05%
	35	1	0.3874766074	661.483	0.3874766074	658.922	0.00%	0.39%	0.3874766074	659.695	0.00%	0.27%
		2	0.3874766074	659.227			0.00%	0.05%			0.00%	-0.07%
		3	0.3874766074	659.296			0.00%	0.06%			0.00%	-0.06%
		4	0.3874766074	658.922			0.00%	0.00%			0.00%	-0.12%
		5	0.3874766074	659.547			0.00%	0.09%			0.00%	-0.02%
	40	1	0.3519302829	1249.166	0.3519302829	1248.879	0.00%	0.02%	0.3520122849	888.5724	-0.02%	40.58%
		2	0.3520669529	648.468			0.04%	-48.08%			0.02%	-27.02%
		3	0.3520669529	648.066			0.04%	-48.11%			0.02%	-27.07%
		4	0.3520669529	648.283			0.04%	-48.09%			0.02%	-27.04%
		5	0.3519302829	1248.879			0.00%	0.00%			-0.02%	40.55%

Full results of CPLEX solver (cont)

					best		error	speed	mean		error	speed
dataset	cardinality	time (n)	TE	seconds	TE	seconds			TE	seconds		
indtrack5	50	1	0.3364703323	1099.809	0.3364703323	1099.05	0.00%	0.07%	0.3378335668	1099.4732	-0.40%	0.03%
		2	0.3432865047	1100.124			2.03%	0.10%			1.61%	0.06%
		3	0.3364703323	1099.236			0.00%	0.02%			-0.40%	-0.02%
		4	0.3364703323	1099.147			0.00%	0.01%			-0.40%	-0.03%
		5	0.3364703323	1099.05			0.00%	0.00%			-0.40%	-0.04%
	75	1	0.2392183371	1035.608	0.2332214418	1636.469	2.57%	-36.72%	0.2373213146	1276.2558	0.80%	-18.86%
		2	0.2392183371	1036.656			2.57%	-36.65%			0.80%	-18.77%
		3	0.2357301199	1636.807			1.08%	0.02%			-0.67%	28.25%
		4	0.2392183371	1035.739			2.57%	-36.71%			0.80%	-18.85%
		5	0.2332214418	1636.469			0.00%	0.00%			-1.73%	28.22%
	100	1	0.1928306749	1924.459	0.1928306749	1924.459	0.00%	0.00%	0.1941520556	1143.641	-0.68%	68.27%
		2	0.1951126599	723.11			1.18%	-62.43%			0.49%	-36.77%
		3	0.1928966252	1323.968			0.03%	-31.20%			-0.65%	15.77%
		4	0.1949601590	1023.655			1.10%	-46.81%			0.42%	-10.49%
		5	0.1949601590	723.013			1.10%	-62.43%			0.42%	-36.78%
indtrack6	50	1	0.5616636628	1740.242	0.5616636628	1740.073	0.00%	0.01%	0.5616636628	1740.5826	0.00%	-0.02%
		2	0.5616636628	1740.073			0.00%	0.00%			0.00%	-0.03%
		3	0.5616636628	1741.102			0.00%	0.06%			0.00%	0.03%
		4	0.5616636628	1741.096			0.00%	0.06%			0.00%	0.03%
		5	0.5616636628	1740.4			0.00%	0.02%			0.00%	-0.01%
	75	1	0.4502628404	1390.309	0.4479601694	2291.379	0.51%	-39.32%	0.4498023062	1571.0194	0.10%	-11.50%
		2	0.4502628404	1392.202			0.51%	-39.24%			0.10%	-11.38%
		3	0.4479601694	2291.379			0.00%	0.00%			-0.41%	45.85%
		4	0.4502628404	1391.057			0.51%	-39.29%			0.10%	-11.46%
		5	0.4502628404	1390.15			0.51%	-39.33%			0.10%	-11.51%
	100	1	0.4241671005	481.532	0.4241671005	480.108	0.00%	0.30%	0.4241671005	480.4504	0.00%	0.23%
		2	0.4241671005	480.108			0.00%	0.00%			0.00%	-0.07%
		3	0.4241671005	480.117			0.00%	0.00%			0.00%	-0.07%
		4	0.4241671005	480.342			0.00%	0.05%			0.00%	-0.02%
		5	0.4241671005	480.153			0.00%	0.01%			0.00%	-0.06%

