ROYAUME DU MAROC
Université Mohammed V
- RABAT -
جامعة محمد الخامس
– الرباط –
Faculté des sciences
كلية العلوم

CENTRE D'ETUDES DOCTORALES - SCIENCES ET TECHNOLOGIES

N° d'ordre **3132**

THESE

En vue de l'obtention du : ***DOCTORAT***

Structure de Recherche : Laboratoire de Recherche en Informatique et Télécommunications
Discipline : Sciences de l'ingénieur
Spécialité : Informatique

Présentée et soutenue le : 14/09/2018 par :

Mohamed EL YAFRANI

A study of problems with multiple interdependent components: understanding, modeling and algorithms

JURY

Abdelhakim AMEUR EL IMRANI	*PES,*	*Faculté des Sciences de Rabat,*	*Président*
Salma MOULINE	*PES,*	*Faculté des Sciences de Rabat,*	*Rapporteur*
Mohamed OUZINEB	*PH,*	*Institut National de Statistique et d'Economie Appliquée,*	*Examinateur*
Markus WAGNER	*PH,*	*University of Adelaide,*	*Rapporteur*
Myriam DELGADO	*PH,*	*Federal University of Technology of Paraná,*	*Examinateur*
Belaïd AHIOD	*PH,*	*Faculté des Sciences de Rabat,*	*Directeur de thèse*

Année Universitaire : 2017-2018

Acknowledgement

This thesis has been prepared within the laboratory of research in computer science and telecommunications (LRIT) at Mohammed V University under the direction and supervision of Pr. Belaïd Ahiod.

First, I would like to express my sincere gratitude to my thesis advisor Pr. Belaïd Ahiod, Professor of computer science at Mohamed V Univerisity, for introducing me to combinatorial optimisation, metaheuristics and to scientific research in general. He did not only teach me about optimisation, algorithms and writing papers, but most importantly, he taught me how to be an independent researcher and how to conduct ethical research.

I would like to express my gratitude to Pr. Abdelhakim Ameur El Imrani, Professor of computer science at Mohammed V University in Rabat for his feedback, assistance and for accepting to be the jury president for my defence.

My sincere appreciation and gratitude to Pr. Salma Mouline, Professor of computer science at Mohammed V University in Rabat for accepting to review my thesis and for her valuable comments and feedback.

I would like to thank Pr. Mohamed Ouzineb, Professor of computer science at the National Institute of Statistics and Applied Economics (INSEA) for accepting to evaluate my research and for his insightful comments.

Perhaps, what I enjoyed the most during these fours years of work is meeting other researchers sharing the same passion and collaborating with them. I would like to thank Dr Markus Wagner, Senior Lecturer at The University of Adelaide, for constantly supporting my research, inviting me to the University of Adelaide to conduct research together, accepting to attend my defence as a jury member and reviewing my dissertation.

Many thanks to Pr. Myriam Delgado, Professor at Graduate Program in Electrical and Computer Engineering at the Federal University of Technology of Paraná, for being among the jury members and for these fruitful years of collaboration, challenging questions and contributions.

I am also very grateful to Dr Mohammad Reza Bonyadi, Principal Advisor at the Data Science department at Rio Tinto, for assisting me many times during my PhD thesis. The third part of my thesis about supply chains is originally due to him and we are engaged to continue working together in the near future.

Many thanks to Pr. Ricardo Lüders, Professor of computer science at the Federal University of Technology of Paraná, for being part of my thesis and for his insights, feedback and contributions.

I am also very grateful to all my friends who have supported me during the last few years. Thank you Safaa, Marcella, Chaymaa, Asmae, Mehdi, Anass, Hicham, Mohamed, Nadir, Aziz, Yassine and Shelvin. Many thanks to all my colleagues and professors from AI_group and LRIT for their constant support and feedback.

I am also thankful to all the organisations who supported my research with grants :

— July 2018 : IEEE grant to attend WCCI 2018 in Rio de Janeiro, Brazil
— April 2017 : ACM grant to attend the "50 Years of the ACM Turing Award Celebration Conference" in San Francisco, California
— September 2016 : IEEE CIS Graduate Student Research Grant to visit the University of Adelaide
— May 2016 : ACM Travel grant to attend GECCO 2016 in Denver, Colorado
— October 2015 : QCRI Scholarship to attend AICCSA 2015 in Marrakech, Morocco

Finally, I would not have made it this far without the love and support of my family, and for that, I will be always grateful to them. A special thank you to my beloved cousins Karima, Hafida, Boumedien and Hossain.

Abstract

Many real-world optimisation problems are composed of multiple interacting sub-problems. However, few researchers look into tackling these problems using metaheuristics and evolutionary computation. In this thesis, we focus on studying problems with multiple interdependent sub-problems. To achieve our goal, we start by providing examples of these problems that are inspired from logistics and supply chain management. Then, we provide formal definitions of different types of dependencies in multi-component problems. Afterwards, two case studies are carried out. The first centers on the travelling thief problem, a benchmark model introduced to help researchers investigate the interdependence in real-world problems. The problem is introduced, analysed from different perspectives, and heuristic, metaheuristic, and hyperheuristic solutions are proposed to tackle it. The second case study is a problem inspired by real-world supply chain optimisation. We introduce the problem with the aim of proposing a more realistic benchmark problem that embeds a different class of dependency between components.

Keywords: Interdependence, Multi-component problems, Metaheuristics, Combinatorial optimisation.

Résumé

Plusieurs problèmes d'optimisation réels sont composés de multiples sous-problèmes en interaction. Cependant, peu de chercheurs se penchent sur la résolution de ces problèmes en utilisant les métaheuristiques et le calcul évolutionnaire. Dans cette thèse, nous nous concentrons sur l'étude de problèmes avec multiple sous-problèmes interdépendants. Pour atteindre notre objectif, nous commençons par proposer des problèmes inspirés par la logistique et la gestion de la chaîne d'approvisionnement. Ensuite, nous fournissons des définitions formelles des différents types de dépendances dans les problèmes multi-composants. Par la suite, deux études de cas sont réalisées. La première se focalise sur le problème du voyageur voleur, qui est un modèle de référence introduit pour aider les chercheurs à étudier l'interdépendance dans les problèmes du monde réel. Le problème est présenté, analysé sous différentes perspectives, et des solutions heuristiques, métaheuristiques et hyper-heuristiques sont proposées pour y faire face. La deuxième étude de cas est un problème inspiré de l'optimisation des chaînes d'approvisionnement réelles. Nous introduisons le problème dans le but de proposer un problème de référence plus réaliste qui illustre une classe différente de dépendance entre composantes.

Mots-clés : Interdépendance, Problèmes multi-composants, Métaheuristiques, Optimisation Combinatoire.

Résumé détaillé

Les chercheurs dans le domaine du calcul évolutionnaire et des métaheuristiques sont généralement intéressés par les problèmes de benchmark NP-complets. De tels problèmes comprennent le problème du voyageur de commerce, le problème de planification d'atelier et le problème du sac à dos parmi beaucoup d'autres. Résoudre ces problèmes à l'optimalité nécessite un temps exponentiel à moins que P=NP. les problèmes NP-complets sont d'une grande importance dans de nombreux domaines, y compris la logistique, l'ingénierie, la sécurité informatique et la biologie computationnelle. Cependant, certains chercheurs ont commencé à observer un écart important entre les problèmes de benchmark considérés dans la littérature et les problèmes d'optimisation réels, en particulier ceux provenant des applications industrielles.

Proposer des benchmarks de problèmes d'optimisation est une pratique très importante car elle permet de fournir une plate-forme publique aux chercheurs pour tester, analyser et comparer leurs algorithmes. Cependant, de nombreux aspects des problèmes d'optimisation réels restent négligés ou vaguement étudiés. Une liste non exhaustive de ces aspects est la suivante : Contraintes strictes et souples : Les contraintes strictes sont les contraintes que la solution doit satisfaire pour être implémentable (faisable). Un exemple est la limite de capacité maximale du sac-à-dos dans le problème du sac-à-dos. Les contraintes souples sont des contraintes qui sont préférables mais non requises pour être satisfaites. Un exemple est les préférences spécifiées par l'utilisateur dans les problèmes de planification. Dépendances externes : Cet aspect est lié à des événements se produisant en dehors du système sur lequel nous n'avons aucun contrôle direct. Le mieux que nous puissions faire est de les reconnaître et de les prendre en considération tout en optimisant les opérations à l'intérieur du système. Un exemple de tels événements peut être la concurrence avec d'autres systèmes, l'insuffisance inattendue de matières premières et la stochasticité du trafic lors de la livraison ou du transport de marchandise. Composition et dépendances internes : Cet aspect survient lorsque le système est composé de plusieurs sous-problèmes interactifs. Ici, nous supposons également que chaque sous-problème est un problème NP-complet lorsqu'il est isolé du système. Ces problèmes seront appelés Problèmes à multiples composantes interdépendantes (PMCI).

Dans cette thèse, nous allons nous concentrer sur le dernier aspect mentionné. Plus spécifiquement, nous souhaitons comprendre les dépendances entre les composants d'un PMCI, modéliser efficacement un PMCI, et aborder un PMCI en utilisant des méthodes

heuristiques, métaheuristiques et hyper-heuristiques.

La motivation derrière l'utilisation de métaheuristiques pour résoudre les problèmes multi-composants avec des dépendances est due à deux facteurs principaux. Le premier est le bon rapport la qualité des solutions et le temps d'exécution. En effet, dans de nombreux cas, il peut être préférable d'obtenir une «bonne» solution approximative dans un délai «équitable» plutôt que de consacrer du temps et des ressources pour générer une solution optimale. C'est particulièrement le cas dans les problèmes d'optimisation de la chaîne d'approvisionnement et d'ingénierie. Le deuxième facteur est la flexibilité de ces approches. En fait, en raison de leur définition abstraite, les métaheuristiques ont été appliquées avec succès à un grand nombre de problèmes d'optimisation réels.

Comme le suggère le titre de cette thèse, notre objectif principal est de comprendre, de modéliser et de concevoir des algorithmes pour des problèmes multi-composants avec des dépendances internes. Pour atteindre nos objectifs, nous proposons des définitions formelles des différents types de dépendance, proposons une classification des dépendances et étudions des cas particuliers de PMCI. Organisation du document et contributions

Première partie

Résumé

Reconnaître que les problèmes d'optimisation du monde réel ont plusieurs composantes interdépendantes peut être facile. Cependant, fournir un modèle générique et formel pour les dépendances entre composantes peut être une tâche difficile. En fait, un PMCI peut être considéré simplement comme un problème d'optimisation unique et les dépendances entre composantes pourraient être étudiées en examinant la décomposabilité du problème et les corrélations entre les sous-problèmes. Dans ce travail, nous tentons de définir un PMCI en raisonnant d'un point de vue inverse. Au lieu de considérer un problème décomposable, nous modélisons plusieurs problèmes (les composantes) et définissons comment ces composantes peuvent être connectées.

Dans la partie I, nous introduisons des notions liées aux problèmes avec des composants interdépendants. Nous commençons par introduire des exemples réalistes de la logistique et de la gestion de la chaîne d'approvisionnement pour illustrer la nature composite et les dépendances de ces problèmes dans le chapitre 1. Ensuite, dans le chapitre 2, nous fournissons notre proposition pour formaliser et classifier la dépendance dans les problèmes multi-composants.

Contributions

Dans le chapitre 2, nous proposons un modèle mathématique pour trois classes de dépendances :

- Dépendance de fitness : représente l'influence d'une composante sur une autre en changeant la fitness des solutions.
- Dépendance de faisabilité : représente l'influence d'un composant sur un autre en changeant l'ensemble des solutions réalisables.
- Dépendance temporelle : illustre les connexions entre composantes lorsque le processus d'optimisation d'une composante est limité dans une fenêtre temporelle et

que l'autre composante ne peut pas démarrer le processus d'optimisation avant que la première composante fournisse une solution partielle.

Deuxième partie

Résumé

Afin de réduire l'écart entre la recherche académique en optimisation et métaheuristiques qui se concentre sur les problèmes à composante unique, et les problèmes rencontrés dans l'industrie qui sont des problèmes à plusieurs composantes avec dépendance complexe, des chercheurs ont proposé un problème de benchmark nommé le problème du voyageur voleur (en anglais : Travelling Thief Problem ou TTP).

Dans la partie II, nous nous concentrons sur l'étude du TTP sous différents angles. Dans le chapitre 3, nous commençons par présenter formellement le TTP. Ensuite, nous analysons le problème en étudiant l'impact de la dépendance sur la complexité de la fonction objectif et sur le processus de recherche pour chaque composante. Ensuite, nous décrivons la base de donnée officielle des instances. Dans le chapitre 4, nous fournissons une analyse plus approfondie en utilisant les réseaux d'optima locaux, qui est un outil pour effectuer l'analyse du paysage de fitness (fitness landscape) sur les problèmes combinatoires. Dans le chapitre 5, nous introduisons des stratégies d'initialisation, des algorithmes de recherche locale et étudions un autre modèle pour le TTP basé sur l'optimisation multi-objectif. Ensuite, nous proposons des solutions heuristiques plus sophistiquées capables d'attaquer de très grandes instances TTP dans le chapitre 6. Enfin, dans le chapitre 7 nous proposons l'utilisation d'hyper-heuristiques pour concevoir automatiquement des algorithmes pour le TTP basés sur des heuristiques de bas niveau.

Contributions

L'étude du TTP a abouti à de multiples contributions sur l'analyse du problème et la conception de l'algorithme. Dans le chapitre 4, nous proposons l'utilisation d'une représentation compressée du paysage de fitness appelée réseaux d'optima locaux pour le TTP. Cette analyse nous a permis de tirer des conclusions importantes sur la nature de l'espace de recherche du TTP.

Dans le chapitre 5, nous proposons deux algorithmes de recherche locale basés sur la combinaison des voisinages des composants. Les expériences ont montré que ces heuristiques sont compétitives avec les algorithmes de l'état de l'art pour les instances de petite et moyenne taille, mais qu'elles ont des problèmes de s'adapter à des instances larges en raison de leur grande complexité de calcul. Néanmoins, ces approches sont importantes car elles abordent le TTP dans son ensemble, ce qui est une caractéristique importante qui s'est avérée utile pour réaliser l'analyse du paysage de fitness mentionnée ci-dessus.

En plus de la nature composite du TTP et de l'existence de dépendances, la base de donnée des instances de référence introduit de nombreuses instances à grande échelle avec des tailles allant jusqu'à 85900 villes et 858990 articles. Pour résoudre ce problème, nous présentons dans le chapitre 6 deux adaptations de métaheuristiques. La première est basée sur l'algorithme de recuit simulé et la recherche locale, et le deuxième est un algorithme mémétique. Ces approches ont été capables de surpasser les algorithmes de l'état de l'art pour de nombreuses instances TTP de différentes tailles.

Dans le chapitre 7, nous présentons deux approches pour automatiser le processus de conception d'heuristiques. La première est une approche dite "online" basée sur un algorithm d'estimation de distribution, et la seconde est une approche dite "offline" basée sur la programmation génétique.

Troisieme Partie

Résumé

Bien que le TTP est un problème intéressant qui reflète des caractéristiques importantes de situations réelles, il a été critiqué pour ne pas être plus réaliste. Dans la partie III, nous fournissons un modèle inspiré de l'optimisation de la chaîne d'approvisionnement. Le problème proposé illustre le processus de fabrication dans une chaîne d'approvisionnement à plusieurs niveaux et implique une dépendance temporelle. Dans ce travail, nous introduisons ce problème sous le nom de MJSSP (Multi-component Job Shop Problem).

Dans le chapitre 8, nous introduisons le modèle mathématique du MJSSP et fournissons des détails sur la représentation du problème et la nature des dépendances dans le problème. Dans le chapitre 9, nous proposons un algorithme simple capable de générer des solutions réalisables. L'algorithme proposé est basé sur l'idée de sélectionner le premier intervalle de temps pour chaque tâche séparément.

Contributions

Nous introduisons un nouveau problème qui illustre la dépendance temporelle dans l'optimisation de la chaîne d'approvisionnement à plusieurs niveaux dans le chapitre 8. Le problème est basé sur le problème de Job Shop qui est utilisé pour modéliser le processus de planification dans chaque usine du système. Le problème est de concevoir de manière à être compréhensible tout en étant assez proche des problèmes réels de chaînes d'approvisionnement. Dans le chapitre 9, nous introduisons un algorithme aléatoire-glouton simple pour le MJSSP. L'algorithme peut générer des solutions réalisables sans prêter attention à leur qualité. L'objectif d'un tel algorithme est d'avoir un point de départ et une approche d'initialisation pour des algorithmes plus sophistiqués.

To my beloved family and friends.

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

Researchers in the field of evolutionary computation and metaheuristics are usually interested in $\mathcal{NP}$-complete benchmark problems. Such problems include the Travelling Salesman Problem, the Job Shop Scheduling Problem and the Knapsack Problem among many others. Solving these problems to optimality requires an exponential time unless $\mathcal{P} = \mathcal{NP}$. $\mathcal{NP}$-complete problems are of a great importance in many fields including logistics, engineering, computer security and computational biology. However, some researchers started to observe a significant gap between the benchmark problems considered in state-of-the-art literature and real-world optimisation problem, especially the ones coming from industrial applications [62, 4].

Benchmarking optimisation problems is very important as it provides a public platform for researchers to test, analyse and fairly compare their algorithms. However, many real-world aspects remain neglected or loosely studied. A non-exhaustive list of such aspects is the following.

— Hard and soft constraints: Hard constraints are the constraints that the solution must satisfy in order to be implementable (feasible). An example is the knapsack capacity limit in the Knapsack Problem. Soft constraints are constraints that are prefered but not required to be satisfied. An example is user-specified preferences in planning problems.
— External dependencies: This is related to events happening outside the system over which we do not have any direct control. The best we can do is to acknowledge them and take them into consideration while optimising the operations inside the system. Example of such events can be competition with other systems, unexpected raw material shortage, and stochasticity of traffic while delivering or transporting items.
— Composition and internal dependencies: This aspect arises when the system is composed of multiple interacting sub-problems. Here, we also imply that each sub-problem is an $\mathcal{NP}$-complete problem when isolated from the system. These problem will be referred to as Problems with Multiple Interdependent Components (PMIC).

In this thesis, we will focus on the last abovementioned aspect. More specifically, we are interested in understanding the dependencies between the components in a PMIC, efficiently modelling a PMIC, and how to tackle a PMIC using heuristics, metaheuristics

and hyperheuristics.

The motivation behind using metaheuristics to tackle multi-component problems with dependencies is due to two main factors. The first is the quality/runtime tradeoff. Indeed, in many cases, obtaining a "good" approximated solution in a "fair" amount of time can be better than spending time and resources to generate an optimal solution. This is especially the case in supply chain optimisation and engineering problems. The second is the flexibility of these approaches. In fact, due to their abstract definition, metaheuristics have been successfully applied to a wide range of real-world optimisation problems.

As the title of this thesis suggests, our goal is to understand, model, and design algorithms for multi-component problems with internal dependencies. To achieve our goals, we provide formal definitions of different types of dependency, propose a classification of dependencies, and study special cases of PMICs.

Document organisation and contributions

Part I

Recognising that real-world optimisation problems have multiple interdependent components can be quite easy. However, providing a generic and formal model for dependencies between components can be a tricky task. In fact, a PMIC can be considered simply as a single optimisation problem and the dependencies between components could be investigated by studying the decomposability of the problem and the correlations between the sub-problems. In this work, we attempt to define PMICs by reasoning from a reverse perspective. Instead of considering a decomposable problem, we model multiple problems (the components) and define how these components could be connected.

In Part I, we introduce notions related to problems with mutliple interndependent components. We start by introducing realistic examples from logistics and supply chain management to illustrate the composite nature and dependencies in these problems in Chapter 1. Afterwards, in Chapter 2 we provide our proposal to formalise and classify dependency in multi-component problems. The proposed model is based on definitions proposed in Bonyadi et al. [5].

Contributions

In chapter 2, we propose a mathematical model for three classes of dependencies:
— *fitness dependency* represents the influence of a component on another by changing the fitness of the solutions.
— *feasibility dependency:* represents the influence of a component on another by changing the set of feasible solutions.
— *time dependency:* illustrate the connections between components when the optimisation process for a component is bounded in a time window, and the other component can not start the optimisation process before the first component provides a partial solution.

Part II

In order to reduce the gap between the academic research, which focuses on single component problems, and the problems found in industry, which are multi-component problems with complex dependency, Bonyadi et al. [4] proposed a benchmark problem called the Travelling Thief Problem (TTP).

In Part II, we focus on investigating the TTP from different perspective. In Chapter 3, we start by formally introducing the TTP. Then we analyse the problem by studying the impact of dependency on the complexity of the objective function and on the search process for each component. Afterwards, we describe the TTP benchmark dataset introduced by Polyakovskiy et al. [70]. In Chapter 4, we provide a more in-depth analysis using local optima networkds, which is a tool to perform fitness landscape analysis on combinatorial problems. In Chapter 5, we introduce initialisation strategies, local search algorithms, and investigate an alternative model for the TTP based on multi-objective optimisation. Afterwards, we propose more sophisticated heuristic solutions able to tackle very large TTP instances in Chapter 6. In Chapter 7, we propose the use of hyperheuristics to automatically design algorithms for the TTP based on low-level heuristics.

Contributions

The investigation of the TTP resulted in multiple contributions on the analysis of the problem and algorithm design. In chapter 4, we propose the use of a compressed fitness landscape representation called local optimsa networks for the TTP. This analysis allowed us to draw important conclusions about the nature of the search space of the TTP. In chapter 5, we propose two local search algorithms based on the combination of the components' neighbourhoods. The experiments showed that these heuristics are competitive with state-of-the-art algorithms for small and mid-size instances, but that they have scalling issues due to their high computational complexity. Nevertheless, these approaches are important as they tackle the TTP as a whole, which is an important feature which proved to be useful to conduct the fitness landscape analysis abovementioned. Besides the TTP's composite nature and the existence of dependencies, the TTP benchmark dataset introduces many challenging large-scale instances with up to 85900 cities and 858990 items. To tackle this issues, in Chapter 6 we present two metaheuristic adaptions. The first is based on the simulated annealing algorithm and the hill climbing local search, and the second is a memetic algorithm. These approaches were able to surpass state-of-the-art algorithms for many TTP instances of various sizes. In chapter 7, we introduce two approaches to automate the heuristic design process. The first is an online approach based on an estimation of distribution algorithm, and the second is an offline approach based on genetic programming.

Part III

While the TTP is an interesting problem that reflects important features of real-world situations, it has been critised for not being more realistic. In part III, we provide a model that is inspired from real-world supply chain optimisation. The proposed problem illustrates the manufacturing process in a multi-level supply chain, and embeds a time dependency. In the rest of this document, we will refer to this problem as the Multi-

component Job Shop Scheduling Problem (MJSSP).

In Chapter 8, we introduce the mathematical model of the MJSSP and provide details about the problem representation and the nature of dependencies in the problem. In Chapter 9, we propose a simple algorithm able to generates feasible solutions. The proposed algorithm is based on the idea of selecting the earliest time slot for each task separately.

Contributions

We introduce a new problem that illustrates time dependency in multi-level supply chain optimisation in Chapter 8. The problem is based on the standard Job Shop Scheduling Problem which is used to model the scheduling process at each plant in the system. The problem is design in a way to be comprehensible while being close enough to real-world supply chains. In chapter 9, we introduce a simple randomised greedy algorithm for the MJSSP. The algorithm can generate feasible solutions without giving attention to their quality. The goal of such algorithm is to have a starting point and initialisation approach for more sophisticated algorithms.

Part I

Multi-component problems with dependencies

CHAPTER 1

Multi-component problems with internal dependencies: illustrative examples and high-level definitions

A problem with multiple interdependent components is an optimisation problem that embeds multiple sub-problems, usually called components, where the components can not be solved in isolation. This means that solving each component to optimality does not necessarily guarantee obtaining an optimal overall solution if the other components are not considered.

In this chapter, we provide an informal introduction using two illustrative examples. The examples are designed in order to reflect aspects from real-world optimisation problems, while kept as simple as possible.

1.1 The problem of finding warehouse locations and distribution routes

Let us start by considering the following problem inspired by the Location-Routing Problem [69]. Informally, the problem states as follows.

Given a set of m potential distribution center (DC) locations and the associated cost for establishing a warehouse, a set of n clients and their associated requests. Each client is to be associated to a regional DC. The size of the warehouse depends on the number of clients it serves. Therefore, the cost of establishing a warehouse also depends on the number of associated clients. Deliveries are done in routes where a vehicle serves multiple clients in one trip. We suppose that each DC has one vehicle to serve all the clients, and that the transportation cost depends on the total distances of the trips.

The goal is to find a set of k locations (where k is a constant lower than n) and a set of routes where each client is served by its regional DC, such that the warehouse establishing and the transportation costs are minimised.

1.1.1 Composition

Given the above statement, it is clear that the problem described embeds the following two components.

— **A variant of the Facility Location Problem (FLP) [13]:** which corresponds to the problem of finding a set of k locations to establish the warehouses such that it covers a regional area and reaches as many clients as possible. A solution for this problem is represented as a set of k locations and thier associated clients.

— **A variant of the Vehicle Routing Problem (VRP) [16]:** which is the problem of finding a route (where a route is a set of ordered clients) for each DC such that each client gets served by a regional DC [77]. A solution for this VRP variant is a set of k routes.

The overall objective function can be expressed as the sum of the two components:

Minimise *sum of distances between client and facility + facility establishment costs + sum of route distances*

1.1.2 Dependencies between the components

The interdependence between the two components can be easily recognised. In order to better understand the dependencies between the VRP and the FLP, let us consider the following scenarios.

— **Changing a warehouse location** might change the total distance of the routes. Meaning that changing the FLP solution might impact the objective value of the VRP solution.

— **Changing a client's region** makes the routes associated non-feasible. Meaning that changing the FLP solution impacts the feasibility of the VRP solution.

— **Changing the routes by assigning a client to another route** makes the set of associated clients non-feasible. This means that changing the VRP solution impacts the feasibility of the FLP solution.

What we can learn from the above problem is that in real-world situations, optimisation problems can be composed of multiple interdependent components. Furthermore, in the example we introduced two types of dependencies: the first one influences the objective value of solutions (*fitness dependency*), while the second influences their feasibility (*feasibility dependency*).

1.2 The problem of demand scheduling and truck loading

As a second illustrative example, let us consider the logistics problem expressed in the following statement.

A company owns a plant containing m machines. The plant manfactures on-demand items of p products by processing every item on every machine given a specific order. The plant must follow a weekly schedule in order to manufacture its products. At the end of every week, the finished items (defined by their volumes) should be delivered to be loaded in rented containers in order to make deliveries. This process of manufacturing and containers loading is repeated every week until the overall demand is satisfied.

The goal is to find a set of week schedules and a set of packing plans such that the

delay and renting rate of containers are minimised.

1.2.1 Composition

It is clear that the described problem is composed of the following sub-problems.

— **The Job Shop Scheduling Problem (JSSP) [37]:** which corresponds to the problem of finding a week schedule that assigns manufacturing tasks to machines.

— **The Bin Packing Problem (BPP) [44]:** which corresponds to the problem of finding the optimal packing plans for containers.

In a sense, due to the time window constraint, the problem is composed of many JSSP and BPP sub-problems. In fact, the number of these sub-problems is in its own a variable and should be minimised, which translates to minimising the overall delay. Therefore, the objective function of the overall problem can be defined as follows:

Minimise *total delay + containers renting rate*

1.2.2 Dependencies between the components

The concept of dependecy in this problem is very different from the one explained in the first example. In fact, changing a schedule of a given week may not only change the feasibility or objective value of the BPP solution; but it can change the dimension of the BPP problem.

Let us explain: when changing a week schedule –say to another one that produces more items– the number of items to be loaded into containers increases, i.e., the problem dimension increases. We will refer to this type of dependency as *Time dependency*.

The same conclusions can be drawn between the schedule for a given week and the Job Shop Scheduling Problem of the following week. Indeed, changing the schedule for a given week w may change the number of tasks to be scheduled for the week $w + 1$. In other words, the dimension of the JSSP is dependent on the schedule adopted the week before.

Based on these two illustrative examples, we can informally define a problem with multiple interdependent component as a problem composed of multiple sub-problems that are non-separable. The non-separability can be due to three factors: impact on the objective value, impact on feasibility and impact on the dimension of a sub-problem.