Full results of CPLEX solver (cont)

					best		error	speed	mean		error	speed
dataset	cardinality	time (n)	TE	seconds	TE	seconds			TE	seconds		
indtrack7	50	1	1.0546479030	900.787	1.0546479030	900.717	0.00%	0.01%	1.0546479030	900.7666	0.00%	0.00%
		2	1.0546479030	900.717			0.00%	0.00%			0.00%	-0.01%
		3	1.0546479030	900.79			0.00%	0.01%			0.00%	0.00%
		4	1.0546479030	900.809			0.00%	0.01%			0.00%	0.00%
		5	1.0546479030	900.73			0.00%	0.00%			0.00%	0.00%
	75	1	0.6861147431	1200.932	0.6861147431	1200.932	0.00%	0.00%	0.7359533412	960.7924	-6.77%	24.99%
		2	0.7484129907	900.77			9.08%	-24.99%			1.69%	-6.25%
		3	0.7484129907	900.865			9.08%	-24.99%			1.69%	-6.24%
		4	0.7484129907	900.638			9.08%	-25.01%			1.69%	-6.26%
		5	0.7484129907	900.757			9.08%	-25.00%			1.69%	-6.25%
	100	1	0.4204524529	900.684	0.4204524529	900.51	0.00%	0.02%	0.4204524529	900.746	0.00%	-0.01%
		2	0.4204524529	900.682			0.00%	0.02%			0.00%	-0.01%
		3	0.4204524529	900.51			0.00%	0.00%			0.00%	-0.03%
		4	0.4204524529	901.093			0.00%	0.06%			0.00%	0.04%
		5	0.4204524529	900.761			0.00%	0.03%			0.00%	0.00%
indtrack8	50	1	0.7314498723	900.886	0.7314498723	900.847	0.00%	0.00%	0.7314498723	900.8882	0.00%	0.00%
		2	0.7314498723	900.89			0.00%	0.00%			0.00%	0.00%
		3	0.7314498723	900.923			0.00%	0.01%			0.00%	0.00%
		4	0.7314498723	900.847			0.00%	0.00%			0.00%	0.00%
		5	0.7314498723	900.895			0.00%	0.01%			0.00%	0.00%
	75	1	0.4285623453	600.712	0.4285623453	600.695	0.00%	0.00%	0.4285623453	600.842	0.00%	-0.02%
		2	0.4285623453	600.699			0.00%	0.00%			0.00%	-0.02%
		3	0.4285623453	600.712			0.00%	0.00%			0.00%	-0.02%
		4	0.4285623453	600.695			0.00%	0.00%			0.00%	-0.02%
		5	0.4285623453	601.392			0.00%	0.12%			0.00%	0.09%
	100	1	0.3088583478	600.582	0.3088583478	600.582	0.00%	0.00%	0.3088583478	600.6108	0.00%	0.00%
		2	0.3088583478	600.621			0.00%	0.01%			0.00%	0.00%
		3	0.3088583478	600.615			0.00%	0.01%			0.00%	0.00%
		4	0.3088583478	600.624			0.00%	0.01%			0.00%	0.00%
		5	0.3088583478	600.612			0.00%	0.00%			0.00%	0.00%

Full results of CPLEX solver (2000 time limit)

					best		error	speed	mean		error	speed
dataset	cardinality	time (n)	TE	seconds	TE	seconds			TE	seconds		
indtrack3	30	1	0.4817540665	4088.126	0.4817540665	4081.7	0.00%	0.16%	0.4817540665	4083.0522	0.00%	0.12%
		2	0.4817540665	4081.801			0.00%	0.00%			0.00%	-0.03%
		3	0.4817540665	4081.786			0.00%	0.00%			0.00%	-0.03%
		4	0.4817540665	4081.848			0.00%	0.00%			0.00%	-0.03%
		5	0.4817540665	4081.7			0.00%	0.00%			0.00%	-0.03%
	35	1	0.4271480572	4010.589	0.4271480572	4010.429	0.00%	0.00%	0.4288284607	4372.5212	-0.39%	-8.28%
		2	0.4307304420	3819.137			0.84%	-4.77%			0.44%	-12.66%
		3	0.4271480572	4010.429			0.00%	0.00%			-0.39%	-8.28%
		4	0.4271480572	6011.956			0.00%	49.91%			-0.39%	37.49%
		5	0.4319676901	4010.495			1.13%	0.00%			0.73%	-8.28%
	40	1	0.3780275727	4057.55	0.3780275727	4057.55	0.00%	0.00%	0.3799451395	4057.2168	-0.50%	0.01%
		2	0.3804245312	4057.193			0.63%	-0.01%			0.13%	0.00%
		3	0.3804245312	4056.983			0.63%	-0.01%			0.13%	-0.01%
		4	0.3804245312	4057.228			0.63%	-0.01%			0.13%	0.00%
		5	0.3804245312	4057.13			0.63%	-0.01%			0.13%	0.00%
indtrack4	30	1	0.4384358902	4042.023	0.4293869816	4041.764	2.11%	0.01%	0.4338427148	4041.9346	1.06%	0.00%
		2	0.4365712738	4042.025			1.67%	0.01%			0.63%	0.00%
		3	0.4316850168	4041.987			0.54%	0.01%			-0.50%	0.00%
		4	0.4293869816	4041.764			0.00%	0.00%			-1.03%	0.00%
		5	0.4331344114	4041.874			0.87%	0.00%			-0.16%	0.00%
	35	1	0.3874766074	4068.638	0.3874766074	4061.598	0.00%	0.17%	0.3874766074	4063.2372	0.00%	0.13%
		2	0.3874766074	4062.045			0.00%	0.01%			0.00%	-0.03%
		3	0.3874766074	4062.151			0.00%	0.01%			0.00%	-0.03%
		4	0.3874766074	4061.598			0.00%	0.00%			0.00%	-0.04%
		5	0.3874766074	4061.754			0.00%	0.00%			0.00%	-0.04%
	40	1	0.3519302829	6052.486	0.3519302829	6052.248	0.00%	0.00%	0.3519576169	6453.2968	-0.01%	-6.21%
		2	0.3520669529	4050.43			0.04%	-33.08%			0.03%	-37.23%
		3	0.3519302829	6052.248			0.00%	0.00%			-0.01%	-6.21%
		4	0.3519302829	8057.117			0.00%	33.13%			-0.01%	24.85%
		5	0.3519302829	8054.203			0.00%	33.08%			-0.01%	24.81%

Full results of CPLEX solver (2000 time limit) (cont)