1.3 What differentiates multi-component problems from single-component ones?

Problems with multiple interdependent components can be simply thought of as another way to look at decomposition and non-separability. However, there are some key features that distinguish problems with multiple interdependent components from previous related works:

1. **Intractability of components:** there are many existing examples allowing to argue that real-world problems are composed of sub-problems that are themselves intractable [69, 42, 63, 40, 41, 71, 82]. As mentioned earlier, we are interested in problems where all the components are $\mathcal{NP}$-complete.

2. **Heterogeneity or natural decomposability of the overall problem:** meaning that components can be considered as different optimisation problems. While this may not be of interest from a computational complexity perspective (since all the components are $\mathcal{NP}$-complete), it is highly important from a modelling, representation and heuristic point-of-view. In fact, some $\mathcal{NP}$-complete problems are harder to model, represent and find approximate solutions for.

1.4 Conclusion

The main goal of this chapter was to introduce the idea of dependency between component through two comprehensive examples. In the next chapter, we provide our attempt to formally define the dependencies between components in an optimisation problem.

CHAPTER 2

A formal model for problems with multiple interdependent components

In this chapter, we propose formal definitions of problems with multiple interdependent components and classify three forms of dependencies. Our model builds on definitions introduced by Bonyadi et al. [5], while extending and formalising the concepts with the aim of presenting a more accurate model.

2.1 Preliminaries

Let $P = \{P_1, \ldots, P_n\}$ be a set of minimisation problems called *components*, where a component $P_i, i \in \{1, \ldots, n\}$, is defined by the following.

- We note I^{P_i} the set of all possible P_i instances with a fixed dimension denoted m^{P_i}.
- $\mathcal{S}^{P_i}$ denotes the set of all possible solution configurations of P_i. That is, using an N-bitstring representation, the set of all 2^N possible configurations.
- Given an instance $x \in I^{P_i}$, we note $F^{P_i}(x) \subseteq S^{P_i}$ the set of feasible solutions.
- χ^{P_i} is a function that associates an instance from I^{P_i} to a given $(n-1)$-tuple containing a solution configuration for every other component $P_j, j \in \{1, \ldots, n\} \setminus \{i\}$ (Eq. 2.1).

$$\chi^{P_1} \colon \mathcal{S}^{P_1} \times \cdots \times \mathcal{S}^{P_{i-1}} \times \mathcal{S}^{P_{i+1}} \times \cdots \times \mathcal{S}^{P_n} \to I^{P_i} \tag{2.1}$$

- We note Z^{P_i} the component objective function of P_i that associates a real number z to an instance x of P_i, a solution s of x, and $n-1$ solution configurations for every other problem $P_j, j \in \{1, \ldots, n\}$, as expressed in Equation 2.2.

$$\begin{aligned} Z^{P_i} \colon \chi^{P_i}(\overline{s}^{P_1}, \ldots, \overline{s}^{P_{i-1}}, \overline{s}^{P_{i+1}}, \ldots, \overline{s}^{P_n}) \times & \\ F^{P_i}(x) \times & \\ \mathcal{S}^{P_1} \times \cdots \times \mathcal{S}^{P_{i-1}} \times \mathcal{S}^{P_{i+1}} \times \cdots \times \mathcal{S}^{P_n} \to \mathbb{R} & \\ (x, s, \overline{s}^{P_1}, \ldots, \overline{s}^{P_{i-1}}, \overline{s}^{P_{i+1}}, \ldots, \overline{s}^{P_n}) \mapsto z & \end{aligned} \tag{2.2}$$

2.2 Instance-dependency — Dependency of fitness and feasibility

We start by defining a general type of dependency where a change of solution for one problem impacts the instance of the other problem (Definition 1).

Definition 1 (Instance-dependency)**.** Let P_i and P_j be two components in P. We say that P_i is instance-dependent on P_j (notation $P_i \leftarrow P_j$) if $\exists s_1^{P_j}, s_2^{P_j} \in \mathcal{S}^{P_j}$ such that:

$$\chi^{P_i}(\overline{s}^{P_1}, \ldots, s_1^{P_j}, \ldots, \overline{s}^{P_n}) \neq \chi^{P_i}(\overline{s}^{P_1}, \ldots, s_2^{P_j}, \ldots, \overline{s}^{P_n})$$

$\forall \overline{s}^{P_k} \in \mathcal{S}^{P_k}, \forall k \in \{1, \ldots, n\} \setminus \{i, j\}$

The first illustrative example in chapter 1 demonstrated two forms of dependencies. The first is when a change of the solution on one problem impacts the fitnesses of solutions for the other problem as expressed in Definition 2. While the second can be seen as a stronger form of instance dependency where the impact of changing the solution of a problem impact the feasibility of the solutions as shown in Definition 3. In our definitions, we conjecture that an instance dependency can only occur in one of these two forms.

Definition 2 (Fitness-dependency)**.** Let P_i and P_j be two components in P. We say that P_i is fitness-dependent on P_j (notation $P_i \xleftarrow{fitness} P_j$) if
— $P_i \leftarrow P_j$.
— $\forall x_1, x_2 \in I^{P_i}, F^{P_i}(x_1) = F^{P_i}(x_2)$.

Definition 3 (Feasibility-dependency)**.** Let P_i and P_j be two components in P. We say that P_i is feasibility-dependent on P_j (notation $P_i \xleftarrow{feasibility} P_j$) if
— $P_i \leftarrow P_j$.
— $\exists x_1, x_2 \in I^{P_i}, F^{P_i}(x_1) \neq F^{P_i}(x_2)$.

Note that when both fitness- and feasibility-dependency are possible between two problems (for instance by considering two different change operators), fitness-dependency can be ommited as it is considered a weaker form of dependency.

Using Definitions 2 and 3, we can construct a directed graph $G = (P, D)$, called the graph of dependencies, consisting of the set of nodes P which corresponds to the set of problems previously mentioned; and the set D of edges, which are ordered pairs of elements of P, and which represent the dependencies between the problems.

If the graph G is connected, we call P a *multi-component problem with internal dependencies* or a *problem with multiple interdependent components*[1], P_i a component of P, and Z^{P_i} a component objective function. We can then define either the overall objective function of P, noted Z^P in terms of component objectives (Equation 2.3); or consider P as a multi-objective problem where the objective functions correspond to the component objective functions.

Figure 2.1 illustrates the graph of dependency for the Location-routing problem introduced in Chapter 1. Note that changing the FLP solution might impact both the fitness and the feasibility of solution. However, the fitness-dependency is ignored as previously explained.

1. the word interdependent here stands for internal dependencies and not for multual dependencies

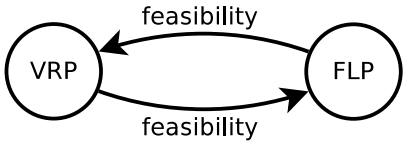


Figure 2.1: The LRP graph of dependency

$$Z^P : \chi^{P_1}(s^{P_2}, \ldots, s^{P_n}) \times \cdots \times \chi^{P_n}(s^{P_1}, \ldots, s^{P_{n-1}}) \to \mathbb{R} \qquad (2.3)$$
$$(s^{P_1}, \ldots, s^{P_n}) \mapsto z$$

A convenient way to define Z^P is by using a weighted sum of the component objectives as expressed in Equation 2.4.

$$Z^P(s^{P_1}, \ldots, s^{P_n}) = \sum_{k=1}^{n} \alpha_k Z^{P_k}(s^{P_k}) \qquad (2.4)$$

where α_k is a real number.

Definition 4. Let P be a minimisation problem and $Z^P(.)$ its objective function. We define $\widetilde{P}$ as the decision problem associated with a given problem P. That is, given $k \in \mathbb{R}$, is there a solution s to $\widetilde{P}$ such that $Z^P(s) \leq k$?

By considering the weighted sum function, we can easilly prove –without considering of the existence of internal dependencies– the $\mathcal{NP}$-completeness of the overall problem when all the components are in $\mathcal{NP}$ and at least one component is $\mathcal{NP}$-hard.

Claim: Let $P = \{P_1, \ldots, P_n\}$ be a minimisation problem with n interdependent components defined by the weighted sum objective function in Equation 2.4. If $\forall P_i \in P$ such that $\widetilde{P_i}$ is in $\mathcal{NP}$ and $\exists P_j \in P$ such that $\widetilde{P_j}$ is $\mathcal{NP}$-hard, Then $\widetilde{P}$ is $\mathcal{NP}$-complete.

First, we need to prove that P is in $\mathcal{NP}$. Since all the components of P are in $\mathcal{NP}$, the solutions can be verified in polynomial time. Thus, the objective function can also be calculated in polynomial time, i.e. the overall solution $(s^{P_1}, \ldots, s^{P_n})$ can be verified in polynomial time.

Second, to prove that $\widetilde{P}$ is $\mathcal{NP}$-hard, we need to reduce a known $\mathcal{NP}$-hard problem to P in polynomial time. To do so, we can reduce $\widetilde{P_i}$ to $\widetilde{P}$ by setting $\alpha_i = 1$, and $\alpha_j = 0, \forall j \neq i$, which concludes the proof.

In this thesis, we are interested in problems where each component is associated to an $\mathcal{NP}$-complete decision problem. In state-of-the-art, these problems are referred to as multi-hard problems [73], multi-silo problems [40, 41], or simply multi-component problems [4]. These referred works argue that many real-world optimisation problems are composed of multiple interacting sub-problems, where each sub-problem is practically intractable.

2.3 Time dependency

Time dependency can be seen as another form of dependency between components where the notion of *time window* is considered. In order to define this notion, we need

to consider the following additional points.

— There is a subset of components in P that receive a stream of data as input. In addition, this process of data feeding can be done on multiple time slots.

— Some components in P might have a time constraint. Which means that such components have a limited amount of time to process the data stream and generate a solution to the problem instance. If such solution cannot be generated, a smaller instance is derived such that it is possible to generate solutions for it with respect to the allowed time window. In other words, the components themselves can be decomposed into smaller sub-problems to satisfy the time constraint.

Definition 5 (Time dependency)**.** Let P_i and P_j be two components in P. We say that P_i is time dependent on P_j (notation $P_i \xleftarrow{time} P_j$) if

— P_j receives a data stream from P_i. This data stream depends exclusively on the instance considered for P_i.

— A change of the instance considered for P_i induces a change of the instance for P_j. More precisely, the dimension of the P_j instance is equal to the size of the P_i instance from which P_j receives the data stream.

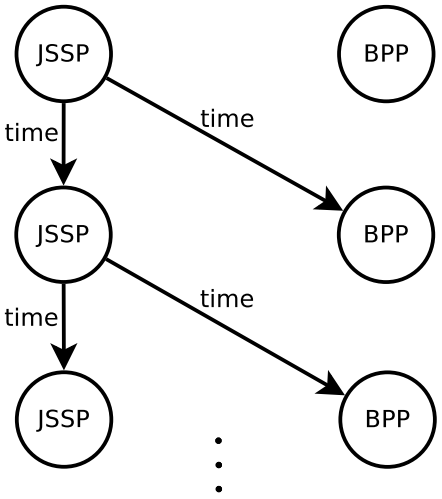


Figure 2.2: The graph of dependency for the problem of demand scheduling and truck loading

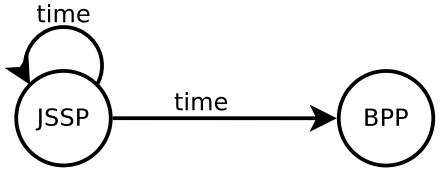


Figure 2.3: A compressed version of the graph of dependency for the problem of demand scheduling and truck loading

A problem with time dependencies can be modelled using a directed graph with an undetermined number of vertices, where the vertices represent the components' sub-problems, and edges represent the connections from a time window d to the next one $d+1$. Figure 2.2 illustrates the graph of dependecy for the problem of demand scheduling and truck loading briefly presented in Chapter 1.

Another possibility to model a problem with time dependencies is to use a regular graph, where the vertices represent the components and the edges represent the connections between the components' sub-problems from a time window d to $d + 1$ as shown in Figure 2.3 for the same problem.

2.4 Conclusion

In this chapter, we introduced a formal model that defines the concept of dependencies in multi-component problems in a general fashion. The model is based on former research Bonyadi et al. [5] and our experience with multi-component problems with dependencies.

Our goal was to propose a general mathematical model in order to distinguish and classify dependencies in multi-component optimisation problem. The model allowed us to generate the graph of dependencies of a given problem based on our definitions of multiple dependency types. We believe that these results will be important and could be used as building blocs for future theoretical investigation of multi-component problems with dependencies.

Part II

The travelling thief problem: a case study of instance dependency

CHAPTER 3

The travelling thief problem: a first attempt to model problems with multiple interdependent components

In this chapter, we present the Travelling Thief Problem (TTP), a benchmark problem that attempts to reflect dependencies between components in real-world optimisation problems. The problem is mathematically formulated, and the dependency between the two components is analysed by studying the impact of combination and dependency on the complexity of the objective function.

3.1 Problem formulation

Herein, we formulate the Travelling Thief Problem which combines two other well-known benchmark problems: the Traveling Salesman Problem (TSP) and the Knapsack Problem (KP). In the TTP, we consider n cities and the associated distance matrix $\{d_{ij}\}$ such that $i, j \in \{1, \ldots, n\}$. We suppose there are m items scattered in these cities, each item k having a profit p_k and a weight w_k. A thief with his rented knapsack is going to visit all these cities once while picking up some items to fill his knapsack. We note W the maximum capacity of the knapsack and v_{min} and v_{max} denote the minimum and maximum possible velocity respectively. In addition, we consider the following points:

— Each item is available in only one city. We note $\{A_i\}$ as the availability vector such as $A_i \in \{1, \ldots, n\}$. $\{A_i\}$ contains the reference to the city that contains the item i.

— We suppose that the knapsack is rented, and we note R as the renting price per time unit.

— The velocity of the thief changes accordingly to the knapsack weight. The heavier the knapsack gets, the slower the thief becomes. v_{x_i} denotes the velocity of the thief at city x_i as defined in equation 3.1.

$$v_{x_i} = v_{max} - C \times w_{x_i} \tag{3.1}$$

where $C = \frac{v_{max} - v_{min}}{W}$ is a constant value and w_{x_i} represents the weight of the

knapsack at city x_i.

The goal of the problem is to find the tour x and the picking plan z that maximise the total travel gain G defined in equation 3.2.

$$G(x,z) = g(z) - R \times f(x,z) \tag{3.2}$$

where $g(z) = \sum_{k=1}^{m} p_k \times z_k$ is the total value of the items subject to $\sum_{k=1}^{m} w_k \times z_k \leq W$, and $f(x,z) = \sum_{i=1}^{n-1} t_{x_i,x_{i+1}} + t_{x_n,x_1}$ is the total travel time such that $t_{x_i,x_{i+1}} = \frac{d_{x_i,x_{i+1}}}{v_{x_i}}$ is the travel time from x_i to x_{i+1}.

A TTP solution is represented as follows:

— The tour $x = (x_1, ..., x_n)$ is a vector containing the ordered list of cities.

— The picking plan $z = (z_1, ..., z_m)$ is a binary vector such as z_i is equal to 1 if the item i is picked, 0 otherwise.

It is clear given the TTP formulations that the type of dependency it embeds is a fitness-dependency. The graph of dependency of the TTP is illustrated in Figure 3.1.

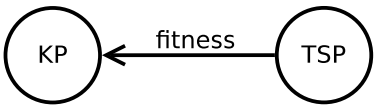


Figure 3.1: The TTP graph of dependency

The NP-hardness of the TTP is trivial as it can be seen as a generalization of the TSP or the KP, both known to be NP-hard [68, 57]. The TTP can be reduced to a TSP by assigning a zero to all the profits. Similarly, it can be seen as a KP if one city is uniquely considered or if the renting rate is set to zero.

3.2 Dependency analysis

The interdependence between the KP and the TSP has already been investigated. For instance, it has been shown, using a counter example, that optimising the sub-problems in separation does not guarantee the finding of good solutions for the overall problem[4]. Therefore, finding good global solutions requires an algorithm that takes the interdependence of components into consideration, which makes the design of such an algorithm quite difficult.

In this section, the interdependence in the TTP is analysed through its effect on solving the problem given particular search operators.

3.2.1 In the TSP component

The first consequence of the interdependence in the TTP relies on the fact that the TSP component stops being a symmetric problem. This is due to the metric change in the TTP, which considers the time of travel instead of the distance. The travel time from a city A to another city B is not equal to the travel time from B to A. Indeed, the cities may contain items of different weights which leads to different velocities.

Therefore, when using the 2-OPT heuristic, retrieving the objective value from a

tour in which two arcs have been exchanged has a linear complexity $O(N)$, such as N is the number of cities in the sub-tour, unlike the symmetric TSP, in which the retrieval of the objective value for neighbour solutions can be performed instantly ($O(1)$).

In the TTP, the time of travel changes according to the weight of the knapsack. This dynamic aspect of the TTP makes it even harder to tackle the problem. However, by keeping track of the time and weight at each city, it is possible to recover the objective value of a 2-OPT neighbour solution in order to avoid using the objective function which is more expensive.

Consider a TTP solution (x, z) such as $x = x_1, \ldots, x_n$ is the tour and $z = z_1, \ldots, z_m$ is the picking plan. Let $f(x, z)$ be the total travelling time defined in equation 3.3.

$$\begin{aligned} f(x,z) = & \frac{d_{x_1,x_2}}{v_{max} - C \times w_{x_1}^{acc}} + \cdots + \\ & \frac{d_{x_{i-1},x_i}}{v_{max} - C \times w_{x_{i-1}}^{acc}} + \frac{d_{x_i,x_{i+1}}}{v_{max} - C \times w_{x_i}^{acc}} + \frac{d_{x_{i+1},x_{i+2}}}{v_{max} - C \times w_{x_{i+1}}^{acc}} \\ & + \cdots + \frac{d_{x_n,x_1}}{v_{max} - C \times w_{x_n}^{acc}} \end{aligned} \tag{3.3}$$

where w_a^{acc} denotes the current weight at city a.

Let (x^{ij}, z) be the neighbour solution of (x, z) such as the 2-OPT operator is applied at the indices i and j. Computing the total travelling time for (x^{ij}, z) can be done using equation 3.4.

$$\begin{aligned} f(x^{ij},z) = & \frac{d_{x_1,x_2}}{v_{max} - C \times w_{x_1}^{acc}} + \cdots + \\ & \frac{d_{x_i,x_j}}{v_{max} - C \times w_{x_i}^{acc}} + \frac{d_{x_j,x_{j-1}}}{v_{max} - C \times (w_{x_i}^{acc} + w_{x_j}^{reg})} + \frac{d_{x_{j-1},x_{j-2}}}{v_{max} - C \times (w_{x_i}^{acc} + w_{x_j}^{reg} + w_{x_{j-1}}^{reg})} + \cdots + \\ & \frac{d_{x_{i+1},x_{j+1}}}{v_{max} - C \times (w_{x_{i+1}}^{acc} + w_{x_j}^{reg} + w_{x_{j-1}}^{reg} + \cdots + w_{x_{i+1}}^{reg})} + \\ & \frac{d_{x_{j+1},x_{j+2}}}{v_{max} - C \times w_{x_{j+1}}^{acc}} + \cdots + \frac{d_{x_n,x_1}}{v_{max} - C \times w_{x_n}^{acc}} \end{aligned} \tag{3.4}$$

where w_a^{reg} represents the weight of all the items taken from city a.

3.2.2 In the KP component

The second consequence of interdependence occurs when picking or unpacking items (i.e. using the bit-flip operation). In fact, it is impossible to recover the travelling time in a constant time since the change affects all the terms that come after the region of change (i.e. the term corresponding to the city from which the item was packed or unpacked). Nevertheless, the objective value of a neighbour solution can be recovered by reusing the terms before the region of change and keeping track of time and the weight of the original solution.

This technique has a linear complexity $O(N)$, such as N is the number of cities from the location where the bit-flip happens to the last city in the tour.

In this second case, we consider the solution (x, z^k) such as the item z_k is picked (or unpacked) from city i.

The value $f(x, z^k)$ can be recovered partially from $f(x, z)$ by reusing the terms before the region of change, and keeping track of time and weight of the original solution.

$$f(x, z^k) = \frac{d_{x_1,x_2}}{v_{max} - C \times w_{x_1}} + \cdots + \frac{d_{x_{i-1},x_i}}{v_{max} - C \times w_{x_{i-1}}} + \frac{d_{x_i,x_{i+1}}}{v_{max} - C \times (w_{x_i} + \Delta_w)} + \frac{d_{x_{i+1},x_{i+2}}}{v_{max} - C \times (w_{x_{i+1}} + \Delta_w)} + \cdots + \frac{d_{x_n,x_1}}{v_{max} - C \times (w_{x_n} + \Delta_w)} \tag{3.5}$$

where Δ_w represents the difference of weight after applying the bit-flip operator.

3.3 On the impact of component heuristics

Most researchers consider the KP component as the most critical part of the TTP and mainly focused on it to design solvers. We believe that this assumption is fairly justified. However, none of the TTP related papers we are aware of discussed or investigated this aspect. Performing multiple experiments on the TTP shows that the KP component provides more room for optimisation compared to the TSP component. Herein, we experimentally show that the total gain (G) increases more significantly when the KP part is tackled compared to when the TSP part is optimised.

To perform our study, we start by generating an initial solution combining a near-optimal tour generated using the Lin-Kernighan heuristic, and an optimal picking plan generated using dynamic programming. Afterwards, a component solvers is applied individually to solve the problem. We are interested in the rate of improvement brought by each component solver.

Table 3.1: A comparison of the impact of component solvers on the objective function (G), the travel time (f), and the final profit (g) for a small-size TTP instance (51 cities and 150 items)

	Initial solution	After TSP hill climbing	After KP hill climbing
G	-779	283	6101
f	1070	1019	642
g	21247	21247	19325

Table 3.1 summarizes the results of the experiment. We report the objective value (second row), the travel time (third row), and the final profit (fourth row) for the initial solution (second column), the solution after applying the hill climbing to the TSP component (third column), and the solution after applying the hill climbing to the KP component (fourth column). Note that for simplicity reasons, the results are reported for only one instance. Nevertheless, all the other tests we have performed for other instances show the same behaviour.

Clearly, improving the picking plan has more impact on the overall objective than improving the tour. The results tend to show that this behaviour is not due the per-

formance of the component solvers, but instead to the problem nature itself. Equation 3.2 provides a simple explanation: *both the final profit and the travel time depend on the picking plan, while the tour is only present in the travel time's function. Therefore, optimising the picking plan has more impact on the overall objective.*

It is worth noting that a second version of the TTP was proposed in the original paper named TTP2 [4]. In TTP2, in addition to the speed variation constraint (Equation 3.1), it is also supposed that the value of items drops with time. We believe that this would be a more balanced model due to the fact that both the final profit (g) and the travel time (f) are dependent on the picking plan and the tour. This creates a two-way (mutual) fitness-dependency between the two components which increases the problem difficulty. An important challenging aspect is that it would be difficult to develop speedup approaches to improve the runtime, making very large instances even harder to solve.

3.4 Benchmark dataset and problem properties

The TTP instances can be classified according to the following properties and diversification parameters.

— **Number of cities (n):** The only parameter belonging to the TSP component in the TTP definition. The TTP benchmark library [70] gets its TSP component from the TSPLIB database which was introduced in [75] which contains 81 instances.

— **Item Factor ($\mathcal{F}$):** Represents the number of items per city. Each city, except the first one, has the same number of items. The total number of items is $m = (n-1) \times \mathcal{F}$.

— **Profit-value correlation ($\mathcal{T}$):** Defines the correlation among the weight (w_i) and profit (p_i) of each item. Three correlations have been defined in the TTP library, namely, *uncorrelated (unc)*, *uncorrelated with similar weight (usw)*, and *bounded strongly correlated (bsc)*.

— **Knapsack capacity class ($\mathcal{C}$):** Ranges between 2 and 10. $\mathcal{C}$ is a factor occurring in the maximum weight of the knapsack which is given in Equation 3.6.

$$W = \frac{\mathcal{C}}{11} \sum_{x=2}^{n} \sum_{y=1}^{\mathcal{F}} w_{xy} \tag{3.6}$$

where the term $C/11$ is used to limit W, i.e., class $C = 10$ enlarges W around to 90%, allowing more objects in the knapsack, and w_{xy} represents the weight [70].

The combination of these different problem properties results in a very large dataset of $9,270$ instances. Due to the large size of this dataset, our experiments will usually focus on a representative subset of instances.

Note that all the algorithms discussed in this work have a maximum runtime limit of 600 seconds, which is used as a standard for TTP algorithms.

3.5 Conclusion

In this chapter, we introduced the Travelling Thief Problem (TTP), a recently proposed benchmark problem that attempts to reflect dependencies between components in real-world optimisation problems. The TTP has two components based on the classical Travelling Salesman Problem and the Knapsack Problem. After formulating the problem mathematically, we briefly analysed the dependency between the two components by studying the impact of combination and dependency on the complexity of the objective function. Finally, we introduced the official TTP benchmark dataset and the parameters used to diversity the TTP database.

CHAPTER 4

A fitness landscape analysis of the travelling thief problem

In combinatorial optimisation, a fitness landscape of a given problem can be defined as a graph where nodes represent solutions, and edges represent the existence of a neighbourhood relation given a move operator [89]. However, there are two main issues with this representation. Firstly, defining the neighbourhood matrix for the entire set of possible solutions can be a very expensive task even for small instances. Secondly, it is hard to extract useful information about the search landscape given the fitness landscape graph.

In our analysis of the TTP, we use the compressed representation called Local Optima Networks (LONs) proposed by Ochoa et al. [66]. In this representation, the fitness landscape is represented as a graph whose nodes are associated with local optima (or basins of attraction) and edges indicate connectivities between the local optima. Two basins of attraction are connected if at least one solution within a basin has a neighbour solution within the other given a defined move operator. LONs provide a simplified yet very useful representation of the search space of combinatorial problems, which can be exploited mainly by using measures and indices from graph theory. LON characteristics have also been found to correlate with the performance of heuristic search algorithms [39]. In this chapter, we study two non-trivial problem properties, namely the knapsack capacity and the correlation between the weights and the profits of the items.

The main goal is to understand the search space structure of the TTP using basic local search heuristics, and to distinguish the most impactful non-trivial problem features. Thus, we study the knapsack capacity and the profit-weight correlation. In this work, we (i) propose a problem classification based on these two features, (ii) create a large set of enumerable TTP instances according to the features, (iii) generate a LON for each instance using two hill climbing variants, (iv) explore/exploit, using specific measures, the obtained LONs to gain some insights into their structure and characteristics.

The problem difficulty can be recognised using LON topological properties or information about the basins of attraction. Our analysis on such information corroborates the general idea that, using basic local search techniques there is a direct correlation between lower knapsack capacity and higher problem difficulty. In addition, other LON

examinations revealed some real-world networks' characteristics such as cliqueness and sparse interconnectivity. We believe that these insights might contribute to the design of efficient local search-based heuristics for the TTP, and to understanding the search space structure of complex problems with interdependent components in general.

4.1 Fitness landscape analysis

Fitness landscapes illustrate the association between search and fitness space, often depicted as rugged surfaces with many local peaks of different heights flanked by valleys of different depths [46, 45]. Given a specific landscape structure, defined as the triple (S, F, N) where S is the search space, F the objective function, and N the neighbourhood operator [89], a heuristic can be seen as a strategy for navigating this structure in the search for the highest peak or near-optimal solutions [83]. Fitness landscape analysis (FLA) has been applied to investigate the dynamics of evolutionary algorithms and single-solution heuristics for optimisation and design problems [86]. In addition, FLA can help predict the performance of those heuristics by using the search cost models for instance [83].

Landscape models have been used to make specific predictions regarding the behaviour of local search techniques, evolutionary algorithms, and other metaheuristics [90]. The behaviour is generally illustrated by the cost required to locate a solution with a given quality threshold given a problem instance. These models can also identify which features of the fitness landscape are responsible for the problem difficulty during the search process [89].

In order to understand the structural organisation of the local optima in combinatorial landscapes, Ochoa et al. [66] proposed the Local Optima Networks (LONs) as a simplified fitness landscape model. There, the fitness landscape is represented as a graph of connected local optima.

A local search heuristic $\mathcal{A}$ defines a mapping from the solution space S to the set of locally optimal solutions S^*. A solution i in the solution space S is a local maximum given a neighbourhood operator N if $F(i) \geq F(s), \forall s \in N(i)$. Each local optimum has an associated basin of attraction. In general, the basin of attraction of a local optima i is the set composed of all the solutions that, after applying a local search procedure starting from each of them, the procedure returns i. Thereby, the basin of attraction associated to a local optimum i is the set $B_i = \{s \in S | \mathcal{A}(s) = i\}$. The size of the basin of attraction of i is the cardinality of B_i. Given a neighbourhood operator, a connection between two attraction basins is created if at least one solution in one basin has a neighbour solution in the other basin.