					best		error	speed	mean		error	speed
dataset	cardinality	time (n)	TE	seconds	TE	seconds			TE	seconds		
indtrack5	50	1	0.3262242002	6037.928	0.3262242002	6037.928	0.00%	0.00%	0.3299795845	6562.5776	-1.14%	-7.99%
		2	0.3358173832	6209.233			2.94%	2.84%			1.77%	-5.38%
		3	0.3275122720	8184.286			0.39%	35.55%			-0.75%	24.71%
		4	0.3301720335	6209.279			1.21%	2.84%			0.06%	-5.38%
		5	0.3301720335	6172.162			1.21%	2.22%			0.06%	-5.95%
	75	1	0.2361913582	4144.317	0.2303772115	6018.324	2.52%	-31.14%	0.2355017966	5670.2916	0.29%	-26.91%
		2	0.2355475211	5996.649			2.24%	-0.36%			0.02%	5.76%
		3	0.2303772115	6018.324			0.00%	0.00%			-2.18%	6.14%
		4	0.2361745552	8053.79			2.52%	33.82%			0.29%	42.03%
		5	0.2392183371	4138.378			3.84%	-31.24%			1.58%	-27.02%
	100	1	0.1933674189	6040.9	0.1913976229	6052.919	1.03%	-0.20%	0.1928520757	6423.068	0.27%	-5.95%
		2	0.1932859769	5992.46			0.99%	-1.00%			0.22%	-6.70%
		3	0.1930081938	8071.2			0.84%	33.34%			0.08%	25.66%
		4	0.1913976229	6052.919			0.00%	0.00%			-0.75%	-5.76%
		5	0.1932011661	5957.861			0.94%	-1.57%			0.18%	-7.24%
indtrack6	50	1	0.5800727299	4140.453	0.5535545988	14250.481	4.79%	-70.95%	0.5611861660	8552.7564	3.37%	-51.59%
		2	0.5535545988	14250.481			0.00%	0.00%			-1.36%	66.62%
		3	0.5554091199	6081.436			0.34%	-57.32%			-1.03%	-28.90%
		4	0.5631437585	8187.802			1.73%	-42.54%			0.35%	-4.27%
		5	0.5537506228	10103.61			0.04%	-29.10%			-1.32%	18.13%
	75	1	0.4480971056	8198.638	0.4480971056	8198.638	0.00%	0.00%	0.4504525637	7797.4722	-0.52%	5.14%
		2	0.4528131606	6196.514			1.05%	-24.42%			0.52%	-20.53%
		3	0.4508800938	4193.152			0.62%	-48.86%			0.09%	-46.22%
		4	0.4487814937	12201.735			0.15%	48.83%			-0.37%	56.48%
		5	0.4516909649	8197.322			0.80%	-0.02%			0.27%	5.13%
	100	1	0.4241671005	2182.583	0.4241671005	2181.703	0.00%	0.04%	0.4241671005	2182.1914	0.00%	0.02%
		2	0.4241671005	2181.703			0.00%	0.00%			0.00%	-0.02%
		3	0.4241671005	2182.296			0.00%	0.03%			0.00%	0.00%
		4	0.4241671005	2182.173			0.00%	0.02%			0.00%	0.00%
		5	0.4241671005	2182.202			0.00%	0.02%			0.00%	0.00%

Full results of CPLEX solver (2000 time limit) (cont)

					best		error	speed	mean		error	speed
dataset	cardinality	time (n)	TE	seconds	TE	seconds			TE	seconds		
indtrack7	50	1	0.8348347767	15812.003	0.7887382371	21828.03	5.84%	-27.56%	0.8360679215	14206.7174	-0.15%	11.30%
		2	0.8690301543	7791.06			10.18%	-64.31%			3.94%	-45.16%
		3	0.8305198688	15802.474			5.30%	-27.60%			-0.66%	11.23%
		4	0.8572165704	9800.02			8.68%	-55.10%			2.53%	-31.02%
		5	0.7887382371	21828.03			0.00%	0.00%			-5.66%	53.65%
	75	1	0.4922419658	31836.885	0.4922419658	31836.885	0.00%	0.00%	0.5044398222	24204.046	-2.42%	31.54%
		2	0.5326039350	13761.298			8.20%	-56.78%			5.58%	-43.14%
		3	0.5081437705	19792.285			3.23%	-37.83%			0.73%	-18.23%
		4	0.4934420080	27814.186			0.24%	-12.64%			-2.18%	14.92%
		5	0.4957674316	27815.576			0.72%	-12.63%			-1.72%	14.92%
	100	1	0.3656001852	18395.292	0.3629090012	22418.574	0.74%	-17.95%	0.3642397247	22006.1488	0.37%	-16.41%
		2	0.3635149517	22400.927			0.17%	-0.08%			-0.20%	1.79%
		3	0.3629090012	22418.574			0.00%	0.00%			-0.37%	1.87%
		4	0.3655342920	24421.952			0.72%	8.94%			0.36%	10.98%
		5	0.3636401935	22393.999			0.20%	-0.11%			-0.16%	1.76%
indtrack8	50	1	0.5094721331	28016.323	0.5094721331	28016.323	0.00%	0.00%	0.5314499795	22413.7172	-4.14%	25.00%
		2	0.5374485779	16009.389			5.49%	-42.86%			1.13%	-28.57%
		3	0.5471905919	22012.441			7.40%	-21.43%			2.96%	-1.79%
		4	0.5502560210	22012.586			8.01%	-21.43%			3.54%	-1.79%
		5	0.5128825737	24017.847			0.67%	-14.27%			-3.49%	7.16%
	75	1	0.3484761695	8982.795	0.3255922828	20994.889	7.03%	-57.21%	0.3438993922	11386.815	1.33%	-21.11%
		2	0.3484761695	8988.997			7.03%	-57.18%			1.33%	-21.06%
		3	0.3255922828	20994.889			0.00%	0.00%			-5.32%	84.38%
		4	0.3484761695	8986.46			7.03%	-57.20%			1.33%	-21.08%
		5	0.3484761695	8980.934			7.03%	-57.22%			1.33%	-21.13%
	100	1	0.2638826793	8568.2	0.2528709111	20587.671	4.35%	-58.38%	0.2596258929	11771.5206	1.64%	-27.21%
		2	0.2528709111	20587.671			0.00%	0.00%			-2.60%	74.89%
		3	0.2587465974	10568.162			2.32%	-48.67%			-0.34%	-10.22%
		4	0.2587465974	10568.401			2.32%	-48.67%			-0.34%	-10.22%
		5	0.2638826793	8565.169			4.35%	-58.40%			1.64%	-27.24%

Full results of GA

					best		error	speed	mean		error	speed
dataset	cardinality	time (n)	TE	seconds	TE	seconds			TE	seconds		
indtrack3	30	1	0.4827071149	2678.164	0.4817540665	2754.086	0.20%	-2.76%	0.4848047982	2508.5464	-0.43%	6.76%
		2	0.4900292799	2394.599			1.72%	-13.05%			1.08%	-4.54%
		3	0.4847667649	2363.902			0.63%	-14.17%			-0.01%	-5.77%
		4	0.4847667649	2351.981			0.63%	-14.60%			-0.01%	-6.24%
		5	0.4817540665	2754.086			0.00%	0.00%			-0.63%	9.79%
	35	1	0.4298864841	1865.912	0.4298864841	1862.885	0.00%	0.16%	0.4298864841	1866.0688	0.00%	-0.01%
		2	0.4298864841	1867.026			0.00%	0.22%			0.00%	0.05%
		3	0.4298864841	1866.965			0.00%	0.22%			0.00%	0.05%
		4	0.4298864841	1867.556			0.00%	0.25%			0.00%	0.08%
		5	0.4298864841	1862.885			0.00%	0.00%			0.00%	-0.17%
	40	1	0.3874157560	2785.62	0.3869218700	2673.548	0.13%	4.19%	0.3873169788	2780.5576	0.03%	0.18%
		2	0.3869218700	2673.548			0.00%	0.00%			-0.10%	-3.85%
		3	0.3874157560	2817.714			0.13%	5.39%			0.03%	1.34%
		4	0.3874157560	2816.173			0.13%	5.33%			0.03%	1.28%
		5	0.3874157560	2809.733			0.13%	5.09%			0.03%	1.05%
indtrack4	30	1	0.4307038194	2018.706	0.4307038194	2016.925	0.00%	0.09%	0.4307038194	2051.24	0.00%	-1.59%
		2	0.4307038194	2129.487			0.00%	5.58%			0.00%	3.81%
		3	0.4307038194	2068.525			0.00%	2.56%			0.00%	0.84%
		4	0.4307038194	2016.925			0.00%	0.00%			0.00%	-1.67%
		5	0.4307038194	2022.557			0.00%	0.28%			0.00%	-1.40%
	35	1	0.3739008486	2347.229	0.3739008486	2331.39	0.00%	0.68%	0.3739008486	2347.869	0.00%	-0.03%
		2	0.3739008486	2334.5			0.00%	0.13%			0.00%	-0.57%
		3	0.3739008486	2348.307			0.00%	0.73%			0.00%	0.02%
		4	0.3739008486	2377.919			0.00%	2.00%			0.00%	1.28%
		5	0.3739008486	2331.39			0.00%	0.00%			0.00%	-0.70%
	40	1	0.3447495884	2715.349	0.3446569876	3722.255	0.03%	-27.05%	0.3449156089	3056.0724	-0.05%	-11.15%
		2	0.3450571562	3101.476			0.12%	-16.68%			0.04%	1.49%
		3	0.3450571562	2611.205			0.12%	-29.85%			0.04%	-14.56%
		4	0.3450571562	3130.077			0.12%	-15.91%			0.04%	2.42%
		5	0.3446569876	3722.255			0.00%	0.00%			-0.07%	21.80%