Figure 4.1 represents a simplified illustration of the attraction basins (black circles), their local optima (red big dots), the solutions that converge to the local optima when applying the local search (black small dots), and the connections between the local optima (blue lines). Note that the figure is kept simple for visualisation purposes, and more sophisticated heuristics with explorative operators are expected to result in many more interconnections between the attraction basins.

Local optima network properties for permutation-based problems have been studied in [17]. Furthermore, some works analysed the correlation between LON features and the performance of search heuristics [85, 9, 67].

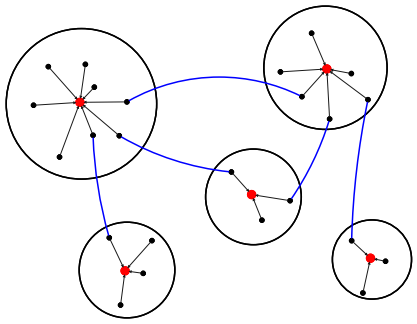


Figure 4.1: A simplified illustration of the attraction basins and the connectivity in local optima networks.

4.2 Local search heuristics

In order to conduct our analysis, we use a basic local search framework for the TTP. The framework is a deterministic hill climber, designed with the main goal of investigating the structure of the problem search space. The pseudocode is described in Algorithm 1, and it can have multiple implementations depending on the chosen neighbourhood operators. Note that $\mathcal{N}_{TSP}(.)$ and $\mathcal{N}_{KP}(.)$ represent the neighbourhood functions for the TSP component and the KP components respectively.

Algorithm 1 A basic local search heuristic framework for the TTP

```
1  s ← initial solution
2  while there is an improvement do
3      for each s* ∈ N_TSP(s) do
4          for each s** ∈ N_KP(s*) do
5              if F(s**) > F(s) then
6                  s ← s**
7              end if
8          end for
9      end for
10 end while
```

Although most state-of-the-art heuristics proposed for the TTP have a two-stage structure [1] [60, 32, 28], the use of an embedded neighbourhood structure is crucial for this study. In fact, two-stage heuristics divide the TTP solution into two components at each iteration and generate a large number of landscapes (one landscape for each sub-problem at each heuristic iteration) — this makes it virtually impossible to investigate the local optima structure of the overall problem. On the other hand, a joint-neighbourhood structure generates a problem specific neighbourhood function and preserves homogeneity of the TTP solutions. This helps the heuristic to easily and efficiently depict the structure and connectivity of local optima. This local search framework was explored in [27] and showed a competitive performance for small TTP instances compared with other basic stochastic heuristics. However, the approach shows some drawbacks, notably with scalability and exploration abilities (a detailed report on the performance of these approaches is provided in chapter 5).

1. This family of heuristics solve the problems by tackling each sub-problem individually using a heuristic search. The process is then iterated multiple times depending on the stopping criteria.

In the context of this study, we consider two local search variants based on Algorithm 1. The first (named J2B) uses the *2-OPT* neighbourhood as the $\mathcal{N}_{TSP}(.)$ neighbourhood, while the second (named JIB) uses the *insertion* operator. In both variants, the one-bit-flip operator is used to construct the $\mathcal{N}_{KP}(.)$ neighbourhood.

4.3 Experimental setting

4.3.1 Instance Generation

The number of all possible solutions for a TTP instance is at most $(n-1)! \times 2^m$, which makes the enumeration and study of the standard instances impractical. Thus, an instance generator has been specially implemented to produce enumerable instances. In order to generate the local optima networks and identify the basins of attractions, we use small instances with 7 cities and 6 items (one per city, except for the starting one).

The generator has been designed following the directives in [72]. The authors computed the renting rate R (i.e., Equation 4.1) by using existing solvers to get the best picking plan for the KP [2] component and the near-optimal tour for the TSP. [3] As we are using small instances in this study, our generator obtains the renting rate for each instance by applying an exhaustive search to find the optimal solution for each component.

$$R = \frac{g(Z_{opt})}{f(X_{opt}, Z_{opt})} \tag{4.1}$$

where Z_{opt} and X_{opt} represent the optimal picking plan and the optimal tour respectively

The TSP component is fixed, i.e., the set of coordinates is the same for all the generated instances. As we are interested in two problem features, namely the knapsack capacity and profit-weight correlation, the capacity class is varied between $\mathcal{C} = 2$ and $\mathcal{C} = 10$, and all three correlation variants are considered.

Note that, for very small TTP instances, the first capacity class ($\mathcal{C} = 1$) can not be used with the *uncorrelated with similar weights* instances where the knapsack weights are ranged in $[10^3, 10^3 + 10]$. Indeed, following Equation 3.6, for $n = 7$ and $\mathcal{F} = 1$, the minimal value of W is $\frac{7}{11} \times 10^3$ which is smaller than the value of any items.

Therefore, 27 classes of the TTP are considered. For each class, 100 samples are generated with the aim of analysing their fitness landscapes.

4.4 Results and analysis

In this section, we analyse the local optima networks obtained using local search heuristics for the Travelling Thief Problem to achieve some insights about the structure of the search space. Furthermore, we study the basins of attraction and their relationship with some LON properties looking for additional information about the search difficulty.

Table 4.1: General LON and basins' statistics for the J2B heuristic.

$\mathcal{T}$	$\mathcal{C}$	$\overline{n_v}$	$\overline{n_e}$	$\overline{z}$	$\overline{C_r}$	$\overline{C}$	$\overline{l}$	π	$\overline{S}$
	2	$412.57_{430.67}$	$6820.46_{6059.62}$	$35.57_{10.78}$	$0.25_{0.25}$	$0.64_{0.14}$	$1.83_{0.33}$	1_0	1_0
unc	3	$385.29_{405.53}$	$8878.22_{9778.08}$	$41.15_{17.19}$	$0.25_{0.25}$	$0.7_{0.14}$	$1.47_{0.29}$	1_0	1_0
	4	$269.09_{316.48}$	$8816.11_{11170.62}$	$47.55_{22.84}$	$0.43_{0.31}$	$0.77_{0.12}$	$1.57_{0.32}$	1_0	1_0
	5	$144.3_{205.49}$	$4808.71_{8776.75}$	$45.85_{23.36}$	$0.59_{0.27}$	$0.82_{0.1}$	$1.4_{0.27}$	1_0	1_0
	6	$98.69_{105.59}$	$3435.64_{5583.68}$	$47.54_{26.02}$	$0.68_{0.22}$	$0.86_{0.08}$	$1.31_{0.22}$	1_0	1_0
	7	$73.39_{122.2}$	$2300.69_{6553.74}$	$39.95_{21.72}$	$0.78_{0.2}$	$0.89_{0.08}$	$1.21_{0.2}$	1_0	1_0
	8	$36.75_{29.71}$	$745.25_{1041.48}$	$28.54_{16.86}$	$0.88_{0.13}$	$0.93_{0.05}$	$1.11_{0.13}$	1_0	1_0
	9	$37.81_{33.38}$	$827.4_{1621.9}$	$28.95_{18.61}$	$0.88_{0.13}$	$0.93_{0.05}$	$1.11_{0.13}$	1_0	1_0
	10	$29.84_{27.64}$	$537.29_{979.27}$	$23.42_{14.9}$	$0.91_{0.12}$	$0.94_{0.05}$	$1.08_{0.12}$	1_0	1_0
	2	$1664.24_{598.47}$	$21720.48_{7192.31}$	$26.44_{1.06}$	0.01_0	$0.5_{0.02}$	$2.32_{0.04}$	1_0	1_0
usw	3	$3041.01_{583.12}$	$37929.45_{7130.12}$	$24.96_{0.12}$	0_0	$0.54_{0.02}$	$2.34_{0.01}$	1_0	1_0
	4	$1482.3_{1295.18}$	$60411.82_{37942.75}$	$100.55_{25.33}$	$0.19_{0.2}$	$0.63_{0.08}$	$1.88_{0.28}$	1_0	1_0
	5	$2564.18_{1216.57}$	$85908.57_{28969.87}$	$72.58_{12.66}$	$0.04_{0.03}$	$0.59_{0.04}$	$2.11_{0.11}$	1_0	1_0
	6	$392.98_{358.23}$	$30770.63_{31941.48}$	$125.66_{49.46}$	$0.46_{0.17}$	$0.75_{0.08}$	$1.53_{0.18}$	1_0	1_0
	7	$724.31_{517.63}$	$54555.59_{36874.79}$	$150.15_{11.71}$	$0.31_{0.17}$	$0.69_{0.05}$	$1.68_{0.18}$	1_0	1_0
	8	$126.41_{38.3}$	$5472.07_{3486.59}$	$80.43_{23.21}$	$0.65_{0.08}$	$0.82_{0.04}$	$1.34_{0.08}$	1_0	1_0
	9	$137.14_{77.98}$	$7707.84_{6948.17}$	$94.31_{33.58}$	$0.76_{0.12}$	$0.86_{0.05}$	$1.23_{0.12}$	1_0	1_0
	10	129.54_{17}	$5739_{1303.12}$	$87.29_{10.69}$	$0.68_{0.02}$	0.83_0	$1.31_{0.02}$	1_0	1_0
	2	$675.06_{617.18}$	$12372.51_{9253.91}$	$41.61_{18.27}$	$0.18_{0.21}$	$0.58_{0.15}$	$1.93_{0.32}$	1_0	1_0
bsc	3	$674.4_{596.53}$	$19009.72_{15837.79}$	$58.82_{17.54}$	$0.21_{0.2}$	$0.63_{0.11}$	$1.85_{0.28}$	1_0	1_0
	4	$751.39_{1068.2}$	$27887.16_{32217.37}$	$77.93_{22.55}$	$0.31_{0.25}$	$0.68_{0.13}$	$1.72_{0.31}$	1_0	1_0
	5	$461.03_{705.52}$	$20919.83_{29958.57}$	$77.61_{27.9}$	$0.46_{0.3}$	$0.76_{0.13}$	$1.54_{0.32}$	1_0	1_0
	6	$240.02_{440.31}$	$12698.73_{26078.34}$	$83.92_{29.49}$	$0.59_{0.23}$	$0.8_{0.09}$	$1.41_{0.23}$	1_0	1_0
	7	$158.74_{144.52}$	$8268.63_{11296.75}$	$83.21_{31.8}$	$0.65_{0.18}$	$0.83_{0.07}$	$1.34_{0.18}$	1_0	1_0
	8	$117.99_{112.51}$	$5651.66_{9206.02}$	$73.82_{28.99}$	$0.75_{0.15}$	$0.87_{0.06}$	$1.24_{0.15}$	1_0	1_0
	9	$73.84_{43.2}$	$2566.15_{3007.44}$	$57.54_{22.32}$	$0.84_{0.1}$	$0.9_{0.04}$	$1.15_{0.1}$	1_0	1_0
	10	$59.45_{31.52}$	$1766.42_{1773.52}$	$49.11_{20.44}$	$0.87_{0.08}$	$0.92_{0.04}$	$1.12_{0.14}$	1_0	1_0

4.4.1 Topological properties of local optima networks

Tables 4.1 and 4.2 report the average values for various network properties measured on TTP instances for the two local search heuristics. These properties are often used for LON analyses Ochoa et al. [66]. Values are averaged over 100 randomly generated instances, and subscript numbers represent the standard deviation. Each instance is represented by its corresponding LON.

The graph metrics are explained as follows. $\overline{n_v}$ and $\overline{n_e}$ represent the mean number of vertices (or nodes) and the mean number of edges over all the generated LONs respectively. $\overline{z}$ is the mean of the average degrees. $\overline{C}$ is the mean of the average clustering coefficients. $\overline{C_r}$ is the mean of the average clustering coefficients of corresponding random graphs (i.e. random graphs with the same number of vertices and mean degree). $\overline{l}$ is the mean of the average shortest path lengths between any two local optima. π is the connectivity rate, which represents the probability over the 100 samples that the LON is a connected graph. Finally, $\overline{S}$ is the mean number of non-connected components (sub-graphs).

A first insight into the search difficulty can be drawn from the values of $\overline{n_v}$ and $\overline{n_e}$. We can clearly see that the number of vertices and the number of edges generally decrease when the knapsack capacities increase for both local search variants. This implies that the hardness of search decreases when the knapsack capacity increases. Intuitively, this makes sense, as instances with very large knapsack capacities are less constrained than those with medium and particularly small capacities.

The mean average degree $\overline{z}$ increases with the capacity class, and it decreases when

2. As available at http://www.diku.dk/ pisinger/codes.html
3. Accessible at http://www.math.uwaterloo.ca/tsp/concorde/downloads/downloads.htm

Table 4.2: General LON and basins' statistics for the JIB heuristic.

$\mathcal{T}$	$\mathcal{C}$	$\overline{n_v}$	$\overline{n_e}$	$\overline{z}$	$\overline{C_r}$	$\overline{C}$	$\overline{l}$	π	$\overline{S}$
	2	$387.14_{365.26}$	9183.38_{13002}	$39.09_{25.26}$	$0.27_{0.28}$	$0.55_{0.26}$	$1.44_{0.91}$	$0.9_{0.28}$	35.85_{131}
unc	3	$376.01_{446.51}$	12933.22_{22815}	$42.69_{33.54}$	$0.36_{0.32}$	$0.66_{0.22}$	$2.92_{0.82}$	$0.93_{0.25}$	$6.47_{25.46}$
	4	$320.67_{492.12}$	12570.34_{26489}	$52.26_{33.63}$	$0.39_{0.28}$	$0.73_{0.13}$	$1.47_{0.68}$	$0.95_{0.19}$	$1.18_{1.24}$
	5	$233.42_{243.77}$	9516.94_{14459}	$57.78_{33.89}$	$0.39_{0.23}$	$0.73_{0.14}$	$1.47_{0.58}$	$0.96_{0.19}$	$1.37_{2.5}$
	6	$194.99_{174.73}$	7490.62_{10059}	$55.48_{30.93}$	$0.45_{0.26}$	$0.78_{0.09}$	$1.48_{0.44}$	$0.98_{0.14}$	$1.39_{3.8}$
	7	$126.03_{135.36}$	4494.36_{7314}	$49.7_{26.9}$	$0.57_{0.23}$	$0.82_{0.08}$	$1.42_{0.24}$	1_{0}	1_{0}
	8	$74.71_{53.4}$	2042.5_{2536}	$41.24_{21.61}$	$0.64_{0.19}$	$0.84_{0.07}$	$1.3_{0.38}$	$0.98_{0.14}$	$1.09_{0.8}$
	9	$52.66_{34.52}$	1133.33_{1289}	$34.09_{15.65}$	$0.75_{0.17}$	$0.87_{0.08}$	$1.25_{0.18}$	1_{0}	1_{0}
	10	$35.89_{24.86}$	601.03_{754}	$26.27_{11.84}$	$0.84_{0.15}$	$0.92_{0.06}$	$1.15_{0.15}$	1_{0}	1_{0}
	2	1323.35_{583}	49482.86_{27283}	$68.47_{29.66}$	$0.05_{0.02}$	$0.23_{0.11}$	$1.77_{0.98}$	$0.88_{0.32}$	62.65_{172}
usw	3	$565.16_{352.61}$	20563.04_{17740}	$58.86_{34.2}$	$0.11_{0.07}$	$0.36_{0.17}$	$1.37_{1.12}$	$0.82_{0.38}$	45.47_{95}
	4	$1347.42_{1343.72}$	73476.11_{69681}	$99.21_{44.97}$	$0.16_{0.13}$	$0.59_{0.11}$	$1.5_{0.99}$	$0.87_{0.33}$	$1.54_{1.78}$
	5	$966.26_{1025.38}$	67640.3_{72485}	$116.73_{49.69}$	$0.36_{0.29}$	$0.66_{0.16}$	$1.51_{0.78}$	1_{0}	1_{0}
	6	$477.38_{379.32}$	31049.01_{37191}	$100.73_{46.12}$	$0.24_{0.06}$	$0.72_{0.06}$	$1.76_{0.07}$	1_{0}	1_{0}
	7	$603.95_{493.46}$	53514.32_{43998}	$142.09_{65.47}$	$0.4_{0.24}$	$0.71_{0.12}$	$1.57_{0.35}$	$0.99_{0.1}$	$1.06_{0.6}$
	8	$154.05_{33.9}$	5261.68_{2411}	$66.2_{14.5}$	$0.44_{0.07}$	$0.78_{0.02}$	$1.55_{0.07}$	1_{0}	1_{0}
	9	$133.16_{82.67}$	6758.7_{6925}	$76.72_{42.27}$	$0.57_{0.12}$	$0.82_{0.05}$	$1.37_{0.36}$	$0.98_{0.14}$	$1.02_{0.14}$
	10	$93.19_{10.58}$	2590.2_{774}	$54.36_{12.14}$	$0.58_{0.08}$	$0.79_{0.02}$	$1.41_{0.08}$	1_{0}	1_{0}
	2	$460.48_{545.19}$	15032.99_{18235}	$53.76_{29.55}$	$0.27_{0.23}$	$0.59_{0.19}$	$1.77_{0.3}$	$0.96_{0.17}$	$1.47_{4.15}$
bsc	3	$586.22_{676.15}$	26870.47_{34510}	$67.94_{40.08}$	$0.31_{0.26}$	$0.62_{0.19}$	$1.55_{0.65}$	$0.94_{0.22}$	$7.27_{41.69}$
	4	$580.64_{790.3}$	32339.18_{42829}	$86.15_{45.67}$	$0.34_{0.24}$	$0.66_{0.15}$	$1.47_{0.72}$	$0.93_{0.25}$	$2.85_{11.65}$
	5	$716.18_{927.8}$	41679.47_{55873}	$89.52_{40.42}$	$0.37_{0.26}$	$0.7_{0.14}$	$1.61_{0.38}$	$0.99_{0.1}$	$1.02_{0.2}$
	6	$399.69_{566.03}$	26041.55_{42342}	$89.87_{46.88}$	$0.48_{0.26}$	$0.76_{0.11}$	$1.49_{0.36}$	$0.99_{0.1}$	$1.05_{0.5}$
	7	$255.56_{298.19}$	14855.05_{21399}	$86.04_{40.42}$	$0.53_{0.21}$	$0.79_{0.08}$	$1.46_{0.21}$	1_{0}	1_{0}
	8	$160.66_{170.85}$	8491.19_{13020}	$74.02_{38.45}$	$0.65_{0.22}$	$0.83_{0.08}$	$1.32_{0.32}$	$0.99_{0.1}$	$1.01_{0.1}$
	9	$100.06_{70.22}$	3925.34_{4615}	$62.25_{26.43}$	$0.72_{0.16}$	$0.86_{0.06}$	$1.27_{0.16}$	1_{0}	1_{0}
	10	$77.76_{42.95}$	2547.98_{2401}	$55.1_{21.02}$	$0.77_{0.13}$	$0.87_{0.06}$	$1.22_{0.21}$	1_{0}	1_{0}

the capacity class reaches 6 (7 in one case for JIB, $T = usw$).

We can also observe some so-called small-world properties by looking at the clustering coefficients ($\overline{C}$, $\overline{C_r}$) and the mean path lengths ($\overline{l}$). Firstly, the LONs show a significantly higher degree of local clustering compared with their corresponding random graphs. This means that the local optima are connected in two ways: dense local clusters and sparse interconnections, which can be difficult to find and exploit. Secondly, all the LONs have a small mean path length, i.e. any pair of local optima can be connected by traversing only few other local optima. A microscopic view on the values of the mean path length shows that it is proportional to $log(n_v)$. Therefore, a more sophisticated local search-based metaheuristics with exploration abilities, such as Tabu Search [33, 34], could move from a local optima to another within only a few iterations.

Interestingly, the connectivity rate shows that all the LONs generated using J2B are connected; while some of the LONs generated using JIB are disconnected graphs with a significantly high number of non-connected components, which is a notable disadvantage. This may not be surprising as the 2-opt and bit-flip operators, used in J2B, both induce a fully connected landscape, when considered separately for the KP and the TSP [89].

4.4.2 Degree Distributions

Figure 4.2 shows the cumulative degree distribution for some representative classes of the TTP for J2B and JIB in a log-log scale. The cumulative degree distribution function represents the probability $P(k)$ that a randomly chosen node has a degree larger than or equal to k. Though the figures show fluctuations, they allow us to deduce some interesting real-world network properties. We can see that the degree distributions decay slowly for small degrees, while their dropping rate is significantly faster for high

degrees. This behaviour indicates that the majority of local optima have a small number of connections, while a few have a significantly higher number of connections.

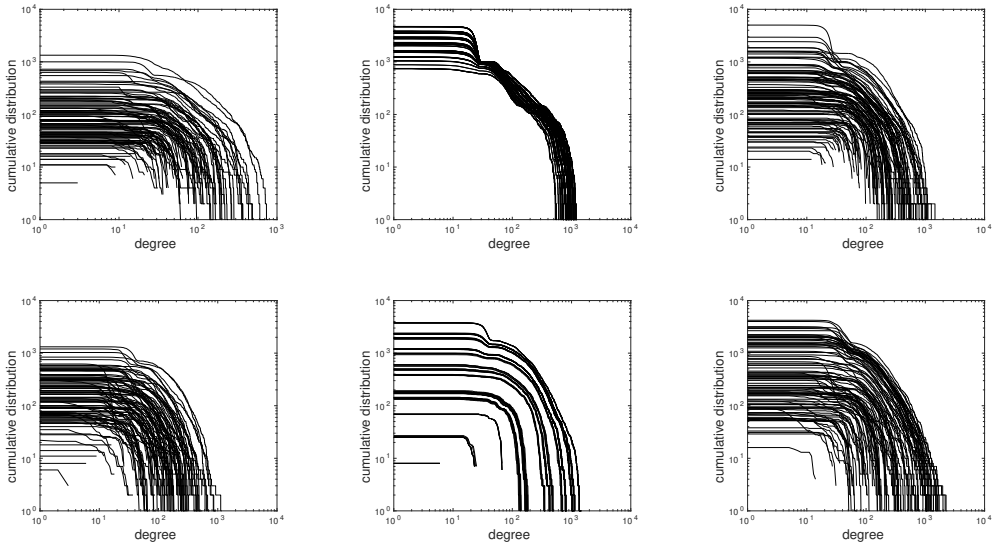


Figure 4.2: Cumulative degree distribution of J2B (top) and JIB (bottom) for $\mathcal{T} = unc$, $\mathcal{C} = 5$ (left), $\mathcal{T} = usw$, $\mathcal{C} = 5$ (middle), and $\mathcal{T} = bsc$, $\mathcal{C} = 5$ (right). All curves are shown in a log-log scale.

Most of the real-world networks have their topological structure more accurately described by a power-law or a scale-free degree distribution $P(k) = k^{-\alpha}$, where $\alpha \in [2, 3]$ is a scaling parameter. The behaviour of local search strategies on networks has been studied according to the degree distribution [1]. Intuitively, the degree distribution allows one to search a power-law graph more rapidly, relying on the fact that the number of edges per node varies considerably from node to node, i.e., its edges do not let us uniformly sample the graph, but they preferentially lead to high degree nodes. This again supports our conjecture about the existence of few nodes with high degree that efficiently connect the entire landscape. A local search algorithm has more chances to move to one of these high degree nodes than to other random nodes, which allows the algorithm to efficiently browse the entire graph.

With the aim of performing a rigorous study on the cumulative degree distributions, we use the Kolmogorov-Smirnov test to investigate the adequacy of power-law [10] and exponential models [20]. The test is performed on all 100 samples for each TTP class shown in Figure 4.2, and the rates at which the test fails to reject the null hypothesis (that the data come from the considered distribution model) are shown in Table 4.3.

Although J2B was able to produce LONs with degree distributions that fit a power-law distribution for many instances, a power-law can not be generalised as a plausible model to describe the degree distribution for all the landscape.

Given this fact, another possibility is to look into fitting the degree distribution to an exponential model of the form $P(k) = \frac{e^{-k/z}}{z}$. This model was proposed by Ochoa et al. [66] to describe the degree distributions for NK models, with tunable problem difficulties. Looking at Figure 4.2, especially the curves produced by JIB, an exponential model seems to be a good alternative. Table 4.3 shows that the Kolmogorov-Smirnov always fails to reject the exponential distribution as a plausible model for all the samples

considered.

Table 4.3: The rates at which the Kolmogorov-Smirnov test fails to reject power-law and exponential as plausible distribution models, with a significance level of 0.1

		T=unc, C=5	T=usw, C=5	T=bsc, C=5
Power-law	J2B	0.22	1	0.53
	JIB	0.39	0.26	0.46
Exponential	J2B	1	1	1
	JIB	1	1	1

4.4.3 Basins of attraction

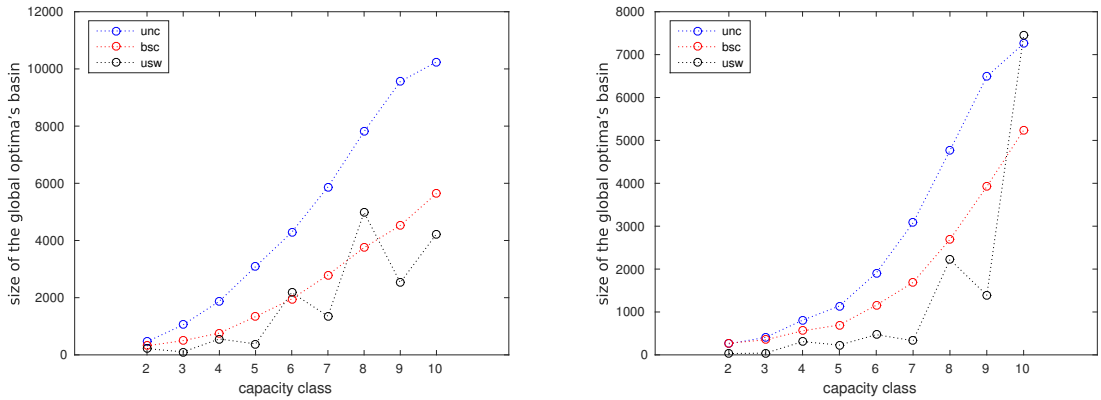


Figure 4.3: Average of the relative size of the basin corresponding to the global maximum for each $\mathcal{C}$ over the 100 TTP instances for J2B (left) and JIB (right).

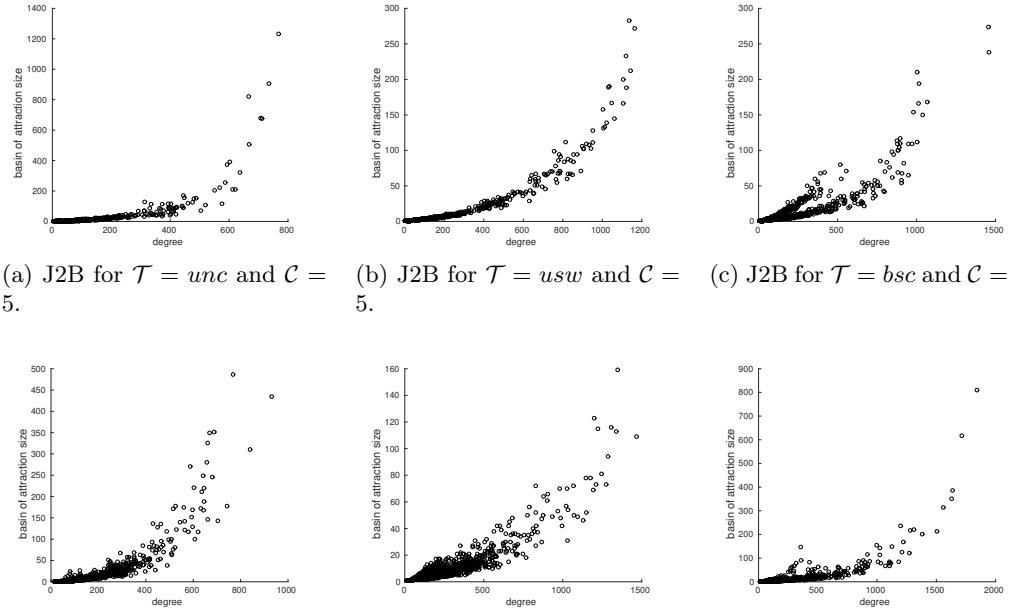
Exponential degree distributions do not provide a straightforward interpretation of the local search strategies' behaviour as power-law does. Therefore, an alternative approach to analyse the difficulty of the search space for the considered heuristics is to look into the size of the basins of attraction. In Figure 4.3 we illustrate the mean sizes of global optimum basins over all the instances for all the capacity classes. The plots show that the basin size generally increases when the capacity class increases.

This correlation is more thoroughly examined and generalized to the mean size of all basins of attraction for some representative instances as shown in Figure 4.4. We can clearly see a correlation between the degrees and the basin sizes.

Next, we take a look into the relationship between the fitness of local optima and their basin size. Figure 4.5 illustrates this relationship for representative instances for the two heuristics.

For J2B, there is a clear positive correlation between the fitness of local optima and their basin size as the size of the basin increases when the fitness value increases. The fact that the solutions having the highest fitnesses belong to large basins makes them more likely to be found using the addressed local search techniques. Furthermore, we can also observe that the number of good quality local optima is smaller compared to low quality local optima.

On the other hand, it is difficult to extract a pattern for JIB due to the high amount



(a) J2B for $\mathcal{T} = unc$ and $\mathcal{C} = 5$.

(b) J2B for $\mathcal{T} = usw$ and $\mathcal{C} = 5$.

(c) J2B for $\mathcal{T} = bsc$ and $\mathcal{C} = 5$.

(d) JIB for $\mathcal{T} = unc$ and $\mathcal{C} = 5$.

(e) JIB for $\mathcal{T} = usw$ and $\mathcal{C} = 5$.

(f) JIB for $\mathcal{T} = bsc$ and $\mathcal{C} = 5$.

Figure 4.4: Correlation between the degree of local optima and their corresponding basin sizes on representative examples.

of volatility at which the fitnesses are scattered relatively to the basin sizes. Therefore, it is safe to assume that the probability that JIB will identify a global optimum is quite low.

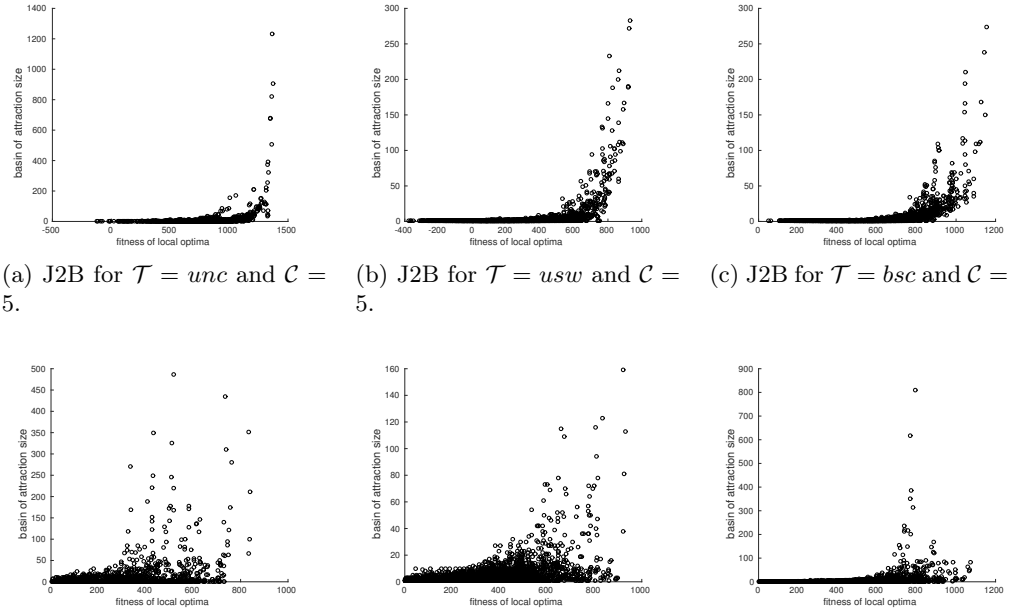
At first sight, it is not clear how this information can be used in a heuristic. However, if techniques like self-adaptation and restarts are used in combination with J2B, then the progress achieved over time can be used in online control as an indicator for the expected achievable solution quality.

Also, if the same local optimum is found over and over again from random starting points, we conjecture that such restarts are not necessary for J2B, whereas exploring these observations is quite difficult in the case of JIB.

From a comparison point of view, J2B seems to be a better local search approach compared to JIB. In a sense, this is not surprising as most well performing heuristics for the TTP, including memetic algorithms, apply the *2-OPT* and *bit-flip* operators in their algorithms mainly to improve their exploitation ability [60, 26, 87]. Note that we are not dismissing the *insertion* operator. In fact, the *insertion* move has been commonly adopted as a disruptive operator in state-of-the-art TSP solvers [50].

4.5 Conclusion

Based on the reported results, we concluded that instances with high knapsack capacities might be easier to solve. The results also showed that the investigated LONs have two small-world properties: strong local clustering and a small mean path length. These properties suggest the existence of dense local connections between some local



(a) J2B for $\mathcal{T} = unc$ and $\mathcal{C} = 5$.

(b) J2B for $\mathcal{T} = usw$ and $\mathcal{C} = 5$.

(c) J2B for $\mathcal{T} = bsc$ and $\mathcal{C} = 5$.

(d) JIB for $\mathcal{T} = unc$ and $\mathcal{C} = 5$.

(e) JIB for $\mathcal{T} = usw$ and $\mathcal{C} = 5$.

(f) JIB for $\mathcal{T} = bsc$ and $\mathcal{C} = 5$.

Figure 4.5: Correlation between the fitness of local optima and their corresponding basin sizes on representative examples.

optima, and that almost any two local optima are connected with few local search iterations. Furthermore, some LONs built using JIB encompass disconnected sub-graphs, while the LONs generated using J2B are always connected, which in turn may make the instances easier to solve for J2B.

In addition to the LON topological properties, the cumulative degree distribution was analysed for a representative set of instances. The Kolmogorov-Smirnov test was used to investigate the adequacy of a power-law and an exponential model. The test showed that, differently from the exponential model, the power-law can not be generalised for all the studied LONs. However, many LONs showed a scale-free behaviour as the Kolmogorov-Smirnov test failed to reject the power-law as a plausible distribution.

The study of attraction basins confirmed that the difficulty of instances drops significantly when the knapsack capacity increases. Finally, the investigation of the relationship between the fitness of local optima and their basin size shows that J2B has better chances to identify the global optima compared to JIB.

The work in this chapter has been accepted for publication in the proceedings of the Genetic and Evolutionary Computation Conference (GECCO) in 2018 [31].

CHAPTER 5

Initialisation, local search heuristics, and multi-objective model

In this chapter, we start by studying solution initialisation approaches for the TTP and their impact on the search process. Then we introduce an initialisation strategy based on heuristics and greedy algorithms. Afterwards, we propose two neighbourhood-based local search heuristics and compare their performances to two state-of-the-art algorithms. The proposed approaches create a TTP-specific neighbourhood that is generated by combining a TSP neighbourhood and a KP neighbourhood. The combination is achieved by applying a cartesian product between the component neighbourhoods. Finally, we investigate the TTP as a bi-objective problem by considering traveling time and profit as the objectives.

5.1 Solution initialisation

5.1.1 On the impact of initialisation

Initial tour

We compare two tour initialization methods. The first approach uses a naive random algorithm to generate a tour. The second uses the Lin-Kernighan heuristic to find a good TSP tour.

In figure 5.1, we illustrate the solution objective value and runtime of 14 different TTP instances of small and medium size. The used KP type is *bounded strongly correlated*, the knapsack category is 06, and the item factor is 3 [70]. Note that this experiment uses the JNB algorithm.

Note that the results shown are neither dependent on the used iterative algorithm, nor the TTP parameters (KP type, knapsack category, and item factor). We have omitted additional results for the sake of simplicity. In addition, we use a log scale to represent the runtimes in order to improve the plots visibility.

It is obvious that the Lin-kernighan initialization technique is far better than the random tour method. As shown above, the results obtained starting with a Lin-Kernighan

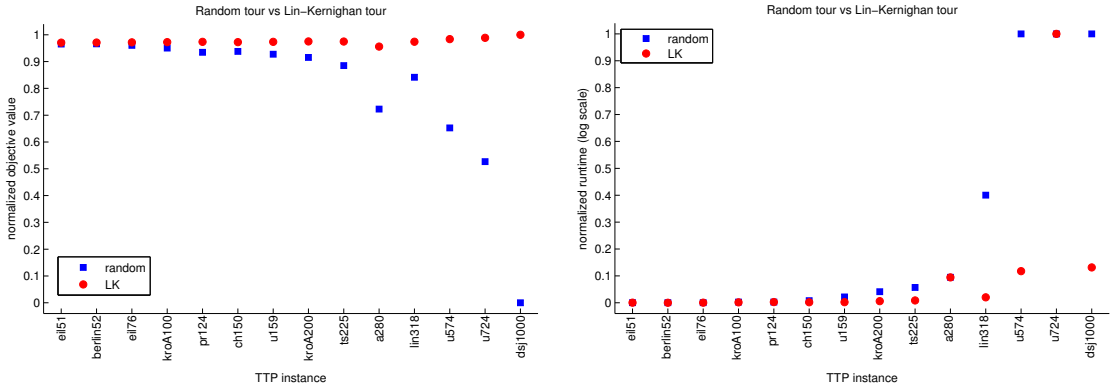


Figure 5.1: Comparison of Lin-Kernighan initial tour and random tour in terms of solution quality (left) and runtime (right)

tour have far better objective values and runtimes.

Initial picking plan

To illustrate the influence of the starting picking plan, we use three initialization strategies: empty knapsack, random picking plan, and a greedy algorithm. These tests were performed using the same set of data presented above.

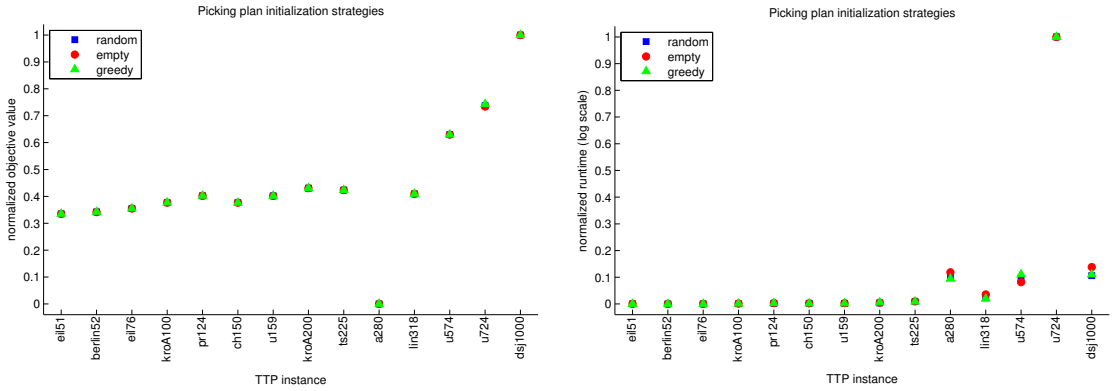


Figure 5.2: Comparison between a random, an empty, and a greedy picking plan in terms of solution quality (left) and runtime (right)

Figure 5.2 illustrates the influence of the initial picking plan. As shown, clearly the initial picking plan has very insignificant influence on the solution quality and runtime. The standard deviation is 0.23% for objective values, and 16.25% for runtimes.

It is clear that the choice of the initial tour has a significant influence on the search process. Tours with a short distance help the iterative algorithm to find a good solution quickly. Whereas the initial picking plan have very slight influence on the solution quality and the runtime.

5.1.2 An initialisation strategy

In the TTP, the initial solution has a big impact on the search algorithm – especially the initial tour. In fact, most current algorithms use a sophisticated heuristic to generate an initial tour (Lin-Kernighan heuristic, Boruvka algorithm, 2-opt, etc.). Thus, the initialisation strategy should be chosen carefully. In our implementation, we also use a heuristic approach to initialise both the tour and the picking plan. Firstly, the initial tour is generated using the Concorde's implementation of the Lin-Kernighan heuristic [2, 53]. Then, the picking plan is initialised using the following approach:

1. The insertion heuristic proposed by Mei et al. [60] is used to fill the knapsack with items. This technique uses three fitness approximations to select the most profitable items.
2. A simple bit-flip search is then applied on inserted items to eliminate some useless items from the picking plan using the objective function.

We will refer to this heuristic as the *insertion & elimination heuristic*.

5.2 Joint neighbourhood algorithms

Joint neighbourhood algorithms are based on the simple idea of combining component neighbourhoods. Herein, we propose an approach to implement neighborhood-based deterministic local search algorithms to solve small and mid-size TTP instances.

5.2.1 JNB: Joint N1-BF

JNB is a neighbourhood-based heuristic that combines the TSP's N1 neighbourhood and KP's one-bit-flip. N1 consists of swapping two adjacent cities. Whereas one-bit-flip consists of toggling the state of an item, i.e. picking the item if it is not picked and unpacking it otherwise. The generated neighbourhood is browsed in order to select a better or the best solution of the set depending on the search strategy.

As in other neighbourhood based algorithms, two versions can be implemented using the same neighbourhood. The first version is the hill climbing algorithm (will be referred to as first-fit), which stops browsing the neighbourhood when a better solution is found. The second version is the steepest ascent (will be referred to as best-fit), which browses the whole neighbourhood in order to select the best solution.

As explained in chapter 4, it is practically impossible to calculate the total travelling time of a neighbour solution in a constant time when a bit is flipped in the picking plan. The best that can be done is to recalculate the value of f starting from the city where the bit-flip happened.

Figure 5.3 illustrates how the objective value of a neighbour solution is recovered in TTP using the JNB algorithm, and a comparison to equivalent operations for KP and TSP. Note that, in the context of KP, the bit-flip region (in black) corresponds to the item's index in the picking plan. Whereas in the TTP, it represents the index of the city from which the item is picked. In addition, $d(i)$ represents the distance between the cities $x(i)$ and $x(i+1)$, $y(i)$ is the i'th item, and $p(i)$ is the profit of the i'th item.

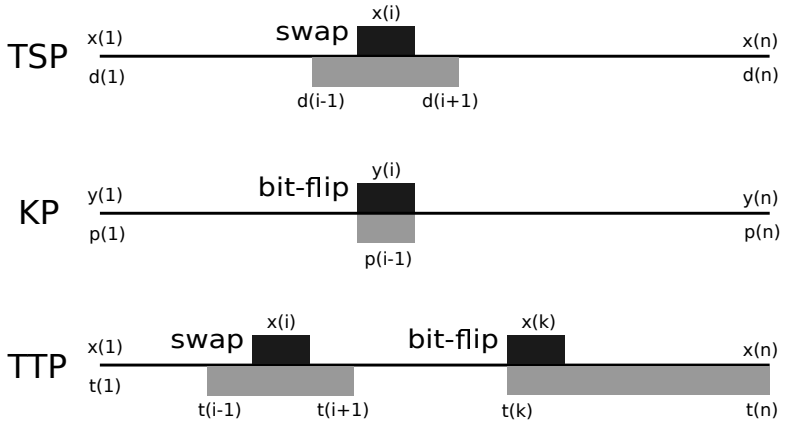


Figure 5.3: Swap applied to a TTP solution and its effect on the total travelling time function, compared to changes in TSP and KP

The recovery operation also needs to keep track of the weight and time accumulation at each city of the tour. For this purpose, we propose three vectors to associate with each TTP solution:

— **Time accumulator (t^{acc}):** a vector that contains the current time at each city of the tour.

— **Weight accumulator (w^{acc}):** a vector that contains the current weight at each city of the tour.

— **Tour mapper (t^{map}):** since each city has a unique integer reference between 1 and n, t^{map} represents a vector that associates the index (position of the city in the tour) to the reference of city. For instance, let x be a tour, the t^{map} associated to x is obtained as shown below:

```
index :  1, 2, 3, 4, 5, 6
x     :  1, 3, 5, 4, 6, 2
t^map :  1, 6, 2, 4, 3, 5
```

The tour mapper is useful to get the position where an item is picked easily instead of browsing tour cities.

These vectors are recalculated for each solution at each heuristic iteration. It would be convenient to associate them to the objective function.

Algorithm 2 JNB (first-fit version)

```
1  x ← initial tour                                   ▷ we always use a Lin-Kernighan tour
2  z ← initial picking plan
3  repeat
4      Evaluate G, t, and p for (x, z)
5      Update t^map, t^acc, and w^acc
6      for i ← 2 ... n − 1 do
7          x̄ ← swap(x, i)
8          t̄^map ← tour mapper for (x̄, z)
9          t̄^acc ← recover time accumulator for (x̄, z)
10         w̄^acc ← recover weight accumulator for (x̄, z)
11         for k ← 1 ... m do
12             z̄ ← bitflip(z, k)
13             if no space left then skip iteration end if
14             calculate Δ_w and Δ_p
15             p̄ ← p + Δ_p
16             i_BF ← t^map_{A_k}                      ▷ index of bit-flip
17             if i_BF = 0 then t̄ ← 0  else t̄ ← t̄^acc_{i_BF − 1} end if
18             for r ← i_BF ... n do
19                 w_c ← w̄^acc_r + Δ_w
20                 t̄ ← t̄ + d_{x̄_r, x̄_{r+1 mod n}} / (v_max − w_c * C)
21             end for
22             Ḡ ← p̄ − R * t̄
23             if Ḡ > G then break loop end if          ▷ first fit
24         end for
25         if Ḡ > G then break loop end if              ▷ first fit
26     end for
27     if Ḡ > G then
28         x ← x̄
29         z ← z̄
30     end if
31 until Ḡ ≤ G
```

Algorithm 2 starts by initializing the tour and picking plan in lines 1 and 2. In our implementation we always use a Lin-Kernighan [53] tour as the initial TTP tour. We use three different techniques to generate the initial picking plan: no items picked, greedy algorithm, and random picking plan. Once the solution is initialized, the objective function calculates the total travel gain G, the travel time t, the total items profit p, and the three vectors t^{acc}, w^{acc} and t^{map} introduced above are updated in line 4 and 5. Then, a swap is applied between two adjacent cities of the tour to generate a neighbour solution, and the vectors $\bar{t^{acc}}$, $\bar{w^{acc}}$ and $\bar{t^{map}}$ associated to the solution $(\bar{x}, z)$ are recovered from the original solution, see lines 6 to 10. For each swapped tour, we associate a bit-flipped picking plan, we ensure that there is space left in the knapsack if an item is picked, and we recover the profit p using the delta technique in line 15. Then, the total time is partially recalculated starting from the position of bit-flip i_{BF} in lines 16 to 21. Finally, in lines 22 to 31, the new total gain is calculated and compared to the current gain, if it improves it we keep the new solution and the process is iterated, otherwise the algorithm stops and the current solution is saved.

Therefore, the worst-case complexity (i.e. bit-flips start in the first city) for one

iteration of JNB algorithm is as follows:

$$\begin{aligned} Comp(JNB) =& Comp(evaluation) + Comp(update) + \\ & Comp(swap) * Comp(bitflip) * Comp(backup) \\ =& O(m * n) + O(m * n) + O(n) * O(m) * O(n) \\ =& O(m * n^2) \end{aligned}$$

where:

— $Comp(evaluation)$ is the complexity of evaluating a TTP solution (line 4).
— $Comp(update)$ is the complexity of updating the associated vectors (line 5).
— $Comp(swap)$ is the swap's complexity (line 6).
— $Comp(bitflip)$ is the bitflip's complexity (line 11).
— $Comp(backup)$ is the complexity of the backup operation applied to the neighbour solution to retrieve the travel time (line 18).

As a result of interdependence between picking and travelling, the backup operation is introduced to partially recalculate the travel time. This operation has a linear worst-case complexity. Nevertheless, the recovery starts from the city of bit-flip instead of the first city which make the algorithm faster.

5.2.2 J2B: Joint 2opt-BF

J2B (standing for Joint 2opt-BF) is a neighborhood-based heuristic based on the same technique proposed in the previous subsection. This heuristic combines the TSP's 2-OPT neighborhood and KP's one-bit-flip. 2-OPT is an operator that consists of inverting a sub-tour of a given TSP tour.

As in JNB, a cartesian product is applied between these two neighborhoods to generate a neighborhood specific to the TTP. Then, the generated neighborhood is browsed in order to select a better (or the best) solution of the set.

The process of recovering the objective value is illustrated in figure 5.4. The figure shows that the recovery is more time complex as the 2-OPT (applied between x_i and x_j) affects all the terms associated to the cities between x_i and x_j. The reason for this is that the non-symmetry of the time function, i.e. $t(c_1, c_2) \neq t(c_2, c_1)$ such as t is the time function, and c_1, c_2 are two given cities.

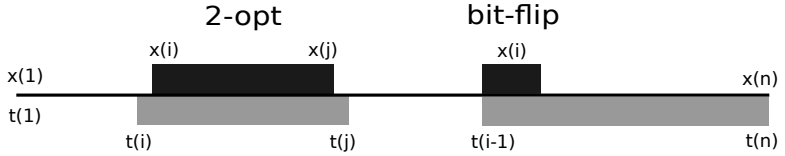


Figure 5.4: 2-OPT applied to a TTP solution and its effect on the total travelling time function

In this heuristic, two more history vectors are needed:

— **Time register (t^{reg}):** a vector that contains the added time at each city of the tour.
— **Weight register (w^{reg}):** a vector that contains the added weight at each city of the tour.

Algorithm 3 J2B (first-fit version)

```
1  x ← initial tour                                   ▷ we always use a Lin-Kernighan tour
2  z ← initial picking plan
3  repeat
4      Evaluate G, t, and p for (x, z)
5      Update t^map, t^acc, w^acc, t^rec, and w^rec
6      for i ← 2 ... n − 1 do
7          for j ← i + 1 ... n do
8              x̄ ← do2opt(x, i, j)
9              t̄^map ← tour mapper for (x̄, z)
10             t̄^acc ← recover time accumulator for (x̄, z)
11             w̄^acc ← recover weight accumulator for (x̄, z)
12             t̄^rec ← recover time register for (x̄, z)
13             w̄^rec ← recover weight register for (x̄, z)
14             for k ← 1 ... m do
15                 z̄ ← bitflip(z, k)
16                 ŵ^rec ← recover weight register for (x̄, z̄)
17                 if no space left then skip iteration end if
18                 calculate Δ_w and Δ_p
19                 p̄ ← p + Δ_p
20                 i_BF ← t^map_{A_k}                              ▷ index of bit-flip
21                 if i_BF = 0 then t̄ ← 0  else t̄ ← t̄^acc_{i_BF−1} end if
22                 if i_BF = 0 then w_c ← 0  else w_c ← w̄^acc_{i_BF−1} end if
23                 for r ← i_BF ... n do
24                     w_c ← w_c + ŵ^acc_r + Δ_w
25                     t̄ ← t̄ + d_{x̄_r, x̄_{r+1 mod n}} / (v_max − w_c ∗ C)
26                 end for
27                 Ḡ ← p̄ − R ∗ t̄
28                 if Ḡ > G then break loop end if                 ▷ first fit
29             end for
30             if Ḡ > G then break loop end if                     ▷ first fit
31         end for
32         if Ḡ > G then break loop end if                         ▷ first fit
33     end for
34     if Ḡ > G then
35         x ← x̄
36         z ← z̄
37     end if
38 until Ḡ ≤ G
```

Algorithm 3 starts by initializing the tour and picking plan using the same technique as in JNB in lines 1 and 2. Once the solution is initialized, the objective function calculates the travel gain G, the travel time t, the total items profit p, t^{map} and the four history vectors introduced above in lines 4 and 5. Then, a 2-OPT is applied between two cities of the tour to generate a neighbor solution and t^{map}. The history vectors are also recovered for this solution in lines 6 to 13. For each obtained tour, we associate a bit-flipped picking plan, and the w^{rec} vector is recovered for the new solution. We ensure that there is space left in the knapsack if an item is picked. Therefore, we recover the profit p using the *delta technique* in lines 14 to 19. Then, the total time is partially recalculated starting from the position of bit-flip i_{BF} in lines 20 to 26. Finally, in lines 27 to 38, the new total gain is calculated and compared to the current gain. If the new gain improves the current one, we keep the new solution and the process is iterated. Otherwise, the algorithm stops and the current solution is returned.

Therefore, the worst-case complexity (i.e. bit-flips start in the first city) for one iteration of J2B algorithm is as follows:

$$\begin{aligned} Comp(J2B) =& Comp(evaluation) + Comp(update) + \\ & Comp(2opt) * Comp(bitflip) * Comp(backup) \\ =& O(m * n) + O(m * n) + O(n^2) * O(m) * O(n) \\ =& O(m * n^3) \end{aligned}$$

where $Comp(2opt)$ is the 2-OPT complexity (line 6).

This algorithm has clearly a higher time complexity due to the use of 2-OPT neighborhood.

Note that the *do2opt* function at line 8 is used for the sake of algorithm readability. In practice, the swap is not applied until the solution is accepted. Instead of applying the 2-OPT exchange, the original solution is used to access neighbor's cities in a constant time using the *get2opt* function in algorithm 4. The function has four parameters: k represents the required city index, x is the tour, i and j are the 2-OPT indices.

Algorithm 4 Access 2-OPT elements

```
1 function GET2OPT(k, x, i, j)
2     if k ≥ i & k ≤ j then
3         return x_{j−k+i}
4     end if
5     return x_k
6 end function
```

5.2.3 Experiments and results

In this subsection, our two algorithms are compared side-by-side with RLS and EA proposed in [70]. We perform tests on various TTP instances from the benchmark suite introduced in the same paper. These instances are selected by varying the TTP parameters as follows:

- **TSP base problems:** eil51, berlin52, eil76, kroA100, pr124, ch150, u159, kroA200, ts225, a280, lin318, u574, u724, dsj1000, rl1304
- **KP types ($\mathcal{T}$):** uncorrelated (unc), uncorrelated with similar weights (usw), bounded strongly correlated (bsc)
- **Knapsack categories ($\mathcal{C}$):** 06, 10
- **Item factor ($\mathcal{F}$):** 1, 10

By combining these parameters, we obtain 180 TTP instances. Note that, the RLS and EA tests were performed on a different machine (Matlab 2014, core i7 2600 CPU 3.4 GHz, with 4GB of RAM, running Windows 7). The runtimes for these algorithms are given for guidance. Also, due to the random behaviour of These algorithms, 30 independent runs were performed. We select the best objective value for our comparison.

On the other hand, we use a Lin-Kernighan initial tour as it is the most efficient initialization strategy, and the greedy algorithm to initialize the picking plan. Since JNB and J2B are deterministic, one run per instance is adequate.

The results are presented in tables 5.1 to 5.6. Note that the runtimes are measured with seconds, the best objective values are highlighted in blue, and the negative ones

are highlighted in gray.