Full results of GA (cont)

					best				mean			
dataset	cardinality	time (n)	TE	seconds	TE	seconds	error	speed	TE	seconds	error	speed
indtrack5	50	1	0.3403977416	6784.186	0.3403977416	6774.716	0.00%	0.14%	0.3403977416	6779.3784	0.00%	0.07%
		2	0.3403977416	6783.362			0.00%	0.13%			0.00%	0.06%
		3	0.3403977416	6779.285			0.00%	0.07%			0.00%	0.00%
		4	0.3403977416	6774.716			0.00%	0.00%			0.00%	-0.07%
		5	0.3403977416	6775.343			0.00%	0.01%			0.00%	-0.06%
	75	1	0.2358116789	12073.042	0.2354881166	11472.411	0.14%	5.24%	0.2365005930	11796.7578	-0.29%	2.34%
		2	0.2359161390	12405.629			0.18%	8.13%			-0.25%	5.16%
		3	0.2354881166	11472.411			0.00%	0.00%			-0.43%	-2.75%
		4	0.2366700747	11684.921			0.50%	1.85%			0.07%	-0.95%
		5	0.2386169559	11347.786			1.33%	-1.09%			0.89%	-3.81%
	100	1	0.1934441783	10047.003	0.1921500400	11242.348	0.67%	-10.63%	0.1930179078	11073.6412	0.22%	-9.27%
		2	0.1933787607	11393.803			0.64%	1.35%			0.19%	2.89%
		3	0.1929032642	11258.798			0.39%	0.15%			-0.06%	1.67%
		4	0.1921500400	11242.348			0.00%	0.00%			-0.45%	1.52%
		5	0.1932132958	11426.254			0.55%	1.64%			0.10%	3.18%
indtrack6	50	1	0.5952057714	6040.738	0.5952057714	6026.084	0.00%	0.24%	0.5952057714	6032.3394	0.00%	0.14%
		2	0.5952057714	6034.62			0.00%	0.14%			0.00%	0.04%
		3	0.5952057714	6028.658			0.00%	0.04%			0.00%	-0.06%
		4	0.5952057714	6026.084			0.00%	0.00%			0.00%	-0.10%
		5	0.5952057714	6031.597			0.00%	0.09%			0.00%	-0.01%
	75	1	0.4544644840	8769.938	0.4517185186	8749.587	0.61%	0.23%	0.4537917612	8776.6182	0.15%	-0.08%
		2	0.4544644840	8793.128			0.61%	0.50%			0.15%	0.19%
		3	0.4544644840	8793.613			0.61%	0.50%			0.15%	0.19%
		4	0.4538468355	8776.825			0.47%	0.31%			0.01%	0.00%
		5	0.4517185186	8749.587			0.00%	0.00%			-0.46%	-0.31%
	100	1	0.4220146967	8724.308	0.4220146967	8724.308	0.00%	0.00%	0.4220146967	8733.4948	0.00%	-0.11%
		2	0.4220146967	8742.488			0.00%	0.21%			0.00%	0.10%
		3	0.4220146967	8730.724			0.00%	0.07%			0.00%	-0.03%
		4	0.4220146967	8731.788			0.00%	0.09%			0.00%	-0.02%
		5	0.4220146967	8738.166			0.00%	0.16%			0.00%	0.05%