Table 5.1: Results for the instances with 1 item per city and uncorrelated item weights/profits ($\mathcal{F} = 1$, $\mathcal{T}$=unc)

	Medium knapsack capacity ($\mathcal{C}$=06)								High knapsack capacity ($\mathcal{C}$=10)							
	EA		RLS		JNB		J2B		EA		RLS		JNB		J2B	
	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time
eil51	3313	1.23	3313	0.85	4744	0.05	4604	0.25	6130	1.39	6130	0.78	7481	0.01	7446	0.05
berlin52	5372	1.25	5372	0.77	5342	0.02	5342	0.09	7980	1.09	7980	0.69	8030	0	8030	0.08
eil76	6176	0.99	6176	0.74	8286	0.01	8519	0.33	8826	0.93	8826	0.84	11196	0.01	11196	0.32
kroA100	9070	1.13	9070	1.02	10468	0.02	10509	2.19	14151	0.99	14151	0.75	15017	0.03	15017	1.45
pr124	16767	1.09	16767	1.37	15378	0.06	15360	1.09	18546	1.12	18546	0.99	17358	0.04	19915	3.86
ch150	14768	1.24	14768	0.8	15201	0.04	15929	19	20171	1.26	20171	0.9	20670	0.04	21343	21
u159	20546	1.14	20546	0.86	20565	0.05	20565	4.87	24885	1.13	24885	1.05	24918	0.08	24918	6.92
kroA200	25450	1.27	25450	0.95	22889	0.15	22889	6.39	30673	1.41	30673	1.05	28404	0.16	28408	6.37
ts225	29451	2.28	29451	0.91	26873	0.23	27025	49	33949	1.19	33949	0.89	31612	0.3	31721	54
a280	36140	1.44	36140	0.94	36865	1.87	36656	54	41358	1.41	41358	1.04	42288	2.5	35547	95
lin318	26310	1.87	26310	1.2	26479	0.46	31058	245	37573	1.76	37573	1.07	37764	0.69	41489	320
u574	67073	2.55	67073	1.27	68361	2.86	68361	600	88101	2.53	88101	1.53	89499	3.14	89497	600
u724	85406	2.91	85406	1.94	86828	7.94	86786	600	109681	2.95	109681	1.93	111158	16	110995	600
dsj1000	76040	5.29	76040	2.09	81296	12	81296	600	150403	5.29	150403	2.16	157383	11	157383	600
rl1304	148740	7.25	148740	3.72	146608	149	144970	600	205093	8	205093	3.25	203860	189	200213	600

Table 5.2: Results for the instances with 1 item per city and uncorrelated but similar item weights ($\mathcal{F} = 1$, $\mathcal{T}$=usw)

	Medium knapsack capacity ($\mathcal{C}$=06)								High knapsack capacity ($\mathcal{C}$=10)							
	EA		RLS		JNB		J2B		EA		RLS		JNB		J2B	
	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time
eil51	1933	0.79	1933	0.79	2929	0	2929	0.08	5420	1	5420	0.85	6519	0.01	6519	0.04
berlin52	3848	1.12	3848	0.81	3856	0	3856	0.05	7232	1	7232	0.69	7297	0	7297	0.07
eil76	1602	1.67	1602	0.81	3993	0.01	3993	0.37	7808	1.23	7808	0.79	10315	0.01	10315	0.35
kroA100	9925	1.39	9925	0.75	9481	0.02	9514	1.08	13880	1.04	13880	1.03	13911	0.03	13926	1.09
pr124	12852	1.14	12852	0.97	10733	0.05	12244	4.6	17943	1.13	17943	0.97	15806	0.07	18941	4.28
ch150	12112	1.47	12112	0.94	12590	0.04	13470	24	18446	1.18	18446	0.79	19023	0.05	19023	4.77
u159	15223	1.67	15223	0.81	15243	0.08	15243	6.79	22772	1.13	22772	1.08	22803	0.06	22803	4.92
kroA200	19921	1.19	19921	1.08	17358	0.14	18581	14	27802	1.18	27802	1.01	25041	0.17	25041	12
ts225	18562	1.29	18562	0.97	17210	0.32	17210	58	31365	1.45	31365	0.98	30895	0.31	30895	32
a280	22469	1.37	22468	1.03	23363	2.01	22896	245	38789	1.54	38789	0.97	39859	2.4	29142	184
lin318	16285	1.49	16285	1.08	16517	0.94	22122	258	33797	1.55	33797	1.11	34057	0.61	36909	583
u574	52481	2.62	52481	1.4	53711	2.97	53711	600	79541	2.47	79541	1.8	81008	1.95	81008	427
u724	59902	3.59	59902	1.9	61964	14	61934	600	100330	3.15	100330	1.56	102892	21	102713	600
dsj1000	41167	6.35	41167	2.6	47885	13	47885	600	149660	5.33	149660	2.28	155635	13	155635	600
rl1304	124420	6.83	124420	4.14	119749	98	117341	600	190956	6.91	190956	2.98	188811	197	185209	600

Table 5.3: Results for the instances with 1 item per city and bounded strongly correlated item weights/profits ($\mathcal{F} = 1$, $\mathcal{T}$=bsc)

	Medium knapsack capacity ($\mathcal{C}$=06)								High knapsack capacity ($\mathcal{C}$=10)							
	EA		RLS		JNB		J2B		EA		RLS		JNB		J2B	
	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time
eil51	5899	0.94	5899	0.79	6649	0.01	6773	0.37	9613	1.04	9613	0.88	11439	0.01	11750	0.11
berlin52	7900	1.04	7900	0.75	7537	0	11896	0.45	7875	0.86	7875	0.78	7993	0	13107	0.73
eil76	9961	1.73	9961	0.81	12215	0.01	12215	0.43	10033	1.06	10033	0.73	14862	0.01	14862	0.58
kroA100	14759	1.01	14747	0.85	18652	0.02	18731	1.35	14762	1.02	14762	0.84	20219	0.03	20219	1.66
pr124	16410	1.79	16410	0.9	25081	0.12	25122	1.44	21770	1.4	21770	0.88	27247	0.05	27277	1.46
ch150	17915	1.28	17915	0.99	18721	0.07	19070	17	17149	1.44	17149	0.81	18422	0.09	18422	9.64
u159	34121	1.16	34119	0.86	34134	0.1	34134	11	37359	1.22	37359	0.93	37448	0.12	39131	19
kroA200	43307	1.25	43307	1.04	40746	0.22	40660	9.08	39016	1.44	39016	1.22	38155	0.3	38179	15
ts225	47606	1.21	47606	0.99	48556	0.55	48556	109	48921	1.29	48921	0.9	51833	0.34	51833	36
a280	44995	2.3	44989	1.09	46478	4.33	42780	486	51302	1.45	51302	1.03	53828	4.46	58121	600
lin318	37323	4.12	37298	1.07	37225	1.21	56344	600	42959	2.4	42959	1.22	43371	1.17	56679	600
u574	110444	2.58	110444	1.31	111935	2.73	111935	600	131604	2.38	131604	1.41	136156	3.26	136156	603
u724	136892	3.44	136890	1.83	133736	25	133038	600	148256	3.55	148256	1.7	146040	47	144040	600
dsj1000	328978	6.4	328979	2.37	332572	14	332544	600	293069	6.37	293069	2.6	306279	11	306279	600
rl1304	291699	8.23	291700	3.51	263521	342	254820	600	303556	6.84	303556	4.66	282758	368	268560	600

Table 5.4: Results for the instances with 10 items per city and uncorrelated item weights/profits ($\mathcal{F} = 10$, $\mathcal{T}$=unc)

	Medium knapsack capacity ($\mathcal{C}$=06)								High knapsack capacity ($\mathcal{C}$=10)							
	EA		RLS		JNB		J2B		EA		RLS		JNB		J2B	
	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time
eil51	44624	2.63	44624	1.08	55061	1.62	54986	0.86	62588	1.8	62588	1	73734	1.49	73763	24
berlin52	67241	1.8	67241	0.94	67690	0.67	67690	31	86050	1.87	86050	1.08	86543	0.66	86543	32
eil76	58616	2.81	58616	1.2	78400	1.75	78400	124	85545	2.67	85545	1.14	107858	2.28	107858	161
kroA100	107974	3.14	107974	1.5	115983	4.76	118105	163	148491	3.14	148491	1.43	150959	4.19	156239	241
pr124	142013	4.13	142013	1.96	139648	7.4	143382	11	185933	4.53	185933	2.17	180553	9.03	186667	10
ch150	126962	6.33	126962	1.96	131701	16	126730	600	188935	5.82	188935	1.82	194376	17	188093	600
u159	188111	5.28	188111	2.31	188404	11	178250	603	242260	6.24	242260	2.29	242539	15	222626	600
kroA200	225584	8.81	225584	3.02	226805	41	227358	65.5	294118	9.93	294118	2.7	296686	41	297184	66
ts225	264424	9.87	264424	3.49	262014	32	139575	600	343983	12	343983	2.93	339645	43	186545	600
a280	305734	17	305734	4.2	70435	500	12371	600	411715	17	411715	3.66	330510	435	27373	600
lin318	275033	26	275033	4.64	258410	71	125336	600	398115	20	398115	5.21	371790	82	-14516	600
u574	681934	69	681934	15	693160	600	692634	600	924203	71	924203	13	937133	600	936234	600
u724	863165	104	863165	17	810622	600	790940	599	1157344	98	1157344	17	1045846	600	1025704	600
dsj1000	1161151	182	1161151	35	1190676	582	1190177	600	1411451	188	1411451	39	1459170	600	1458413	600
rl1304	1666972	319	1666972	72	1154690	600	1117255	600	2183617	275	2183617	67	1558021	600	1514964	600

Table 5.5: Results for the instances with 10 items per city and uncorrelated but similar item weights ($\mathcal{F} = 10$, $\mathcal{T}$=usw)

	Medium knapsack capacity ($\mathcal{C}$=06)								High knapsack capacity ($\mathcal{C}$=10)							
	EA		RLS		JNB		J2B		EA		RLS		JNB		J2B	
	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time
eil51	30880	1.59	30880	1.4	40851	2.37	41248	0.78	52794	2.44	52794	1.06	65425	2.08	65425	84
berlin52	46584	1.72	46584	0.98	46998	1.23	46998	57	79072	1.77	79072	0.99	79530	1.37	79530	66
eil76	41667	2.58	41667	1.21	63035	3.33	63035	242	77510	2.54	77510	1.31	101126	2.71	101126	193
kroA100	88765	2.81	88765	1.87	88561	9.11	93555	396	133914	3.04	133914	1.71	136263	6.46	141826	401
pr124	107625	4.8	107625	2.47	103652	21	23640	600	172977	4.57	172977	1.81	162537	17	125860	600
ch150	91650	7.37	91650	1.75	96605	21	90455	600	170256	4.78	170256	1.82	176079	17	169410	600
u159	128974	6.32	128974	2.95	129279	19	77204	600	217592	8.14	217592	2.71	217931	15	188954	600
kroA200	184058	9.21	184058	2.43	176746	41	177166	68	271192	9.58	271192	2.63	267436	49	268353	66
ts225	188464	11	188464	2.8	186519	56	-156846	600	308174	10	308174	3.18	310089	64	3186	600
a280	229256	21	229256	3.94	53420	600	17861	600	375868	16	375868	3.91	249661	600	35593	600
lin318	185680	22	185680	6.37	162389	210	-142068	600	333455	21	333455	4.92	310881	161	-306403	600
u574	521313	60	521313	15	533296	566	532881	600	816095	53	816095	13	829432	600	828576	600
u724	655029	93	655029	21	507367	600	473236	600	1023238	86	1023238	17	861585	600	828037	600
dsj1000	722132	205	722132	39	757347	600	757041	600	1238041	178	1238041	46	1284257	600	1283538	600
rl1304	1273882	347	1273882	73	-113919	600	-177210	600	1957232	303	1957232	60	822586	600	776959	600

Table 5.6: Results for the instances with 10 items per city and bounded strongly correlated item weights/profits ($\mathcal{F} = 10$, $\mathcal{T}$=bsc)

	Medium knapsack capacity ($\mathcal{C}$=06)								High knapsack capacity ($\mathcal{C}$=10)							
	EA		RLS		JNB		J2B		EA		RLS		JNB		J2B	
	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time
eil51	78874	1.76	78872	1.04	94337	2.16	94200	79	75884	2.18	75884	1.3	100455	0.9	102066	0.33
berlin52	113112	1.69	113096	1.04	113818	1.38	113818	68	117286	1.75	117286	1.03	118167	0.79	137949	45
eil76	100554	3.22	100556	1.69	138159	4.78	138159	362	102447	2.52	102447	1.6	159321	3.08	159321	224
kroA100	169894	4.76	169894	1.72	186547	6.51	204392	393	167127	4.63	167127	1.6	193124	2.13	207846	265
pr124	230814	4.77	230814	1.77	240058	12	258776	10	253586	5.89	253586	1.56	251613	4.11	273036	10
ch150	248409	7.04	248406	2.36	256254	58	233334	600	211512	6.99	211512	2.32	224417	48	201398	600
u159	287344	8.03	287346	2.85	287881	28	240528	600	316843	6.2	316843	2.32	317585	21	283721	600
kroA200	382166	13	382166	3.9	358811	102	360889	66	416638	9.88	416638	3.03	389017	75	392880	64
ts225	446400	13	446400	3.43	438750	62	235789	600	485578	14	485578	3.52	495835	40	336704	600
a280	485872	19	485872	5.22	388762	600	91306	600	581075	18	581075	4.95	570327	600	91587	600
lin318	388663	24	388664	5.25	319823	92	160915	600	364584	20	364584	6.02	284813	49	-4887	600
u574	1038239	63	1038239	16	1052371	600	1051570	600	1249081	60	1249081	13	1275014	600	1271965	600
u724	1397129	132	1397129	28	1127709	599	1096003	600	1535001	102	1535001	27	1360216	600	1323516	600
dsj1000	1697525	172	1697527	41	1695939	600	1695168	600	1877107	197	1877107	49	1954051	600	1951910	600
rl1304	2804854	315	2804854	74	1713492	600	1636605	599	3076322	400	3076322	83	2352288	600	2263461	600

The reported results show that JNB and J2B were able to surpass RLS and EA for many instances. They perform particularly well on small instances. However, both JNB and J2B are far more time consuming than RLS and EA. Notably J2B is more time consuming than JNB. This can be simply explained by noticing that J2B's neighborhood has a higher complexity than JNB. The high time complexity combined with the limited explorative abilities of JNB and J2B explain the substantial decrease in their performance for mid-size and large instances. When the instance size increases, two scenarios can occur:

— The algorithm gets stuck in a local optimum.
— The algorithm uses a high amount of time to browse the generated neighbourhoods. Thus, it is stopped (e.g. most instances with a runtime of 600 seconds).

Note that JNB has a better objective/runtime ratio for larger TTP instances, while J2B has better a solution quality for small TTP instances.

We believe that the proposed approach is important for the following reasons:

1. Neighborhood-based algorithms are the essence of other more efficient algorithms such as Tabu Search [33, 34], Simulated Annealing [47], etc. More importantly, they allow to generate a homogeneous neighbourhood specific to the problem, which makes it possible to conduct fitness landscape studies of the problem (refer to Chapter 4).
2. Our algorithms are specially designed to exploit particular areas in the search space efficiently.
3. The idea of embedding neighbourhood to design local search algorithms is not limited to the TTP, but could be extended to other multi-component problems.

5.2.4 Complexity and improvement directions

Both JNB and J2B have a high time complexity compared to RLS and EA and other state-of-the-art heuristics. This is mainly due to the Cartesian product between the TSP and KP neighborhoods. Therefore, the time complexity of the resulting neighborhood is the product of the two neighborhoods.

A possible solution to reduce the complexity of the 2opt-BF neighborhood is to use the Delaunay triangulation. Indeed, by using this triangulation, each city has 6 surrounding cities on average, which reduces significantly the time complexity. Moreover, for a given city, one can consider only the direct neighbors, the neighbors of neighbors, or even the entire Delaunay set [76].

The objective function (see equation 3.2) also has a high time complexity. However, its complexity can be easily reduced from $O(m \times n)$ to $O(k \times n)$, where k is the number of items per city. This can be done by arranging the items per cities in a key-value data structure. In this case, the key would be the city reference, and the value would be the list of items in that city.

Furthermore, another technique that can be used in order to obtain a better complexity is to combine the neighborhoods sequentially. Thereby, the resulting time complexity would be the sum of the complexities of the two neighborhoods [60, 28, 25].

These approaches are implemented and further investigated in the next chapter.

5.3 Multi-objective optimisation

Herein, we investigate the TTP as a bi-objective problem by considering traveling time and profit as the overall objectives. Our investigations of this bi-objective model show that the best known TTP solutions can be found in the Pareto set region produced by our EMOA. It is even able to compete with three of the best algorithms for the TTP and find better solutions for the single objective model implicitly. For decision makers in the real-world who encounter multi-component problems, this can mean that comparable or even better solutions can be achieved if a multi-objective approach treats the different components as equally important.

5.3.1 Proposed approach

Our proposed algorithm is built around the NSGA-II [18] framework as implemented in jMetal [23]. Instead of specifying the stopping criteria in terms of function evaluations or generations, we use the 10 minute stopping criteria commonly used in the TTP literature. We define two disruptive operators and two local search heuristics as NSGA-II mutations. In the absence of an effective crossover operator, we use the *Null Crossover* to simply clone selected solutions. At the end of each generation the solutions are sorted based on dominance (non-dominated sorting operator). Solutions in each front are further sorted based on their crowding distance. Based on these two operators, the solutions for the next generation are selected. We will refer to this algorithm as *EMOA-TTP*.

5.3.2 Experiments & results

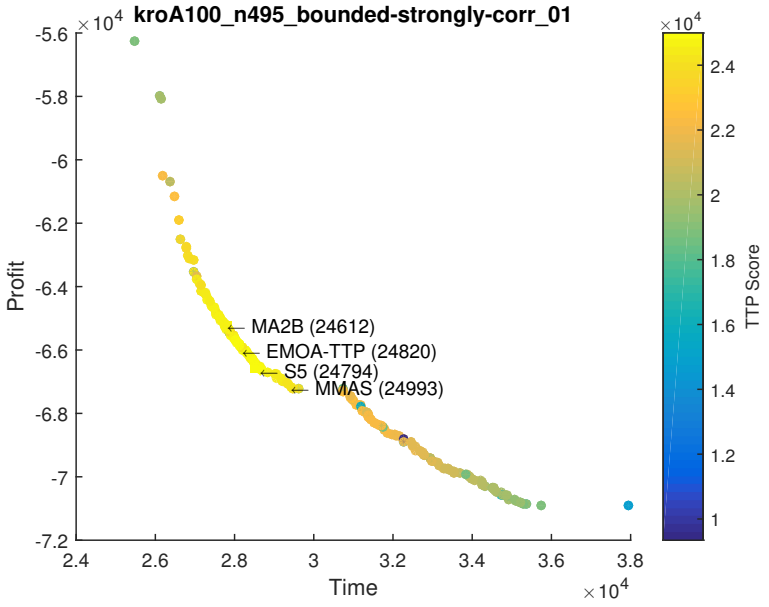


Figure 5.5: The obtained Pareto front for one TTP instance. The colors represent the TTP score.

As a first representative example, we show in Figure 5.5 the obtained Pareto front for one instance. The figure shows that *EMOA-TTP* was able to obtain a set of solutions that represent a trade-off between time and profit. We can also see that the best solutions regarding the TTP score are concentrated in the knee region of the Pareto front. In addition, the solutions obtained using *MMAS* [87], *MA2B* [26], and *S5* [32] are always close to the knee region. This shows that the single objective model is contained in the bi-objective model we investigated, as the TTP score is a simple scalarization of a bi-objective problem by nature.

While the original single-objective TTP formulation allows for a straightforward comparison of objective scores, interpretations of the bi-objective results are more complex. We use the Empirical Attainment Function (EAF) [55, 15] to provide a graphical interpretation of the quality of the outcomes. In Figure 5.6 we show the variance of solution distribution depending on the number of runs of *EMOA-TTP*. The algorithm is run 50

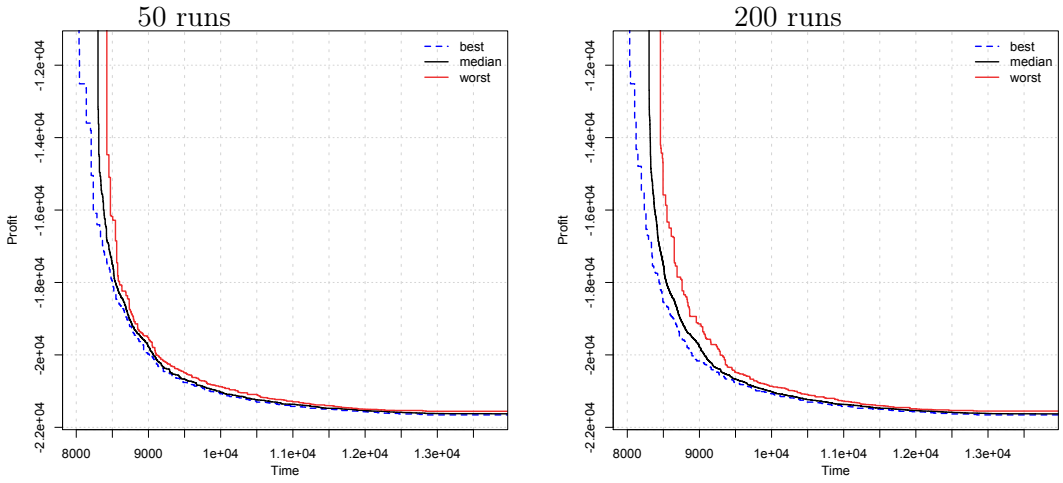


Figure 5.6: Attainment surface plots for 50 and 200 (from left) independent runs of the EMOA-TTP for the instance *berlin52_ n255_ uncorr-similar-weights_ 01*.

and 200 times (from left to right) with randomly chosen seeds. Within each plot, the best results *EMOA-TTP* are shown by the best attainment surface (dashed blue line) and the worst results are shown by the worst attainment surface (solid red line). The median attainment surface corresponds to the region attained by 50% of the runs, which allows us to examine the median case quality of the attained objective vectors.

As we can see, the locations of the attainment surfaces are changing with increasing numbers of independent runs; in particular, the area between the worst and best surfaces increases, as expected for a stochastic algorithm. This means that the distribution of solutions varies and that multiple runs of *EMOA-TTP* have to be performed to compare a statistically significant differences between them at a later stage.

5.4 Conclusion

The study of initialisation strategies showed that the choice of the initial tour has a significant influence on the search process. Whereas the initial picking plan have very slight influence on the solution quality and the runtime.

On the other hand, our proposed local search algorithms JNB and J2B showed a competitive performance compared to RLS and EA. However, both algorithms have a high time complexity compared to other state-of-the-art heuristics. This is mainly due to the Cartesian product between the TSP and KP neighborhoods.

Finally, we investigate the TTP as a bi-objective problem by considering traveling time and profit as the objectives. The obtained results show that some best known TTP solutions can be found in the Pareto set region produced by the EMOA.

The works introduced in this chapter were published in the Applied Soft Computing journal [27] and in the proceedings of the Genetic and Evolutionary Computation Conference (GECCO) in 2017 [29].

CHAPTER 6

Metaheuristics for the travelling thief problem

In this chapter, we introduce a set of meta-heuristic algorithms developed to tackle the Travelling Thief Problem. We propose the following classes of algorithms.

— **Iterating between components:** Instead of embedding component neighbourhoods, this class of algorithms use an itertative structure. These approaches simply take advantage of the natural decomposability of the TTP to divide the search process while optimising the overall objective function.

— **Evolutionary algorithms:** In these approaches, we mainly take advantage of the power of using a population of solutions, genetic operators, and fast local search heuristics.

6.1 A two-stage algorithm

In this section, we propose a two-stage algorithm that combines a hill climbing heuristic with the simulated annealing algorithm.

6.1.1 Performance enhancement techniques

Our algorithm uses the following complexity reduction and performance enhancement techniques.

TSP neighborhood reduction

The Delaunay triangulation [19] is used as a candidate generator for the 2-OPT heuristic. This technique is also used in the memetic algorithm from [60]. Using the Delaunay triangulation to generate candidates reduces the time complexity without significantly decreasing the quality of solutions. In a Delaunay graph, each city has only 6 surrounding cities on average. Thus, the time complexity of the 2-OPT heuristic becomes $O(n)$ instead of $O(n^2)$.

In our implementation, we use the worst-case optimal divide-and-conquer Delaunay triangulation algorithm [38] which has a time complexity of $O(n * log(n))$ [1].

Fast solution evaluation

Instead of using the expensive objective function to evaluate neighbors, the objective value can be recovered more quickly by keeping track of time and weight information at each city of a given tour. The following history vectors are used to perform this operation.

— **Time accumulator (t^{acc}):** a vector that contains the current time at each city of the tour.
— **Weight accumulator (w^{acc}):** a vector that contains the current weight at each city of the tour.
— **Time register (t^{reg}):** a vector that contains the added time at each city of the tour.
— **Weight register (w^{reg}):** a vector that contains the added weight at each city of the tour.

Note that these vectors need to be updated at each iteration of the search heuristic. A similar technique named *incremental evaluation* was proposed in [60]. In our implementation, we go further and use these techniques to recover the vectors themselves. This is quite useful, especially when the problem has a large dimension and the algorithm requires many iterations (e.g. the first improvement hill climbing heuristic) or uses many calls of the evaluation function and changes constantly the solution (e.g. the simulated annealing in Algorithm 6).

Very large-scale instances

For very large instances, local search heuristics such as 2-OPT and the bit-flip search take too long and make very little improvement. Therefore, in order to avoid wasting runtime on small improvements, we use the following techniques:

— **2-OPT with early break:** A threshold acceptance is utilized in our 2-OPT heuristic. Therefore, an improved neighbor solution s' of s is accepted only if the improvement is important (surpasses a threshold T): $\mid G(s') - G(s) \mid > T$.
— **insertion without elimination:** The bit-flip search is skipped for very large instances in the *insertion & elimination heuristic*. Therefore, the elimination task is left to the KP algorithm.

6.1.2 Proposed approach

The proposed heuristic is a single solution algorithm that implements aspects from the CoSolver framework [6]. In addition to the techniques presented above, we use the 2-OPT heuristic [14] to tackle the TSP component and the simulated annealing heuristic [47] to tackle the KP component. The algorithm will be referred to as *CS2SA* (which stands for CoSolver-based with 2-OPT and Simulated Annealing).

The algorithm starts with an initial tour generated using the chained Lin-Kernighan heuristic. The resulting tour is then passed to the *insertion & elimination heuristic* which

1. The source codes of the algorithm are found in the website `http://www.cs.rmit.edu.au/~gl/delaunay.html`

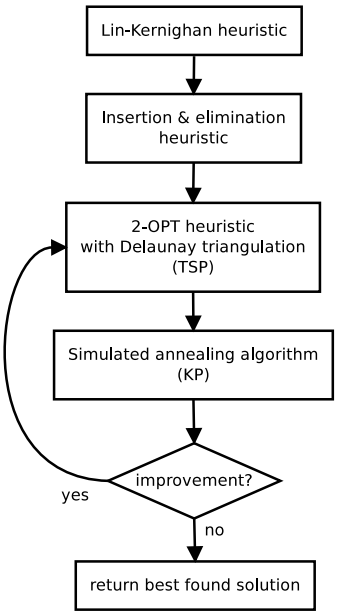


Figure 6.1: CS2SA flowchart

generates a decent initial picking plan. The obtained tour and picking plan are then combined and given to the TSP algorithm which improves the current tour accordingly to the picking plan using the 2-OPT heuristic (see Algorithm 5). Then, the obtained tour is combined with the current picking plan and passed to the KP algorithm which uses the simulated annealing metaheuristic to improve the picking plan (see Algorithm 6). The obtained picking plan is combined with the current tour, and, if no improvements are made, the best-found solution is returned. Otherwise, the tour and picking plan are passed again to the TSP algorithm.

TSP algorithm: The 2-OPT steepest ascent hill climbing heuristic

The 2-OPT heuristic is described in Algorithm 5. The algorithm generates a neighborhood noted $\mathcal{N}$ at each iteration (line 6). The generated neighborhood uses the Delaunay triangulation as explained in the previous sub-section. Then, the objective value recovery technique is used to evaluate neighbors (line 7). The indices of a neighbor and the travel time are stored when a better objective value is found (lines 8-13). At the end of each iteration, if a better solution was found during the neighborhood search, the current solution will be updated and the history vectors will be recovered. The neighborhood search continues until no improvement is made (lines 15-18).

Note that we use the best improvement strategy, which implies that the entire neighborhood is searched at each iteration. Also, the threshold parameter T is set to 0 for small and mid-size instances and -10 for very large instances.

KP algorithm: The simulated annealing

To tackle the KRP sub-problem, we adapt the simulated annealing metaheuristic as shown in Algorithm 6. The algorithm uses a random candidate generator (line 9). Therefore, an internal loop is used to explore more possibilities (line 8). The mutated

Algorithm 5 2-OPT heuristic search for TSKP (best improvement version)

```
1  s ← starting solution
2  f ← starting travel time
3  T ← initialize the threshold parameter
4  repeat
5      improved ← false
6      for s̄ in 𝒩(s) do                          ▷ browse 2-OPT neighborhood
7          f̄ ← evaluate s̄ using objective value recovery technique
8          if f̄ − f < T then                      ▷ acceptance condition
9              i_best = i
10             j_best = j
11             f = f̄
12             improved = true
13         end if
14     end for
15     if improved then
16         s ← apply 2-OPT exchange on s at i_best and j_best
17         recover history vectors
18     end if
19 until not improved
```

solution is evaluated using the objective value recovery technique (line 12). The acceptance test is based on the Boltzmann condition (lines 14-16). If the mutated solution is accepted, the current parameters and solution are updated and the history vectors are recovered (lines 17-21). If a better solution is found, the best-found solution is updated (line 24). At the end of each iteration, the temperature is cooled down (line 25). The algorithm stops when the absolute temperature is reached (line 26).

In our implementation, we use the following parameters:
— $\boldsymbol{T_{abs}}$: the absolute temperature, set to 1.
— $\boldsymbol{T_0}$: the initial temperature.
— $\boldsymbol{\alpha}$: the temperature cooling parameter.
— ***nb_trials***: number of repetitions per iteration. This parameter should be selected more carefully. A high number of trials explores more possibilities but also makes the metaheuristic slow, while a low number of trials explores fewer possibilities but makes the algorithm faster. The value of this parameter is chosen according to the number of items.