Full results of GA (cont)

dataset	cardinality	time (n)	TE	seconds	best TE	best seconds	error	speed	mean TE	mean seconds	error	speed
indtrack7	50	1	0.8992144805	13516.844	0.8992144805	13511.784	0.00%	0.04%	0.8992144805	13512.8946	0.00%	0.03%
		2	0.8992144805	13512.101			0.00%	0.00%			0.00%	-0.01%
		3	0.8992144805	13511.936			0.00%	0.00%			0.00%	-0.01%
		4	0.8992144805	13511.784			0.00%	0.00%			0.00%	-0.01%
		5	0.8992144805	13511.808			0.00%	0.00%			0.00%	-0.01%
	75	1	0.7541368122	13515.27	0.6940551550	13512.061	8.66%	0.02%	0.7348028153	13513.6536	2.63%	0.01%
		2	0.7541368122	13511.892			8.66%	0.00%			2.63%	-0.01%
		3	0.6940551550	13512.061			0.00%	0.00%			-5.55%	-0.01%
		4	0.7175484851	13515.95			3.38%	0.03%			-2.35%	0.02%
		5	0.7541368122	13513.095			8.66%	0.01%			2.63%	0.00%
	100	1	0.4609632707	13511.462	0.4609632707	13511.154	0.00%	0.00%	0.4609632707	13511.5564	0.00%	0.00%
		2	0.4609632707	13511.154			0.00%	0.00%			0.00%	0.00%
		3	0.4609632707	13511.178			0.00%	0.00%			0.00%	0.00%
		4	0.4609632707	13511.514			0.00%	0.00%			0.00%	0.00%
		5	0.4609632707	13512.474			0.00%	0.01%			0.00%	0.01%
indtrack8	50	1	0.7314497887	13520.666	0.7314497887	13520.666	0.00%	0.00%	0.7314497887	13521.3644	0.00%	0.07%
		2	0.7314497887	13523.241			0.00%	0.02%			0.00%	0.09%
		3	0.7314497887	13521.021			0.00%	0.00%			0.00%	0.07%
		4	0.7314497887	13521.128			0.00%	0.00%			0.00%	0.07%
		5	0.7314497887	13520.766			0.00%	0.00%			0.00%	0.07%
	75	1	0.4285623453	13522.899	0.4285623453	13522.5	0.00%	0.00%	0.4285623453	13522.7598	0.00%	0.00%
		2	0.4285623453	13522.536			0.00%	0.00%			0.00%	0.00%
		3	0.4285623453	13523.213			0.00%	0.01%			0.00%	0.00%
		4	0.4285623453	13522.5			0.00%	0.00%			0.00%	0.00%
		5	0.4285623453	13522.651			0.00%	0.00%			0.00%	0.00%
	100	1	0.3081465555	13516.216	0.3081465555	13516.216	0.00%	0.00%	0.3081465555	13517.6894	0.00%	-0.01%
		2	0.3081465555	13521.627			0.00%	0.04%			0.00%	0.03%
		3	0.3081465555	13516.786			0.00%	0.00%			0.00%	-0.01%
		4	0.3081465555	13517.161			0.00%	0.01%			0.00%	0.00%
		5	0.3081465555	13516.657			0.00%	0.00%			0.00%	-0.01%