More details on tuning these parameters are presented in the next section.

6.1.3 CS2SA variants

We propose two versions of the *CS2SA*. The first is called *CS2SA**, a straight implementation of *CS2SA* for which the parameters are optimised using the advanced tuning strategy described in the next sub-section.

The second variant is called *CS2SA-R*, which uses a random restart instead of directly returning the best found solution when no improvements are made. Instead of using a stopping criterion based on improvement existence, *CS2SA-R* uses the entire available time budget.

Algorithm 6 Simulated annealing for KRP

```
1  s ← starting solution
2  s_best ← s
3  G ← starting gain
4  p ← starting profit
5  f ← starting travel time
6  T ← T_0                                          ▷ initialize the temperature parameter
7  repeat
8      for u in nb_trials do
9          k ← pick an item randomly
10         p̄ ← p + p_k
11         if p̄ > knapsack capacity then skip this iteration end if
12         f̄ ← evaluate time using the objective value recovery technique
13         Ḡ ← p̄ − R ∗ f̄
14         μ ← random number between 0 and 1
15         energy_gap ← Ḡ − G
16         if energy_gap > 0 or exp(energy_gap/T) > μ then      ▷ Boltzmann condition
17             G ← Ḡ
18             p ← p̄
19             f ← f̄
20             s ← apply bit flip at k
21             recover history vectors
22         end if
23     end for
24     if improvement made then s_best ← s end if
25     T ← α ∗ T                                    ▷ cool down temperature
26 until T > T_abs                                  ▷ absolute temperature reached
```

6.1.4 Parameters tuning

In this first set of experiments, we investigate the simulating annealing's parameter tuning. Indeed, one of the hardest tasks when implementing the simulating annealing metaheuristic is to find the best values for its parameters. Thereby, the best-known way to find a good setting is to investigate the algorithm empirically. The tuning is performed using the *irace* package [54], which is an *R* framework based on the iterated racing procedure, an extension of the Iterated F-race procedure proposed by Birattari et al. [3].

The proposed simulated annealing has three influencing parameters: the number of trials nb_trials, the initial temperature T_0, and the cooling parameter α. According to multiple experiments, we observed that the number of trials nb_trials is highly dependent on the size of the instance. This is also due to the runtime limit set to 600 seconds as a standard among TTP algorithms. Therefore, this parameter must be chosen adaptively to the number of items m.

A first tuning attempt

Table 6.1: The eight groups of instances used for the tuning strategy, *bsc* stands for *bounded-strongly-correlated* KP type and *usw* stands for *uncorrelated-similar-weights* KP type

Group name	Size range	Instances
SG1	1–129	eil51_n50_bsc_01, berlin52_n51_bsc_01, eil76_n75_bsc_01, kroA100_n99_bsc_01, ch130_n129_bsc_01
SG2	130–495	a280_n279_bsc_01 eil76_n375_usw_05, kroA100_n495_usw_05
SG3	496–990	u724_n723_bsc_01, dsj1000_n999_bsc_01, u159_n790_usw_05, eil76_n750_uncorr_10, kroA100_n990_uncorr_10
SG4	991–3037	rl1304_n1303_bsc_01, a280_n1395_usw_05, fl1577_n1576_bsc_01, u159_n1580_uncorr_10, d2103_n2102_bsc_01, a280_n2790_uncorr_10, u574_n2865_usw_05, pcb3038_n3037_bsc_01
SG5	3038–18511	rl1304_n13030_uncorr_10, usa13509_n13508_bsc_01, brd14051_n14050_bsc_01, d15112_n15111_bsc_01, pcb3038_n15185_usw_05, fl1577_n15760_uncorr_10, d18512_n18511_bsc_01
SG6	18512–75555	rl11849_n59240_usw_05, usa13509_n67540_usw_05, brd14051_n70250_usw_05, pla7397_n73960_uncorr_10, d15112_n75555_usw_05
SG7	75556–169045	usa13509_n135080_uncorr_10, brd14051_n140500_uncorr_10, d15112_n151110_uncorr_10, pla33810_n169045_usw_05
SG8	169046–$+\infty$	pla33810_n338090_uncorr_10

In order to investigate the effect of the instance size on the three parameters and propose a first tuning technique, we consider eight instance groups of different sizes (Table 6.1). The idea consists of optimising all the three parameters for each group of instances. The results of this experiment are reported in Table 6.2. According to the optimised values, the parameter *nb_trials* tends to change in an exponentially decreasing trend accordingly to the size of the instances. On the other hand, the values for α and T_0 vary widely in $[0.90; 0.99]$ and $[100; 1000]$ respectively. Even the elite configurations found by the irace package for the same group of instances vary unexpectedly in these ranges for the same group of instances. This leads us to believe that the size of the instance has a small influence on α and T_0.

Additionaly, this approach presents drawbacks such as issues in covering all the

instances efficiently. The preliminary results obtained for this configuration clearly confirm this statement. Furthermore, since nb_trials is highly dependent on the size of the instances, it would be more adequate to represent this parameter in terms of m.

Table 6.2: A first tuning tentative for optimising α, T_0, and nb_trials for each group of instances.

Group	$\boldsymbol{\alpha}$	$\boldsymbol{T_0}$	$\boldsymbol{nb\_trials}$
SG1	0.9523	4052	$m \times 9566$
SG2	0.9148	4999	$m \times 9043$
SG3	0.9734	5791	$m \times 331$
SG4	0.9815	1479	$m \times 78$
SG5	0.984	124	$m \times 4$
SG6	0.9047	7138	$m \times 0.2876$
SG7	0.975	145	$m \times 0.0511$
SG8	0.9627	6272	$m \times 0.0285$

A more advanced tuning strategy

In order to span all the instances more efficiently and avoid having a complex tuning scheme, we adopt the following steps [2]:

1. nb_trials is optimised for each one of the 8 groups of instances by fixing the values of α and T_0 to 0.95 and 300 respectively. The optimised values for the parameter nb_trials are shown in Table 6.3.
2. The outcoming values for nb_trials are then used in a spline interpolation formula in order to cover all the instance sizes more efficiently. The interpolation function, denoted L, is a concatenation of linear interpolants as expressed in Equation 6.1. Thus, the number of trials for a given number of items m is $nb\_trials = m \times L(m)$.

$$L(x) = \begin{cases} -341 \times x + 58213 & \text{if } x \in [1, 130[\\ -36 \times x + 18583 & \text{if } x \in [130, 496[\\ -0.71 \times x + 1050.71 & \text{if } x \in [496, 991[\\ -0.1631656082 \times x + 511.70 & \text{if } x \in [991, 3038[\\ -0.0009693680 \times x + 18.9449398992 & \text{if } x \in [3038, 18512[\\ -0.0000147255 \times x + 1.2725979945 & \text{if } x \in [18512, 75556[\\ -0.0000011841 \times x + 0.2494646401 & \text{if } x \in [75556, 169046[\\ -0.0000001142 \times x + 0.0686002283 & \text{if } x \in [169046, 338090[\\ 0.3 & \text{if } x \in [338090, +\infty[\end{cases} \tag{6.1}$$

2. The reported values are the ones obtained for tuning CS2SA*. The same approach is used CS2SA-R, but different values were obtained.

Note that the interpolation points (knots) correspond to the size ranges in Table 6.1.

3. Finally, since α and T_0 are loosely dependent on m, an additional tuning round is performed to optimise these two parameters. The resulting values are 0.9578 and 98 respectively.

Table 6.3: Results for tuning *nb_trials* for each group of instances. α and T_0 are set to 0.95 and 300 respectively.

Group	***nb_trials***
SG1	$m \times 57872$
SG2	$m \times 13896$
SG3	$m \times 700$
SG4	$m \times 350$
SG5	$m \times 16$
SG6	$m \times 1$
SG7	$m \times 0.16$
SG8	$m \times 0.03$

It is worth noting that although the *nb_trials* parameter changes in an exponential fashion according to the instance size, spline interpolation is shown to be significantly more stable than linear and exponential interpolation approaches.

Additionally, the same tuning procedure is used for *CS2SA-R*. Obviously, the resulting parameter values (step 1) are different from the ones obtained for *CS2SA**. Therefore, the interpolation function is also different from the one obtained for *CS2SA**. The most notable difference is that the number of trials are smaller. This is simply explained by the fact that *CS2SA-R* consists on repeating *CS2SA* as many times as possible, which leads to lower values for *nb_trials*.

6.2 A memetic algorithm

Our second approach is a Memetic Algorithm that uses the 2-OPT and bit-flip local search techniques with an MPX-based crossover operator [26]. The algorithm uses a population of 40 individuals and a selection rate of 75% (i.e. 75% of individuals will be replaced at each generation by new ones). The blocks of the algorithm are explained bellow. In the rest of this document, we will refer to this algorithm as *MA2B*.

— *Restrictive Local search:* Local search is used as an intensification technique. First, a 2-OPT search is applied to improve the tour while the picking plan remains fixed. Then, the obtained tour is fixed and a bit-flip search is applied in order to improve the picking plan. However, local search is the most expensive operation in a memetic algorithm. Therefore, in order to reduce the runtime, we limit the number of iterations. We use 50 iterations at most in the initialization, and 10 during the search process.

— *Initialization strategy:* The initialization has a big impact on the search heuristic, especially the initial tours. In fact, all state-of-the-art TTP use a sophisticated heuristic to initialize the tours (mostly the Chained Lin Kernighan heuristic). In our framework, we use the following process for each solution in the population:

1. Generate a random tour, and apply the Lin Kernighan heuristic for a limited number of iterations (accordingly to the size of the instance).
2. Use a greedy heuristic to generate a picking plan.
3. Apply the Restrictive Local Search presented above (50 iterations at most).

This approach is shown to be efficient as it generates decent and unique solutions very quickly.

— *Selection technique:* The tournament selection is used to generate 2 parent solutions. We use a tournament population of 6 individuals.

— *Crossover technique:* The used crossover operator is based on the Maximal Preservative Crossover [64]. The technique consists of two steps.

1. To generate the offspring tour, the classical MPX operator is applied to the parent tours.
2. To generate the offspring's picking plan, we use the obtained tour to determine the state of each item. Concretely, if the city containing the item is inherited from the first parent, the item's state is also inherited from the first parent. Otherwise, the state is inherited from the second parent.

There are many crossover operators for the order coding such as the Edge Recombination Crossover (ERX) [93], the Partition Crossover (PX/GPX) [91, 92], and the Order Crossover (OX). The ERX-like operators are shown to perform better than operators like MPX and OX. Indeed, ERX-like operators are used for intensification purpose in evolutionary algorithms. However, as explained in [24], it is preferred to use a more disruptive operator such as MPX in memetic algorithms. Our tests also confirm that MPX performs better than ERX in a memetic algorithm framework. A possible explanation is that MPX has also a diversification effect while the local search does the intensification part.

— *Mutations:* Mutations are used when a generated offspring already exists in the population. The mutation technique consists of the following three steps.

1. Apply the double bridge move twice on the tour.
2. Generate a new picking plan using the greedy heuristic.
3. Apply the Restrictive Local Search.

— *Stopping Criteria:* The search stops when one of the following conditions is met:
  — Limited number of generations (10^6).
  — Limited number of generations without improvement (10^4).
  — Runtime limit reached (10 minutes).

6.3 Experimental study

In this section, we start by studying the impact of the two component solvers on the overall objective. Then, we present the approach used to tune the simulated annealing parameters. Afterwards, the proposed algorithms are compared against two of the best-known TTP algorithms, namely *MATLS* and *S5*.

6.3.1 Experimental settings

Our tests are performed on a subset of the TTP library. In our setting, we consider the following TSP base problems: *eil76, kroA100, ch130, u159, a280, u574, u724, dsj1000, rl1304, fl1577, d2103, pcb3038, fnl4461, pla7397, rl11849, usa13509, brd14051, d15112, d18512, pla33810.*

The instances are divided into 3 categories:

— **Category 1** ($\mathcal{F} = 1$, $\mathcal{T} = 3$, $\mathcal{C} = 1$) **:** 1 item per city, item values and weights are bounded and strongly correlated, small knapsack capacity.

— **Category 2** ($\mathcal{F} = 5$, $\mathcal{T} = 2$, $\mathcal{C} = 5$) **:** 5 items per city, KP uncorrelated but items have similar weights, average knapsack capacity.

— **Category 3** ($\mathcal{F} = 10$, $\mathcal{T} = 1$, $\mathcal{C} = 10$) **:** 10 items per city, KP uncorrelated, high knapsack capacity.

6.3.2 A comparative study

The results of the comparison are reported in tables 6.4, 6.5, and 6.6. The mean objective value (*mean*) is considered in order to compare the performance of algorithms in terms of solution quality. The relative standard deviation (*rsd*) is provided to measure the consistency, predictability and quality of the algorithms. The runtimes are mainly provided for guidance. Note that the runtimes (*time*) are measured in seconds, the *rsd* is given as a percentage, and the best mean objective values are highlighted.

Table 6.4: Results for category 1

instance	MATLS			S5			CS2SA*			CS2SA-R			MA2B		
	mean	rsd	time	mean	rsd	time	mean	rsd	time	mean	rsd	time	mean	rsd	time
eil76	3705	1.38	4.8	3742	0	600	3423	0.25	108	3842	2.33	600	3837	3.61	113
kroA100	4659	1.34	67	4278	0	600	4420	1.07	142	4594	2.26	600	4517	2.03	119
ch130	8876	0.83	62	9250	0	600	8982	0.42	161	9239	0.49	600	9254	0.38	243
u159	8403	1.42	3.07	8634	0	600	8459	0	250	8560	0.97	600	8618	0.67	357
a280	17678	0.54	7.17	18406	0.02	600	17726	0.13	249	18044	1.13	600	18068	0.56	598
u574	26121	2.15	193	26933	0.14	600	26173	1.06	113	25383	2.4	600	26442	1.01	600
u724	48967	1.25	44	50316	0.12	600	49713	0.56	128	47759	1	600	48650	0.66	600
dsj1000	143699	0	44	137631	0.12	598	144219	0	51	144219	0	600	144219	0	2.64
rl1304	75800	1.26	63	80066	0.94	600	75699	0.47	205	73557	1.38	600	77611	0.66	600
fl1577	88375	0.41	142	92343	1.25	597	88006	0.59	305	84590	1.64	600	89581	0.69	600
d2103	113005	0.47	184	120642	0.2	600	118845	0.01	601	117960	1.4	600	117912	0.8	600
pcb3038	148265	1.14	330	160006	0.15	598	145338	0.5	64	145955	1.65	600	153242	0.89	600
fnl4461	247553	0.37	479	262237	0.11	596	240240	0.32	175	240726	0.67	600	252090	0.64	600
pla7397	365506	1.11	578	395156	0.56	591	315154	0.01	600	325852	9.95	600	367877	0.63	600
rl11849	661283	0.29	600	707190	0.24	591	657842	0.34	600	643738	0.46	600	678923	0.14	600
usa13509	747905	0.47	600	809607	0.24	582	682268	0.11	600	700243	3.7	600	768842	0.42	600
brd14051	815511	0.37	600	875018	0.29	586	802424	0.2	600	799957	0.53	600	840444	0.15	600
d15112	871069	0.52	592	939790	0.47	585	870301	0.14	600	871284	0.67	600	907310	0.63	600
d18512	996544	0.77	600	1072308	0.21	582	964757	0.25	553	971153	0.62	600	1026602	0.26	600
pla33810	1730207	0.87	600	1870330	0.78	572	1778827	0.24	600	1668597	1.83	600	1761708	0.84	600

Table 6.5: Results for category 2

instance	MATLS			S5			CS2SA*			CS2SA-R			MA2B		
	mean	rsd	time	mean	rsd	time	mean	rsd	time	mean	rsd	time	mean	rsd	time
eil76	22188	0.78	3.2	20097	0	600	18753	0	231	22032	1.2	600	22051	1.13	158
kroA100	42595	1.31	6.27	39440	0	600	39271	0	54	43712	2.42	600	41839	2.11	199
ch130	61061	0.11	75	58708	1.27	600	50695	0	103	60864	0.72	600	60052	1.38	346
u159	58289	1.33	10	57618	0	600	58090	0	123	60377	1.03	600	59841	1.61	304
a280	110132	2.34	31	109921	0.01	600	107696	0	196	114087	0.84	600	108924	0.78	600
u574	254770	0.71	222	251775	0.05	600	248584	0	116	248402	1.05	600	247997	0.76	600
u724	303435	1.17	193	305977	0.35	600	309636	0	63	303025	1.11	600	297772	0.96	600
dsj1000	340807	1.55	383	342189	0.59	599	332883	0	109	339136	2.21	600	339193	1.09	600
rl1304	572766	1.18	290	575102	0.86	600	585600	0	181	578064	1.41	600	567915	1.51	600
fl1577	609288	1.77	442	607247	1.62	598	636425	0	281	584764	3.23	600	604106	1.87	600
d2103	849632	1.28	495	853587	1.18	600	842520	0	301	849770	2.9	600	851151	1.41	600
pcb3038	1168108	0.45	581	1179510	0.2	597	1193738	0	365	1159828	0.9	600	1133682	0.54	600
fnl4461	1617401	0.3	600	1625856	0.18	596	1628414	0	157	1588086	0.64	600	1564933	0.86	600
pla7397	4178551	3.87	600	4371424	0.79	591	3713314	0	383	3798791	9.37	600	4097672	2.51	600
rl11849	4587848	0.41	600	4630753	0.29	587	4710135	0	600	4668813	1.22	600	4389067	0.1	600
usa13509	7767305	2.02	600	7818124	0.72	579	8115168	0	600	7930181	2.54	600	7584866	1.13	600
brd14051	6492947	1.13	600	6552858	0.61	587	6654162	0	600	6667470	1.09	600	6158142	0.78	600
d15112	6828152	2	589	6991440	1.31	578	7606856	0	600	7561526	3.18	600	6854135	1.93	600
d18512	7164421	1.27	581	7257709	0.68	600	7579996	0.01	600	7324264	1.98	600	6682736	0.9	600
pla33810	15532990	1.1	600	15574552	0.73	566	15704051	0.5	600	14421765	4.51	600	14578524	0.85	600

The results for the three categories show that *CS2SA**, *CS2SA-R*, and *S5* surpass *MATLS* for most TTP instances, especially for mid-size and very large instances (i.e. $m > 1000$). Another observation is that *CS2SA** performs better for large instances, while *CS2SA-R* is generally better for small instances.

Additionally, the runtimes for *CS2SA** is decent and predictable given the size for most instances. While in some executions, *MATLS* reaches the 600 seconds limit even for small instances.

6.3.3 Results analysis and discussion

According to the reported results, we believe that the proposed stochastic simulated annealing algorithm is well suited to solve the nonlinear knapsack component. It shows an excellent balance between exploitation and exploration. This is mainly due to the nature of the knapsack problem and the bit-flip operator. Indeed, modifying a picking plan requires changing the state of only one item. Thus, selecting the item randomly works well in contrast to TSP and the 2-OPT operator, which requires two cities to modify the tour – it is highly unlikely that both would be efficiently chosen randomly. Thus, following this naive reasoning, the 2-OPT neighborhood search is better suited for the TSP component. We ran some tests to confirm this by comparing a heuristic selecting two cities randomly against the 2-OPT neighborhood search. The results confirm that the 2-OPT search offers a better quality/runtime ratio than the random selection.

However, the simple 2-OPT heuristic proposed in our implementation still suffers

Table 6.6: Results for category 3

instance	MATLS			S5			CS2SA*			CS2SA-R			MA2B		
	mean	rsd	time	mean	rsd	time	mean	rsd	time	mean	rsd	time	mean	rsd	time
eil76	88136	0.26	18	85664	0	600	87577	0	49	88099	0.36	600	87905	0.36	261
kroA100	155492	0.02	203	155540	0	600	155585	0	77	155878	0.53	600	155626	0.01	371
ch130	203468	2.21	32	201174	0.85	600	197555	0	75	206561	0.15	600	204817	1.46	567
u159	242567	0.43	23	242495	0.31	600	242201	0	103	249911	0.06	600	244831	0.4	538
a280	426259	0.2	34	429000	0	600	421713	0	77	427056	0.14	600	426671	0.33	600
u574	966207	0.3	134	966344	0.05	600	953997	0	67	952984	0.4	600	952496	0.42	600
u724	1188761	0.4	364	1188364	0.14	600	1191819	0	71	1181819	0.36	600	1169180	0.26	600
dsj1000	1472638	1	516	1479618	0.1	600	1468858	0	152	1464474	0.74	600	1465811	0.47	600
rl1304	2178475	0.21	476	2184853	0.21	600	2198943	0	187	2165629	0.68	600	2168937	0.26	600
fl1577	2466403	0.42	600	2470917	0.36	598	2505291	0	150	2382050	1.88	600	2419876	0.43	600
d2103	3392877	0.25	600	3392172	0.15	599	3373781	0	75	3331873	0.97	600	3347253	0.11	600
pcb3038	4564261	0.19	600	4573748	0.09	596	4612956	0	148	4514051	0.32	600	4470211	0.44	600
fnl4461	6534422	0.23	600	6554497	0.1	592	6545335	0	285	6463969	0.22	600	6379655	0.23	600
pla7397	13865880	2.16	600	14239609	1.01	569	13197756	0	241	13808867	3.94	600	13482516	1.44	600
rl11849	18275210	0.17	597	18314650	0.16	582	18504773	0	601	18203774	0.49	600	17830729	0.13	600
usa13509	25878290	0.53	595	25918971	0.4	568	26436928	0	595	26251061	0.67	600	25135835	0.43	600
brd14051	23672310	0.51	600	23826398	0.5	577	23908555	0	600	23797888	0.95	600	22963843	0.55	600
d15112	25942410	1.11	590	26211266	0.87	577	27183354	0	600	27160800	1.32	600	25518177	0.73	600
d18512	27164500	1.07	596	27427135	0.35	569	27823470	0.11	600	27269855	1.06	600	26200274	0.45	600
pla33810	58003980	0.4	600	57967446	0.39	541	58106820	0.01	600	55631820	1.38	600	55657317	0.44	600

from a lack of exploration. Although the KP component has the most significant impact on the overall problem as previously shown, a more sophisticated heuristic for the TSP component would be more prefered. For instance, in *MATLS* the order crossover operator [36] is used to tackle this problem and explore more possibilities. While Wagner [87] proposed to use a ants to generate tours and exploit the fact that ant systems are not very efficient in solving the TSP. Stating with mediocre tours allows to avoid the bias, and the resulting algorithm was very competitive for small instances. To address this issue, multiple other solutions could be considered. The hybridization with an evolutionary algorithm is from our point of view the most obvious. In fact, we believe that our approach shows promising hybridization opportunities only by performing little changes in our heuristics in order to reduce the runtime (e.g. reduce the number of trials in SA). Additionally, other metaheuristics could be considered such as tabu search [33, 34], the 3-OPT heuristic [52], the ant colony optimisation algorithm [21], and the threshold accepting algorithm [22] to name a few.

As explained in chapter 5, the heuristics used for the TTP are extremely sensitive to the initial solution, especially the tour. Thus, using an evolutionary algorithm with a population of N solutions would require N good initial tours generated with some time-consuming heuristic such as the Chained Lin-Kernighan. Moreover, the initial tours should be mutually nonidentical, which implies using a stochastic heuristic and implementing a mechanism to eliminate redundant tours. In *MATLS*, the Lin-Kernighan heuristic and a Spanning Tree algorithm are used to generate the tours for an initial population of 30 solutions. We observed that this process takes a considerable amount of time, especially for very large instances. In fact, the initialization time is so significant

for large instances that it does not leave enough time for evolving the population, leading to a very low computational effort during the evolution. One advantage of using a single solution heuristic over an evolutionary algorithm for the TTP is that only one initial tour is required. Thus, the initialization heuristic is used only once, which saves a significant amount of time that can be better utilized in the search process. This is one of the main features making *CS2SA**, *CS2SA-R*, and *S5* outperform *MATLS*.

The reported results also show that *S5* surpasses the other algorithms for most instances from the first category. A possible explanation could be related to the technique used by *S5* to tackle the KP sub-problem. *S5* uses a greedy routine based on sorting items according to their profit, weight, and remoteness from the last city. Given that this category has a small number of items (1 item per city) and the smallest knapsack capacity, we believe such a greedy approach is beneficial in solving this particular type of KP, even when the weights and values of the items are highly correlated. Additional experiments were performed on instances with a low correlation rate and the obtained results show that *S5* performed similarly well for these instances. This aspect has also been explored empirically in [26], and it has been suggested that the most influencing KP parameter is the knapsack capacity C.

The memetic algorithms *MA2B* and *MATLS* have a competitive performance on some small and mid-size instances. This behaviour can be explained by the fact that memetic algorithms are difficult to adapt for large TTP instances, which clarifies their mediocre performance compared to *CS2SA* and *S5*.

For the instances with a higher knapsack capacity (categories 2 and 3), *S5* is still competitive among the other heuristics. However, *CS2SA** has a better performance for most instances. Tables 6.5 and 6.6 show that *CS2SA** clearly outperforms the other heuristics for the majority of instances, while *CS2SA-R* is competitive for some small-size instances, which is not surprising as it was designed to exploit the entire time budget using a different tuning. As these instances have a high knapsack capacity, we believe that the superior performance of *CS2SA** is due to its stochastic behaviour based on selecting items randomly, while using the Boltzmann criterion to manage the balance between exploration (early stage of the search) and exploitation of particular search areas. This process works particularly well for high knapsack capacities as this allows us to explore more possibilities for combining items.

The strength of *S5* relies on the use of a greedy routine based on efficient item sorting. On the other hand, the potential of our two heuristics and *MATLS* is mainly due to making strong assumptions about the problem domain, mainly concretized using fast solution evaluation.

6.4 Conclusion

In this chapter, we introduced two algorithms, namely *CS2SA* and *MA2B*, for the TTP. The two heuristics are compared with state-of-the-art algorithms *MATLS* and *S5*. The former is a single-solution heuristic and the latter is a memetic algorithm. Both algorithms showed competitive performance on different TTP instances.

The works introduced in this chapter were published in the Information Sciences journal [28] and in the proceedings of the Genetic and Evolutionary Computation Conference (GECCO) in 2016 [26].

CHAPTER 7

Hyperheuristics for the travelling thief problem

Nowadays, a large number of metaheuristics have been developed for efficiently solving optimisation problems [35, 65]. However, metaheuristics still present some challenging issues regarding their implementation and maintenance. Due to the fact that simple heuristics are easier to implement and maintain, they have been employed instead of more sophisticated metaheuristics based systems [74]. These heuristic methods can generate high quality solutions in a reasonable time budget. However, there are still some challenges such as the difficulty of adapting sophisticated metaheuristics or efficiently combining simple heuristics to solve a complex problem.

Hyperheuristics are high-level search techniques developed to automate through other (meta)heuristics or machine learning techniques the heuristic design process [78, 7]. In other words, hyperheuristics add another level of abstration on top of the search algorithm in order to automate the process of combining and/or generating parts of the algorithm. In this chapter, we use this mechanism in order to find "good" combinations of basic heuristics, called Low-Level Heuristics (LLHs), and operators, either by using a learning process based on genetic programming (offline apprach), or by continuous learning (online approach).

We propose two hyperheuristic approaches that showed a competitve performance for small TTP instances. The first is an online hyperheuristic based on an Estimation of Distribution Algorithm (EDA). While the second is an offline hyperheuristic based on Genetic Programming (GP).

7.1 Heuristic selection using EDA

7.1.1 Estimation of distribution algorithms

EDA is considered a meta-heuristic approach that produces an offspring population by sampling a distribution, normally estimated by a Probabilistic Graphical Model (PGM). Bayesian networks are considered the most prominent PGMs and are often used for modeling multinomial data with discrete variables [51]. These models are also

capable of capturing multivariate interactions between variables.

Each variable is a node associated with a conditional probability distribution of its values given different value settings of its parents. Mathematically, a Bayesian network with directed edges encodes a joint probability distribution which can be expressed using Equation 7.1:

$$p(\mathbf{X}) = \prod_{i=0}^{n-1} p(X_i|Pa_{X_i}) \tag{7.1}$$

where $\mathbf{X} = (X_0, ..., X_{n-1})$ is the vector of variables, Pa_{X_i} is the set of parents of X_i in the network (i.e., the set of nodes from which there exists an edge to X_i) and $p(X_i|Pa_{X_i})$ is the conditional probability of X_i given Pa_{X_i}. This distribution can be used to generate new candidate solutions using the marginal and conditional probabilities in a modeled data set.

In order to estimate the network structure, several algorithms can be used such as simple greedy algorithms, hill-climbing heuristics, and evolutionary algorithms. In our EDA implementation, we adopt the *K2* algorithm, a greedy local search technique that applies the *K2* metric [12]. *K2* starts by assuming that a given node does not have parents. Then, at each step, it gradually adds the edges that increase the scoring metric the most until no further improvement is possible.

EDAs have been applied as a high-level search technique within a hyperheuristic context in order to solve a number of optimisation problems [74]. In [81] the author compares different hyperheuristics methodologies including a Bayesian heuristic approach which determines the probability distribution of each heuristic based on its historical performance. This method was applied to a variety of discrete problems and showed promising results. Uludağ et al. [84] proposed *HH-PBIL2*, a framework joining EDAs and hyperheuristics for solving discrete dynamic environment problems. This approach uses multi-population combining off-line and online learning to deal with random and cyclic dynamic environments. The results show a good performance over different types of dynamic environments.

7.1.2 The proposed approach

The goal of our proposed approach is to consider EDA as a heuristic selection technique, aiming to evolve combinations of LLHs while looking for good problem solutions.

Our proposal is based on LLHs selection methodology as presented in [74]. Furthermore, in our implementation we explore the interactions (between the used LLHs) encoded into the bayesian network structure used as a PGM. Obtaining the probability distributions of related LLHs supports the generation of new solutions with correlated characteristics, besides providing a representative model for variables interactions. In addition, we apply a surrogate assisted model based on RBFNs to promptly evaluate the solutions sampled by the PGM.

In this section, we detail the HSEDA framework emphasizing its main characteristics.

Encoding scheme

Every solution is represented by an integer vector with M elements, $\mathbf{x} = (x_1, ..., x_M)$, denoting the decision variables, with element $x_m \in \{1, ..., L\}$, $m = 1...M$, L is the total

number of LLHs and x_m indicates the particular LLH to be used at the m-th position in the sequence which the combination identifies.

Within the HSEDA context, a solution is equivalent to a combination of heuristics and a decision variable is equivalent to an LLH [74].

The LLHs are either a component heuristic or a disruptive operation. In the following, we present the list of candidate LLHs.

— KP_{BF}**:** A neighborhood search heuristic targeting the KP part. This heuristic uses a simple bit-flip local search empowered with speedup techniques. It is part of the memetic algorithm MATLS proposed in [60].
— KP_{SA}**:** A simulated annealing adapted to the KP sub-problem, which is used in CS2SA presented in [26].
— TSP_{2-opt}**:** A 2-opt based local search heuristic used for the TSP component. It is usually used in multiple TTP algorithms [59, 26].
— TSP^*_{swap}**:** A disruptive move for the TSP sub-problem that randomly swaps two cities.
— TSP^*_{4-opt}**:** A double bridge move for the TSP sub-problem that randomly selects the cities.
— $KP^*_{BF\psi}$**:** A disruptive routine that toggles the state of ψ% of the picking plan items, where $\psi \in \{20, 30, 40\}$.

The first three operators are local search heuristics obviously used for intensification purposes. While the remaining are disruptive operators designed to explore more efficiently the TTP search space.

The fitness of a heuristic combination depends on its performance on the given instance. The low-level heuristics resulting from a specific combination are sequentially applied to the problem instance, starting from an initial TTP solution.

The framework

The proposed approach searches the space of possible combinations of LLHs. The main steps performed by HSEDA are described in Algorithm 7.

The *Initialization* phase loads the problem instance and randomly generates an initial population Pop^1 of N solutions. Each solution $\mathbf{x}$ is a sequence of LLHs of size M. By a sequence we mean an ordered set of LLHs in which repetition is possible.

The *EvaluateFitness* phase, Step 6 in Algorithm 7, calculates the fitness based on the TTP objective function, i.e. the fitness measures the quality of the sequence while attempting to improve the initial TTP solution.

In the *Selection* phase, a total of N_{PGM} solutions are selected from the population Pop^g, based on their fitness, to compose the Pop^g_{PGM}, where g is the current generation.

Afterward, HSEDA starts, at Step 9, the PGM construction phase in *Probabilistic-ModelEstimation*, according to Pop^g_{PGM} population. Aiming to provide a probabilistic model, a bayesian network is modeled using *K2*-metric [12], and its structure and parameters (joint probability distributions) are estimated.

The PGM is used to sample the set of new solutions (Pop_{smp}) (Step 10). New solutions (N_{smp}), are generated from the joint distribution encoded by the network using the probabilistic logic sampling, and evaluated using a surrogate assisted model based on RBFNs. The RBFN aims to save computational resource that will be further

Algorithm 7 A simplified pseudo-code presenting the main components of HSEDA

Require: *Instance*: problem instance
1 N: population size
2 M: solution size
3 N_{PGM}: number of solutions selected to support the probabilistic
4 model estimation
5 N_{smp}, number of solutions sampled from the probabilistic model
6 sr: survival rate
7 N_{sur}, number of solutions selected from the surrogate model
8 $Max_{runtime}$: maximum runtime
9 Max_{ger}: maximum number of generations
Ensure: C_{best}: the best found solution
10 *{Initialization}*
11 $I \leftarrow LoadInstance(Instance)$
12 $Pop^1 \leftarrow RandomGenerate(N, I, M)$ ▷ initial population
13 $g \leftarrow 1$
14 *{Main loop}*
15 **while** $g \leq Max_{ger}$ and $Max_{runtime}$ is not exceeded do **do**
16 **for** each solution $\mathbf{x} \in Pop^g$ **do**
17 $fitness(\mathbf{x}) \leftarrow EvaluateFitness(\mathbf{x}, I)$
18 **end for**
19 *{Learning the probabilistic model}*
20 $Pop^g_{\text{PGM}} \leftarrow \text{Selection}(Pop^g, N_{PGM})$ ▷ truncation selection
21 $PGM \leftarrow ProbabilisticModelEstimation(Pop^g_{\text{PGM}})$
22 *{Sampling}*
23 $Pop_{\text{smp}} \leftarrow Sampling(PGM, N_{smp})$
24 **for** each n solution $\mathbf{x} \in Pop_{\text{smp}}$ **do**
25 $fitness_n \leftarrow EvaluateFitnessRBFN(\mathbf{x}_n, I)$
26 **end for**
27 *{Survival}*
28 $Pop_{\text{smp}} \leftarrow \text{Selection}(Pop^g, N_{\text{sur}})$
29 **for** each n solution $\mathbf{x} \in Pop_{\text{smp}}$ **do**
30 $fitness_n \leftarrow EvaluateFitness(\mathbf{x}_n, I)$
31 **end for**
32 $Pop^{g+1} \leftarrow Survival(\{Pop^g \cup Pop_{\text{smp}}\}, N, sr)$ ▷ new population
33 $g \leftarrow g + 1$
34 **end while**
35 *{Best found solution}*
36 $C_{\text{best}} \leftarrow SelectBest(Pop^{g-1})$

used to extend the evolution. In our approach a solution vector from Pop_{smp} is an untried point for the RBFN model. The training set is composed by the set of solutions $\mathbf{x}$ for which the fitness is calculated based on the objective functions (instead of being approximated).

The solutions from Pop_{smp} are then sorted based on the surrogate function value and the N_{sur} best are selected to have their fitness recalculated by the real fitness function (Steps 14 to 17). The surrogate model is updated along the HSEDA evolutionary process every time the set of new solutions has its real fitness value calculated to compose the next training set.

This reduced sampled population (N_{sur} best individuals selected from Pop_{smp}), which has its fitness calculated by the TTP objective function, is joint with Pop^g to create the new population for the next generation. However, only N solutions are selected in the *Survival* process to proceed in the evolutionary process as a new population Pop^{g+1}. As shown in Step 18, $N * sr$ fittest solutions from Pop^g, where sr is the survival rate, are selected followed by the remaining fittest ones from Pop_{smp}.

This process is iteratively performed until the termination criterion (Max_{ger} or $Max_{runtime}$, whichever comes first) is satisfied. Finally the best solution C_{best} is selected from the population, according the *SelectBest* function in Step 21.

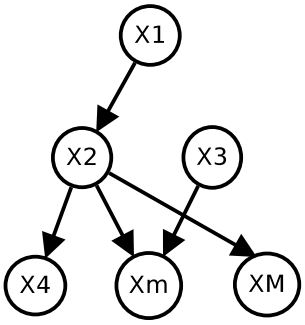


Figure 7.1: Example of a Bayesian Network Structure.

One example of an evolved Bayesian Network (BN) model can be seen in Figure 7.1, with node X_i representing the i-th low-level TTP heuristic to be applied and edges denoting relationships between nodes. The BN structure design considers the *K2* metric to build an edge between a node and its parents. In this case the BN structure indicates that the third heuristic (X_3) is conditionally independent from the first one (X_1). In the sample process, each node value depends on the conditional probability which is learned from the structure and the current set of best solutions. An example of an observation taken from this network is: $\mathbf{x} = [x_1, x_2, x_3, \ldots, x_M] = [KP_{BF}, TSP^*_{swap}, KP^*_{BF20}, KP_{SA}, KP_{BF}, TSP^*_{swap}]$. As the population of heuristic combinations is evolved along the generations, the structure and conditional probabilities are also updated. In this process, different BN structures can be generated at each generation for the given population.

7.2 GP-based hyperheuristic

7.2.1 The proposed approach

In this approach, the goal is to apply GP as a heuristic selection technique, aiming to evolve combinations of heuristics in order to find good problem solutions. Herein, we explain how this strategy can be implemented detailing the proposed algorithm. The motivation behind choosing GP in the context of heuristic selection is mainly due to the fact that GP preserves the correlation between the terminals in sub-trees. The correlations are transferred to the offspring to produce new trees using mainly crossover.

Representation

Every individual is encoded as a tree that represents the program to be executed. The tree's internal nodes are functions, while leaf nodes are terminals, or LLHs.

We use two types of functions:

— **Connectors:** which are used to sequentially execute the child sub-tree from left to right. In our implementation, we use four types of connectors, which we refer to as *con_N*, such as $N \in \{1, \ldots, 4\}$. The first has two child sub-trees, while the second has three.

— **If nodes:** which represent acceptance functions. We use the following three if nodes, each having two child sub-trees:

— **if_improvement:** runs the left sub-tree if the current solution improves the former one, runs the right sub-tree otherwise.
— **if_local_optimum:** runs the left sub-tree if the current solution does not improve the former one; runs the right sub-tree otherwise.
— **if_no_improvement:** runs the left sub-tree if there is no improvement for 20 consecutive iterations; runs the right sub-tree otherwise.

This is rather a simple adaptation compared to other possibilities like using more conditional statements and loops. Nevertheless, these connectors allow representing different combinations of LLHs heuristics.

On the other hand, we use a set of several terminals. The terminals are either a component heuristic or a disruptive operation. In the following, we present a list of the used terminals:

— **term_kpbf:** A neighborhood search heuristic targeting the KP part. This heuristic uses a simple bit-flip hill climbing algorithm empowered with speedup techniques. It is part of the memetic algorithm MATLS proposed in [60].
— **term_kpsa:** A simulated annealing adapted for the KP sub-problem. It is used in CS2SA presented in [26]. The heuristic uses a random neighbor generator using the bit-flip operator. For each fixed picking plan, multiple random neighbors are generated and evaluated.
— **term_tsp2opt:** A 2-opt based local search heuristic used for the TSP component. It is used in many TTP algorithms [61, 59, 26, 25]. This search heuristic generates an entire set of tours based on the 2-opt operator.
— **term_otspswap:** A disruptive move for the TSP sub-problem that randomly swaps two cities.
— **term_otsp4opt:** A double bridge move for the TSP sub-problem that randomly selects the cities. The double bridge is a 4-opt move used as a disruptive operator.
— **term_okpbf$N$:** A disruptive routine that toggles the state of N% of the picking plan items. The item is chosen randomly, and the level of disruption depends on N.

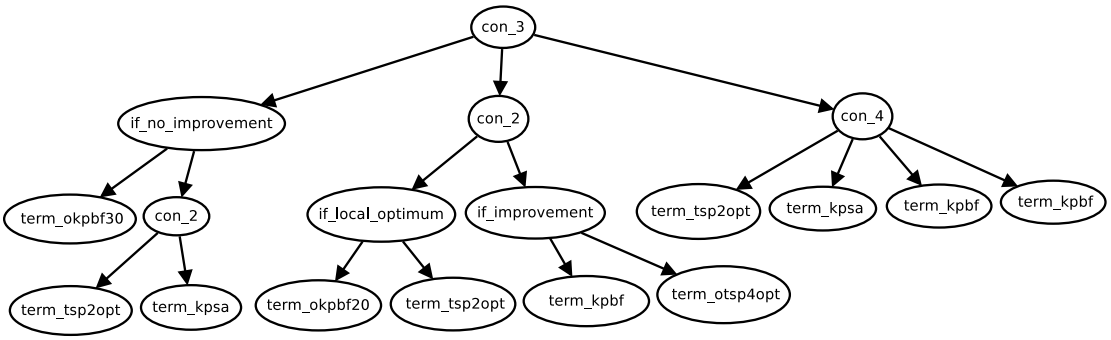


Figure 7.2: Example of a GP individual. The terminals are TTP heuristics, while the internal nodes represent connectors.

An example of a tree individual can be seen in Figure 7.2. Once the leaves of the tree are executed based on an pre-order parse of the entire tree, this example produces the sequence: [*term_ okpbf30 term_ okpbf30 term_ okpbf30 term_ kpbf term_ tsp2opt term_ kpsa term_ tsp2opt*], a GP individual, which will have its fitness assessed.

The fitness of a GP individual M, denoted $fitness_{GP}(M)$, depends on its performance on the given instance. First, the tree is parsed in-order to get the list of leaf

nodes. These are then applied sequentially on the problem instance, starting from an initial TTP solution. The fitness of the individual is set to the achieved TTP objective, according to the Equation 3.2, when the processing of the list of leaf nodes is completed.

The framework

The main steps performed by our approach are described in Algorithm 8. The proposed algorithm makes use of specific strategies presented in the next subsections.

Algorithm 8 GP framework

Require: $Instance$: problem instance
GP_{param}: set of GP parameters
N: population size
$N_{\text{offspring}}$: number of offspring individuals
$Max_{runtime}$: maximum runtime
Max_{ger}: maximum number of generations

Ensure: T: the final individual tree
Initialization
I ← LoadInstance($Instance$)
P ← SetParams(GP_{param})
Pop^1 ← RandomGenerate(N, I, P) ▷ initial population
$g \leftarrow 1$;
Main loop
while $g \leq Max_{ger}$ and $\neg Max_{runtime}$ **do**
  Pop^g.**fitness** ←EvaluateFitness(Pop^g, I, P)
  $Pop_{\text{offspring}}$ ←GeneticOperator($Pop^g, P, N_{\text{offspring}}$); ▷ offspring population
  $Pop_{\text{offspring}}$.**fitness** ←EvaluateFitness($Pop_{\text{offspring}}, I, P$)
  Survival
  Pop^{g+1} ← Survival($\{Pop^g \cup Pop_{\text{offspring}}\}, N$); ▷ new population
  $g \leftarrow g + 1$
end while
Best individual
T_{best} ← SelectBest(Pop^{g-1})

The proposed GP algorithm searches the space of possible heuristic combinations. In the context of our implementation, each GP program is a tree of heuristics. The program is executed in an infix manner to search the space of possible problem solutions in order to find good ones.

The *EvaluateFitness*, in Step 6, calculates $fitness_{GP}$ based on the TTP objective function. In Step 7 *GeneticOperator* applies genetically inspired operations of mutation and crossover in selected parent individuals, in order to produce a new population of programs $Pop_{\text{offspring}}$. The selection of these parent individuals is probabilistically based on fitness. That is, better individuals are more likely to have more child programs than inferior individuals.

The most commonly employed method for selecting these parents in GP is tournament selection [48], which is used in our framework. This method selects a random number of individuals from the population and the best of them is chosen [1].

The genetic operators adopted for the selected parents are standard tree crossover and standard mutation. Given two parents, standard crossover randomly selects a crossover point in each parent tree. Then, it creates the offspring by replacing the sub-tree rooted at the crossover point in a copy of the first parent with a copy of the

1. We apply the Lexicographic Parsimony Pressure technique [56]. If two individuals are equally fit, the tree with less nodes is chosen as the best. This technique has shown to effectively control bloat in different types of problems [80].

sub-tree rooted at the crossover point in the second parent [49]. In standard tree mutation, a randomly created new tree replaces a randomly chosen branch (excluding the root node) of the parent tree [80].

This offspring population $Pop_{\text{offspring}}$ of $N_{\text{offspring}}$ programs is evaluated using the fitness function and merged with the current population Pop^g, where g is the generation number. However, N individuals are selected in *Survival* process to continue in the evolutionary process as a new population Pop^{g+1}, as can be seen in Step 9. The best individual from both Pop^g and $Pop_{\text{offspring}}$ is kept in the new population Pop^{g+1}. The fittest individuals from $Pop_{\text{offspring}}$ followed by the fittest ones from Pop^g compose the remaining $N-1$ individuals from Pop^{g+1}.

This process is iteratively performed until the termination criterion (Max_{ger} or $Max_{runtime}$) has been satisfied, whichever comes first. At the end the best individual T_{best} is selected from the population, according *SelectBest* function in Step 12.

7.3 Experiments

7.3.1 A comparative study of HSEDA

The experiments are performed on a comprehensive subset of the TTP benchmark instances using Matlab. The characteristics of these instances vary widely, and in this work we consider the following diversification parameters:

— All KP correlation types ($\mathcal{T}$) are considered: uncorrelated (unc), uncorrelated with similar weights (usw), and bounded strongly correlated (bsc) types;
— For each TSP and KP combination, the number of items per city (item factor, denoted F) is $\mathcal{F} \in \{01, 05, 10\}$;
— For each TTP configuration, we use 3 different knapsack capacities $\mathcal{C} \in \{01, 05, 10\}$.

To evaluate the proposed hyperheuristic, we use five representative TTP instance groups: *eil51*, *kroA100*, *a280*, *pcb439* and *rat783*. Therefore, using this setting, a total of 135 instances are considered. While significantly larger TTP instances exist, our subset still spans 59% of the instances when measured in the number of cities.

The proposed framework has multiple parameters that need tuning. These parameters are presented in Table 7.1, and are off-line tuned using the automated algorithm configuration package *irace* proposed by López-Ibánez et al. [54].

In the *irace* training phase, we use the following groups of small TTP instances: eil51, berlin52, eil76, and kroA100. These groups are different from the instances considered for our experiments, which were mentioned before and also includes eil51 and kroA100.This way we can test both HSEDA 's ability to create high-quality TTP solutions on the training set, and its ability to generalize to the test set.

In order to provide a statistical analysis of the results, the Friedman test was applied to the obtained objectives values, considering that the results are not normally distributed, based on the Shapiro-Wilk normality test [11]. This test compares a set of instances of a problem and the results reveal all the algorithms have results values with no statistically significant difference can be rejected for all instances. All tests have been executed with a confidence level of 95% ($\alpha = 0.05$) considering the 30 independent runs of each algorithm.

Table 7.2 shows the statistical analysis of pairwise comparisons between HSEDA and

Table 7.1: Parameters of HSEDA algorithm.

Description	Value
Population size N	10
Maximum runtime $Max_{runtime}$	$600s$
Maximum number of generations Max_{ger}	1000
Solution size M	6
Number of solutions selected from surrogate model N_{sur}	$7 * N$
Survival Rate sr	0.75
Number of solutions to the PGM N_{PGM}	N
Number of solutions sampled N_{smp}	1000

the state-of-art algorithms using Dunn-Sidak's post-hoc test with a significance level of $\alpha = 0.05$. When the test result is greater than α, there is no statistical difference between the two techniques. When the test result is less than α, there is statistical difference.

The entries representing a statistically significant difference between HSEDA and the other approach are emphasized (bold). When HSEDA shows better performance (average of its objective values) the background is highlighted.

Table 7.2: Results for pairwise comparisons among HSEDA and state-of-art algorithms using Friedman and Dunn-Sidak's post-hoc tests with $\alpha = 0.05$ for each group of instances.

Instance	HSEDA x MMAS	HSEDA x MA2B	HSEDA x S5
$eil51$	**0.0000**	0.4104	**0.0002**
$kroA100$	0.1519	0.8606	**0.0000**
$a280$	**0.0000**	**0.0017**	0.4260
$pr439$	**0.0002**	1.0000	0.4880
$rat783$	**0.0000**	**0.0001**	1.0000

We can observe from Table 7.2 that there are statistical differences between HSEDA and MMAS for almost all instances set, except for kroA100. HSEDA is better than MMAS for a280, pr429 and rat783 instances and worst for the eil51 group. In comparison with MA2B, HSEDA is better for a280 and rat783 instance sets. On the other hand, the results for S5 show statistical differences for the eil51 and kroA100 instances, where HSEDA is better.

The behaviours of MA2B, MMAS, and S5 are quite expected:

— *MA2B* is a memetic algorithm using expensive local search on a population of 40 individuals and explores longer tours implicitly though a *2-opt* local search heuristic. This makes the algorithm efficient for small instances, but its performance decreases when dealing with larger instances due to the 10 minutes time limit.

— *MMAS* uses artificial ants to explicitly explore longer tours by focusing on good TTP tours. This has been shown to be a very efficient strategy for small instances.

However, its performance decreases significantly on instances having more than 250 cities and 2000 items as shown in [87].

— *S5* has been shown to be very competitive on mid-size and large instances. Its good performance is mainly due to the PackIterative heuristics which is a fast search technique based on a greedy approach. However, S5 is very biased as it does not explore longer tours at all which explains its mediocre performance on small instances.

7.3.2 GP-based approach vs simple GA

In order to show the difference in performance between the proposed approach and a standard GA, we developed a basic GA framework based on one-point crossover. Aiming to make a fair comparison, all the parameters used in the GP-based framework are reconsidered in the GA. Such parameters include the population size, selection operator, stopping criteria, crossover rate, and minimum and maximum solution size. The minimum and maximum depth considered in the GP tree are 2 and 7 respectively. Therefore, the equivalent minimum chromosome size is $2^1 = 2$ and the maximum size correspond to $2^6 = 64$.

Table 7.3: Average (AVG), relative standard deviation (RSD), and the A-test results between genetic programming (GP) and a simple genetic algorithm (GA)

	GP		GA		GP vs GA
	AVG	RSD	AVG	RSD	A-measure
eil51(01,bsc,50)	3732	6.33	**3811**	12.34	0.3
berlin52(01,bsc,51)	3654	7.68	**4056**	2.3	0.04
eil76(01,bsc,75)	3201	7.16	**3329**	7.57	0.34
kroA100(01,bsc,99)	**4174**	4.73	4142	11.66	0.375
a280(01,bsc,279)	16567	4.21	**17301**	3.95	0.22
pr439(01,bsc,438)	**30518**	6.17	30414	10.86	0.51
rat783(01,bsc,782)	**36999**	2.59	31359	17.37	0.85

The two approaches are compared using the average (AVG) and relative standard deviation (RSD) as shown in Table 7.3. Additionally, the A-test is also used in order to gain further insight on the performance of the two approaches. The entries with the best objective values are showed in bold, and the entries representing when GP is showing a better performance than GA are highlighted in blue.

The results show that GP is in general unable to surpass the simple GA. However, the GP approach presents statistically better results for mid-sized instances, i.e. $pr439(438, bsc, 01)$ and $rat783(782, bsc, 01)$.

7.4 Conclusion

In this chapter, we presented two hyperheuristics that showed a competitve performance for small TTP instances. The first is an online hyperheuristic based on an Estimation of Distribution Algorithm (EDA). While the second is an offline hyperheuristic based on Genetic Programming (GP).

The works presented in this chapter were published in the Genetic Programming and Evolvable Machines journal [30] and in the proceedings of the Genetic and Evolutionary Computation Conference (GECCO) in 2017 [58].

Part III

Supply chain optimisation and time dependency

CHAPTER 8

Modelling a supply chain as a Multi-component problem with dependencies

Multi-level supply chain optimisation is one of the most important sources of multi-component problems with dependencies. Companies running supply chain operations on multiple locations have to handle multiple optimisation problems such as production scheduling, vehicle loading, inventory optimisation and distribution. Each of these problems is usually an NP-complete problem. More importantly, dependencies exist between these problems.

In this chapter, we introduce an abstract model illustrating some aspects of supply chain optimisation such as job shop scheduling, material flow between plants and multi-echelon inventory management. We propose to optimise the process of supply chain as an overall problem while thoroughly modelling the scheduling and inventory aspects. The proposed model combines multiple Job Shop Scheduling Problems (JSSP) [37] where every JSSP is a component, and illustrates the dependency between these components using material flow among plants and limited storage capacity. In the rest of this document, we will refer to this new model as the Multi-component Job Shop Scheduling Problem (MJSSP).

8.1 Introduction and background

The proposed problem is mainly represented as a series of job shop scheduling problems, i.e. each component of the supply chain is a variant of the JSSP. Each plant has storage facility with a limited capacity. The availability of items at each plant, except the first, depends on the schedule and material in the previous plants. This in fact links the plants to each other and makes the concept of interdependency more visible in the instances of MJSSP.

The idea of optimising distributed manufacturing systems is not a recent one and has been widely investigated by the operations research community. Wang and Sarker [88] studied a multi-stage supply chain adopting the kanban technique and the just-

in-time strategy to control the material flow. The paper investigates multiple aspect of supply chain management such as inventory optimisation and efficient material flow between the plants. A plant is considered as a general production unit which may be a single machine, a workstation, a city, etc. As a result of using the kanban system, the production is always triggered by the demand at the last plant. The transportation of material from a plant to the next one is triggered by the lot size. The objective function consists on minimising the total inventory cost (raw material, in-progress material, and finished goods). A branch and bound algorithm and a greedy heuristic are proposed in order to solve the problem.

Jia et al. [43] investigated the case when a given job can be processed in several plants. Therefore, the proposed solution tackles two problems: plant selection, and job scheduling. The authors introduce a genetic algorithms combined with Gantt chart to optimise the model using multiple objective function (makespan, production cost, and tardiness).

The above-mentioned works are examples or multiple studies on supply chain optimisation. Most of the models proposed and studied by the operations research and evolutionary computation communities suffer from multiple issues, making them either disliked by researchers or unfit for real-world applications. In fact, some models are simply too complicated to be considered as benchmark problems. While others are too abstract to reflect the features of real-world problems. For instance, we believe that a major drawback of the model proposed in [88] is that it considers a plant as a general production unit, leading to a high level of abstraction. In fact, while the distributed supply chain is considered as a whole, the model ignores job scheduling at the plant level even though the entire system is influenced by the schedules used at each manufacturing unit.

In addition, most benchmark problems studies by the evolutionary computation community are single components of more complex problems (e.g. bin packing, vehicle routing, job shop scheduling, etc.). Ibrahimov et al. [40, 41] investigated methods to tackle this particular aspect among other challenges in real-world supply chain problems. In part II of the study, the authors focus exclusively on supply chain problems with multiple components [1]. Two case studies are investigated. The first is a two-echelon supply chain that consists of the job shop scheduling problem at the first echelon and the vehicle routing problem at the second one. The second model is a higher level industrial problem originating from a global company which produces agricultural chemicals. The model illustrates two main types of components: (1) A plant is an abstraction of a self contained reactor which processes input material. (2) A storage facility is a buffer which holds material until it is used by another plant connected to it.

After the introduction of the Travelling Thief Problem by Bonyadi et al. [4], three competitions were organized around the problem between 2013 and 2017 which caught the attention of many researchers from the EA community. This also resulted in multiple papers proposing metaheuristic and exact approaches to solve the problem. However, while some researchers believe that the TTP has potential real-world applications [61], others criticised it for not being more realistic. In [8], the authors proposed a relaxed version of the problem where multiple thieves are considered as an attempt to make the TTP more realistic. Nevertheless, the fact the TTP only has two components and is not based on realistic situations from logistics and supply chain is another limitation.

1. A component is referred to as a "silo" in the article

Our goal with introducing this new problem is to propose a more balanced model that is comprehensible, while being close enough to real-world supply chains. We also aim at proposing a model that illustrates another class of internal dependency in multi-component problems: Time dependency. Therefore, we propose a multi-component problem having the following features.

— **Moderate abstraction:** the model is designed such that it can be projected to a wide range of standard distributed supply chain problems.
— **Comprehensibility:** the components extend the job shop scheduling problem, which has been widely studied.
— **Realism:** the model has the potential to be applied to various types of real-world supply chains.
— **Flexibility:** The number of components (plants) can be controlled, making the difficulty due to the composition tunable.

Table 8.1 summarizes some state-of-the-art works on supply chain optimisation and outlines some main features introduced by the proposed models. It is clear that our model differs from the others on multiple levels. We consider a plant to be an entity where multiple complex decisions must be made such as job scheduling and determining the number of units to produce. Where those decisions impact the material flow, and therefore, the rest of the system where the plants are distributed sequentially. While Wang and Sarker [88] presents a macro point of view of the system, they neglect important aspects at the plant level. On the other hand, [43] focus on a parallel architecture where the challenge is to find a job schedule among multiple alternative processing plans. Other aspects of divergence include the objective function, solution format, and plant-to-plant transportation trigger to cite a few. Further, the works in [71, 41, 79] mostly model problems from specific domains such as coal mining, production of agricultural chemicals, and other industrial applications.

8.2 The proposed model

8.2.1 The standard Job Shop Scheduling Problem

In the Job Shop Scheduling Problem (JSSP), let $\{J_1, \ldots, J_n\}$ be a set of jobs to be processed on multiple machines $\{M_1, \ldots, M_m\}$. A job consists of a list of ordered tasks where each task must be processed on a specific machine. We note $T_{x,y}$ the task belonging to the job x and to be run on machine y.

In addition, the following constraints must be satisfied.
— Tasks must be scheduled according to the given order, and no parallelism is allowed between tasks from the same job.
— A machine can not run multiple tasks at the same time.

The standard objective is to schedule the tasks on the machines to minimise the length of the schedule, called the makespan, which represents the time it takes from when the tasks are first started until all the tasks are completed.

The are multiple aspects in this benchmark problem that make it unrealistic for practical situations. However, we will be interested in the fact that it does not reflect the complexity and the distributed nature of manufacturing systems. Indeed, big companies possess multiple plants operating on different locations, and the process of manufacturing a given product can be done on multiple plants. In order to increase the overall

performance of the supply chain, a plant cannot optimise the manufacturing process locally without taking into consideration the state of the storage unit in the next plant. In fact, the jobs to be processed by each plant depend on the material flow between the plants of the supply chain.

8.2.2 MJSSP formulation

A company owns m plants to produce n products (brands), a product $i \in \{1, \ldots, n\}$ is generated from a corresponding material i (the material that is ingredient to generate product) which is processed through a chain of plants. A plant k contains P_k machines and one storage with the capacity C_k units of materials. There is also a storage after the last plant called *final storage* for which we suppose that the capacity is infinite (i.e. $C_{m+1} = \infty$). Products are going to be used to fulfill demands at the final storage.

Everything in the model, including costs, transportation, and movement of products take place within the daily buckets (i.e. 24 hours), that is indexed by d.

There is a list of demands, $\{A_{i,d}\}$ and $d \in \{1, \ldots, q\}$, representing the amount of the product i that need to be available at the final storage at day d, where q is the demand length.

Each unit of material (or a product unit) will be associated to a separate job consisting of multiple tasks. However, no task for a given job may start before the previous task for that job (as in the standard JSSP model). Therefore, the total number of jobs is equal to $m \times (\sum_{d=1}^{q} \sum_{i=1}^{n} A_{i,d})$.

Each product generated by the chain of plants is stored at the final storage. To prepare the product i for the final storage, a material i flows from the first plant, processed in all plants in turn, and the product i comes out from the last plant and is stored at the final storage.

Each material i at the day d is topped up by $min\,(u_{i,d}, C_1 - v_d)$ unit in the first storage, where v_d is the total amount of materials in the first storage at the day d, and $u_{i,d}$ is the top-up of the material i at the day d (the maximum amount of raw material that the plant can purshase).

There are $t_{i,k}$ tasks that need to be completed in a given order on each plant k to process a material i and send it to the next plant. A task $1 \leq j \leq t_{i,k}$ to process material i at the plant k by the machine l takes $p_{k,l,i,j}$ time units to be completed. There is an operation-preparation time, o, that need to be considered between each task which is enforced by the machines. This time is needed for the machines to switch between the tasks they are processing, Each machine can only process one task at a time and the processes are not interruptible.

There is a service expense for each plant (S_k per unit of materials per day) that need to be considered for any material in each plant (except in the first plant). Products are then delivered from the final storage to the customers.

Each plant k works at most h hours a day and it can process materials available in its storage to prepare them for the next plant or for the final storage. Every day, fully processed materials in the plant k (a material i is a fully processed in the plant k if all $t_{i,k}$ tasks for that material i has been completed at plant k) are transported to the storage of the plant $k + 1$ (if $k = m$ then they are used to fulfill demands). The transportation of the items from the plant k to the plant $k + 1$ is done between the end

of the working hours of current day and the start of the working hours of the day after for a plant. The transportation activity does not take place during the working hours of the plants. For the materials for which not all of their tasks were completed in a plant k, we need to process their remaining tasks the day after. Note that, if a task is not going to be finished within the starting time of the day and the end of the plant function of that day (h hours after the starting of the day), it should not be executed and it is left for the day after.

The schedule x contains the amount of material that each day is taken from the storage of each plant and is processed in the machines in that plant and the start and end time of the each task of each material that is processed at each plant. The goal is to find x such that it minimises the objective function $C(.)$ defined in Equation 8.1.

$$C(x) = E(x, A) + \sum_{d=1}^{G(x)} \sum_{k=2}^{m+1} S_k V(x, k, d) \tag{8.1}$$

where $E(x, A)$ is a function of the aggregation of the number of days each demand was delayed by if the schedule x is used, $V(x, k, d)$ is the amount of materials in the plant k at day d if we follow the schedule x, and $G(x)$ is the total number of days the schedule x need to fulfill all demands.

Note that, for any schedules x, D is a lower bound for $G(x)$ that might not be achievable. Also, there are $m + 1$ storages and the materials at the storage of the first plant do not include any service expenses.

The objective function proposed in Equation 8.1 is designed in order to reflects two important aspects in a supply chain:

— **The overall throughput:** the first term of the equation corresponds to the delay. Minimising the delay increase the speed at which the system generates money.

— **Operational expenses:** the second term of the equation corresponds to the total cost spent by all the plants in the system.

8.3 Representation and data structures

In our proposed model, we use a simple natural problem and solution codings. The MJSSP solution is coded using a matrix containing schedules for every plant for every simulated day. A schedule is an object containing a set of tasks and their associated time slots.

However, a more efficient data structure could be considered to represent the problem and the solution. For instance, in the standard JSSP, a Directed Acyclic Disjunctive Graph (DADG) is isually used. Figure 8.1 illustrates a JSSP solution coded using a DADG. The nodes in the graph represent the tasks. The solid lines with arrows, which connect consecutive tasks for the same job, correspond to conjunctive constraints. The arrows point in the order in which the tasks must be processed. The dashed lines, which connect tasks for the same machine, correspond to disjunctive constraints. These lines do not have arrows, because the order of these tasks is not given.

In the context of MJSSP, we need to define a mapping between our current representation (in terms of evolutionary algorithms, it could be seen as the phenotype) and the

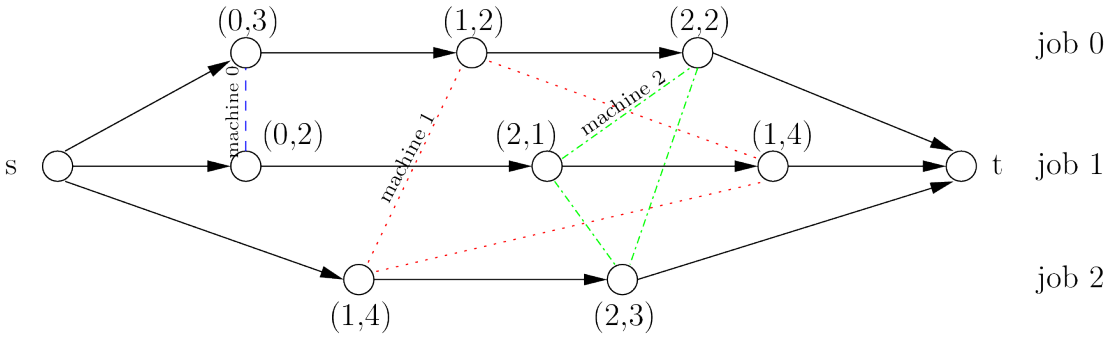


Figure 8.1: A directed acyclic disjunctive graph representating a JSSP instance. Source: `https://acrogenesis.com/or-tools/documentation/user_manual/manual/ls/jobshop_def_data.html`

DADG (the genotype). We believe that using such efficient representation is a key factor for the MJSSP. Using DADG can have multiple advantages, such as the facilitation of handling of scheduling constraints and simplicity of using GA and LS operators.

8.3.1 A representative example

Figure 8.2 represents an instance file of the MJSSP, where:

— $m = 2$ is the number of plants.
— $n = 2$ is the number of products.
— $horizonLength = 20$ is the number of days to prepare the demand and finalise it for delivery.
— $workingHours = 720$ is the number of working hours per day expressed in minutes (12 hours).
— $transportHours = 720$ is the number of hours allowed to transport material between the plants expressed in minutes (12 hours).
— *Product X tasks specifications* represent the number of minutes (first column) a task for a given product X should run on each machine (second column). Note that the tasks are ordered (first row, then second, etc.).
— *Product X demands* is the set of ordered demands for a given product X, where the numbers represent the number of unit that must be produced.
— *Product X topUps* is the maximum number of material units for a product X that the company can purchase from its supplier.

In figure 8.3, we illustrate the manufacturing process for the representative example introduced in Figure 8.2. The figure represents the two plants, their specifications, and the material flow from the supplier to the final storage.

8.4 Problem analysis

In MJSSP, each component of the problem is a local plant job shop scheduling problem. Also, each plant must perform its operations in a limited time window of 24 hours. We can easily recognise the existence of time dependency.

Let us note P_k^d the JSSP sub-problem for the component at plant k during day d, and P_k the JSSP component at plant k. By defining the dependencies between P_k^d,

```
m: 2
n: 2
horizonLength: 10
workingHours: 720
transportHours: 720
Plant 1 specifications:
    Number of machines (P): 3
    Capacity (C): 10
    Charge per day per unit (S): 0
    Product 1 tasks specifications
        50 2
        95 1
        30 3
    Product 2 tasks specifications
        55 3
        20 1
        60 2
Plant 2 specifications:
    Number of machines (P): 2
    Capacity (C): 17
    Charge per day per unit (S): 16
    Product 1 tasks specifications
        95 2
        15 1
    Product 2 tasks specifications
        100 1
        35 2
Product 1 demands
17 4 5
Product 2 demands
20 3 9
Product 1 topUps
7 7 7 7 10 7 7 17 17 17 17 17 17 17 17 7 7 7 7 7 7 7 7 7 7 7 17 10 7 7
Product 2 topUps
5 15 5 15 5 15 5 15 5 15 15 15 15 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 15 5
```

Figure 8.2: An MJSSP instance file

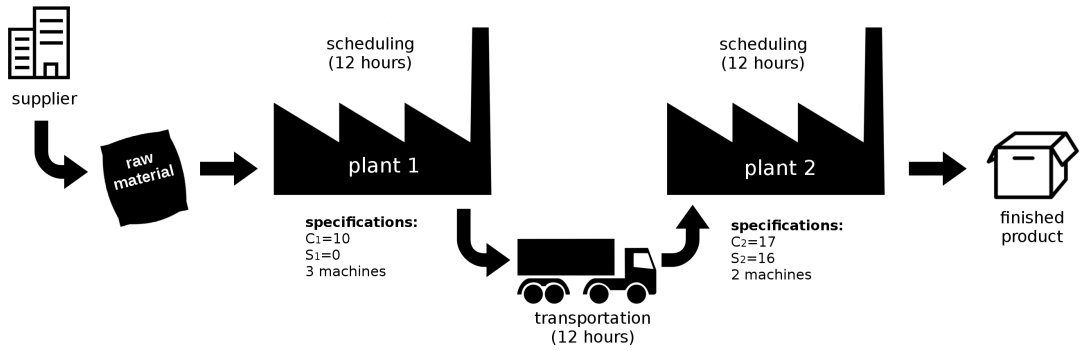


Figure 8.3: Sketch of the manufacturing process following the MJSSP model.

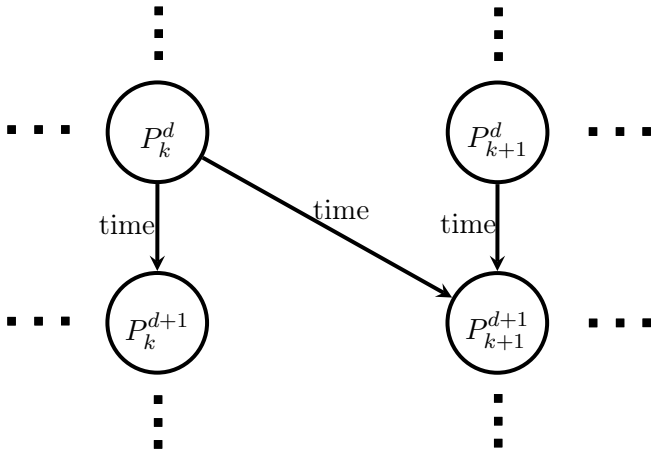


Figure 8.4: The graph of dependency for the Multi-component Job Shop Scheduling Problem

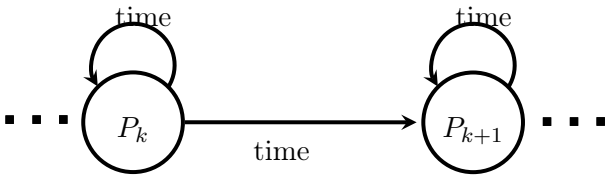


Figure 8.5: The graph of dependency for the Multi-component Job Shop Scheduling Problem (compressed representation)

P_{k+1}^{d}, P_{k}^{d+1} and P_{k+1}^{d+1}, we will be able to define the dependencies in the entire supply chain. Figures 8.4 and 8.5 represent the graph of dependency for the MJSSP and the compressed version of the graph of dependecy respectively.

It is worth noting that, due to the existence of time dependencies, it is impossible to find a solution —regardless of its objective value— for a components in isolation, except for the component associated with the first plant.

8.5 Conclusion

In this chapter, we modelled a supply chain optimisation problem based on the job shop scheduling problem as a multi-component problem with time dependency. We propose to optimise the process of supply chain as an overall problem while thoroughly modelling the scheduling and inventory aspects. The main goal was to propose a model that is abstract, comprehensible and realistic to a certain extent making it possible to use for real-world situations.

Table 8.1: A non-exhaustive list of related works on multi-stage supply chain optimisation and multi-component problems

Reference	**Brief description**	**Optimisation criteria**	**Solution format**	**Components**	**Solving approach**
Wang and Sarker [88]	A multi-stage supply chain using the kanban technique to implement the just-in-time philosophy.	Minimize inventory cost	Number of kanban, number of batches, batch size, production quantity	NA	Branch & Bound, greedy algorithm
Jia et al. [43]	A distributed manufacturing system where a job can be processed at multiple locations.	Minimize makespan, job tardiness, production cost	Schedule affecting a job to a plant/machine	Multiple job-shop scheduling problems	Genetic algorithm
Polyakovskiy et al. [71]	A combination of JIT scheduling and the bin packing problem.	Minimizing earliness and tardiness penalties	Cutting plan	Batch scheduling and bin packing	CP-based heuristic, agent-based heuristic
Ibrahimov et al. [41]	A simplified two-echelon supply chain combining the job shop scheduling problem and vehicle routing.	Minimise the total makespan of all components and the total travel distance by all the trucks	Job schedules and vehicle routes	Job shop scheduling and vehicle routing problem	Cooperative Coevolution
	An industrial problem presenting a network of interdependent plants and storage facilities	Maximisation of product yield, minimisation of product changeovers, adherence to a specified product ratio for the remaining planning period, satisfaction of demand, minimising transportation, inventory, backlogging costs	Overall schedule	NA	Evolutionary algorithm
Schellenberg et al. [79]	A benchmark problem for coal processing and blending.	Maximise the total profit for all the resulting products	Blend plan (fraction of raw materials that each product is composed of and the way each fraction should be processed	Coal blending and coal processing	Genetic algorithm, linear programming
Our model	A distributed manufacturing system processing multiple products sequentially at multiple plants.	Minimize delay and operations cost	Multiple schedules for each plant at a given day	Multiple job-shop scheduling problems	Greedy algorithm in chapter 8

CHAPTER 9

A greedy algorithm for the MJSSP

In this chapter, we propose a simple algorithm that generates feasible solutions. The proposed algorithm is based on the idea of selecting the earliest time slot for each task separately. We include a stochastic aspect by randomly preparing the jobs to schedule (referred to as a batch).

9.1 Proposed algorithm

Algorithm 9 Simplified pseudocode of the random-greedy algorithm for the MJSSP

```
1  Extract the list of jobs for each plant
2  for each simulated day d do
3      Import raw material to first plant until the space limit is reached
4      for each plant k do
5          Generate a random batch B
6          for each job j ∈ B do
7              for each task t ∈ j do
8                  Find the earliest time slot for t if possible
9                  if t is scheduled, tag it as so
10             end for
11         end for
12     end for
13     Transfer material from last plant to the final storage
14     for each plant k, except the last one do
15         Transfer material from k to the next plant
16     end for
17     if all jobs are scheduled then
18         Stop the loop
19     end if
20 end for
21 Output constructed solution
```

The proposed algorithm is detailed in the pseudocode in Algorithm 9. The algorithm

starts by extracting the complete list of jobs to be executed on each plant, where each job corresponds to a unit of material to be produced (Line 1). Then, for each simulated day, import new raw material from the provider to the first plant with respect to the allowed top-ups and plant storage capacity (Line 3). Afterwards, a random batch is generate using unfinished jobs and new ones. The batch consists of randomely ordered jobs to be scheduled 5. For every job task, the algorithm attempts to find the earliest time slot to schedule the task if possible such that the scheduling constraints are satisfied (no parallelism between tasks of the same job, no overalapping between task and respect of task order) (Line 8). Then, if a time slot is found for the task, it is marked as *scheduled* to avoid rescheduling (Line 9). When all the tasks in all the generated batches are covered, the finished products are moved to the final storage (Line 13). Then, for each plant, the algorithm moves the finished material units to the next one to continue the manufacturing process (Line 15). Once all the jobs are scheduled, the process is stopped, and the generated solution is returned (Lines 18 and 18).

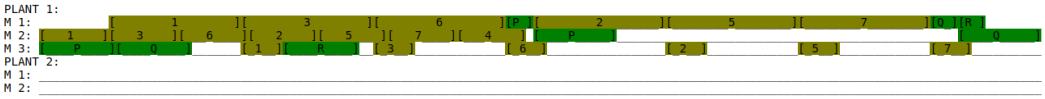
9.2 Simulation and solution

We applied our random-greedy algorithm on the instance presented in Figure 8.2. The resulting solution is a 9 days schedule as shown in Figure 9.1. In the visualisation, each color represents a product, and each symbol represents a job, which is composed of ordered tasks. In addition, the amounts of material at the last storage facility for each product are illustrated in Figure 9.2.

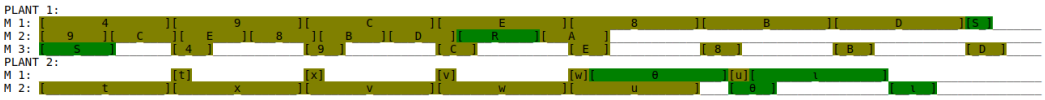
In the generated solution, the first demand was satisfied after 7 days, the second was satisfied after 8 days, and the third was satisfied 9 days. The accumulated delay is equal to 24, and it is calculated using the sum of the individual delays. The total service cost is 1280. Finally, the objective value is equal to 1304.

9.3 Conclusion

This chapter introduces our first attempt of designing a simple algorithm able to generate feasible solutions for the MJSSP. The proposed algorithm selects the earliest time slot for each task separately (greedy strategy). The algorithm is made stochastic by randomly preparing the jobs to schedule.

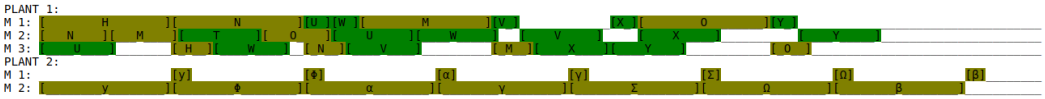


(a) Schedules of day 1

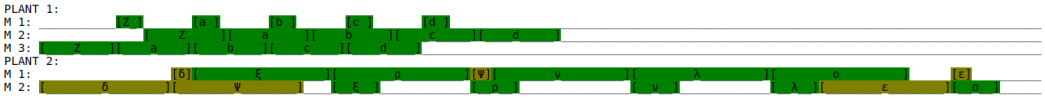


(b) Schedules of day 2

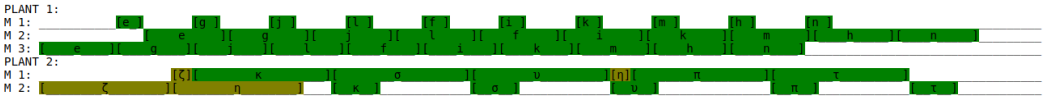
(c) Schedules of day 3



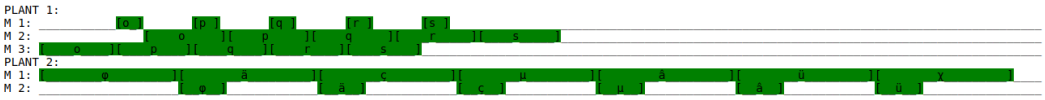
(d) Schedules of day 4



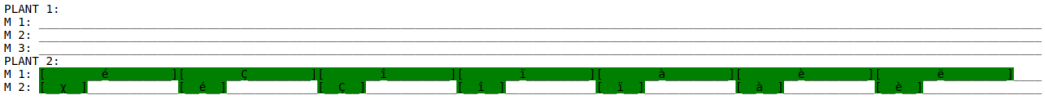
(e) Schedules of day 5



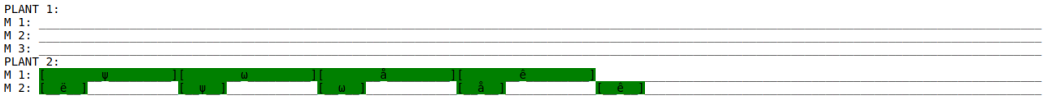
(f) Schedules of day 6



(g) Schedules of day 7



(h) Schedules of day 8



(i) Schedules of day 9

Figure 9.1: Gantt charts of a solution generated using the random-greedy algorithm for the MJSSP instance in 8.2. For each day, the Gantt charts for all 2 plants are illustrated during 9 days. The tasks of the product 1 are shown in yellow while the tasks for the product 2 are shown in green.

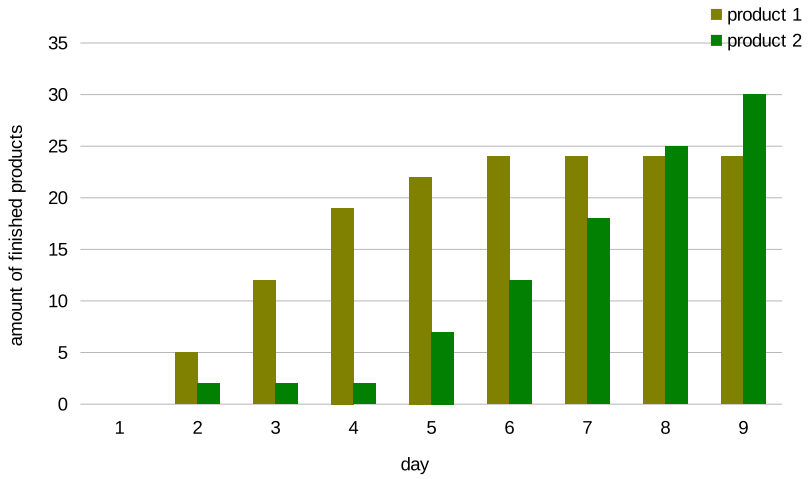


Figure 9.2: Final storage states for every day for the generated solution.

Main conclusions and future perspectives

In this thesis, we studied multiple aspects of problems with multiple interdependent components. In part I, we started by introducing two illustrative examples inspired from real-world industrial problems. Afterward, we provided a formal model for dependencies in multi-component problems which is based on the definitions by Bonyadi et al. [5]. The model covers multiple types of dependencies between components and allowed us to propose a classification of dependencies.

In part II, we thoroughly studied the Travelling Thief Problem (TTP), a benchmark problem that combines the Travelling Salesman Problem (TSP) and the Knapsack Problem (KP). After formally introducing the TTP, we analysed the problem by studying the impact of dependency on the complexity of the objective function and on the search process for each component. We took the analysis one step further by using local optima networkds to perform an in-depth fitness landscape analysis on the problem. Afterwards, we introduced initialisation strategies, local search algorithms, and more sophisticated metaheuristic and hyperheuristic solutions able to tackle TTP instances of various sizes and levels of difficulty.

In part III, we proposed the multi-component Job Shop Scheduling Problem (MJSSP), a model that is inspired from real-world supply chain optimisation. The proposed problem illustrates the manufacturing process in a multi-level supply chain, and embeds a time dependency between its components. We introduced mathematical formulations for the MJSSP and provided details about the problem representation and solution coding and the nature of dependencies in the problem. Moreover, we propose a simple randomised greedy algorithm able to generate feasible solutions based on the idea of selecting the earliest time slot for each job task in the supply chain.

One lesson that could be learned from our study of problems with multiple interdependent components is the importance of inducing domain knowledge in algorithms in order to efficiently tackle these problems. In the future, more efforts will be made to study other important aspects of multi-component probelms in general, and supply chain optimisation in particular. Such aspects include uncertainty, and bottleneck localisation and handling. Furthermore, we engage to study multi-component problems from a more theoretical perspective.

Bibliography

[1] Adamic, L. A., Lukose, R. M., and Huberman, B. A. (2006). Local search in unstructured networks. *Handbook of Graphs and Networks: from the Genome to the Internet*.

[2] Applegate, D., Cook, W., and Rohe, A. (2003). Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92.

[3] Birattari, M., Yuan, Z., Balaprakash, P., and Stützle, T. (2010). F-race and iterated f-race: An overview. In *Experimental methods for the analysis of optimization algorithms*, pages 311–336. Springer.

[4] Bonyadi, M. R., Michalewicz, Z., and Barone, L. (2013). The travelling thief problem: the first step in the transition from theoretical problems to realistic problems. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 1037–1044. IEEE.

[5] Bonyadi, M. R., Michalewicz, Z., Neumann, F., and Wagner, M. (2016). Evolutionary computation for multicomponent problems: opportunities and future directions. *arXiv preprint arXiv:1606.06818*.

[6] Bonyadi, M. R., Michalewicz, Z., Przybylek, M. R., and Wierzbicki, A. (2014). Socially inspired algorithms for the travelling thief problem. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 421–428. ACM.

[7] Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. (2013). Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724.

[8] Chand, S. and Wagner, M. (2016). Fast heuristics for the multiple traveling thieves problem. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*, pages 293–300. ACM.

[9] Chicano, F., Daolio, F., Ochoa, G., Vérel, S., Tomassini, M., and Alba, E. (2012). Local optima networks, landscape autocorrelation and heuristic search performance. *Parallel Problem Solving from Nature-PPSN XII*, pages 337–347.

[10] Clauset, A., Shalizi, C. R., and Newman, M. E. (2009). Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703.

[11] Conover, W. (1999). *Practical Nonparametric Statistics*. Wiley, third edition.

[12] Cooper, G. and Herskovits, E. (1992). A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347.

[13] Cornuéjols, G., Nemhauser, G. L., and Wolsey, L. A. (1983). The uncapacitated facility location problem. Technical report, Carnegie-mellon univ pittsburgh pa management sciences research group.

[14] Croes, G. (1958). A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812.

[15] da Fonseca, V. G., Fonseca, C. M., and Hall, A. O. (2001). Inferential performance assessment of stochastic optimisers and the attainment function. In *Evolutionary Multi-Criterion Optimization, First International Conference, EMO 2001, Zurich, Switzerland, March 7-9, 2001, Proceedings*, pages 213–225.

[16] Dantzig, G. B. and Ramser, J. H. (1959). The truck dispatching problem. *Management science*, 6(1):80–91.

[17] Daolio, F., Verel, S., Ochoa, G., and Tomassini, M. (2013). Local optima networks of the permutation flow-shop problem. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 41–52. Springer.

[18] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

[19] Delaunay, B. (1934). Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(793-800):1–2.

[20] Deng, W., Li, W., Cai, X., and Wang, Q. A. (2011). The exponential degree distribution in complex networks: Non-equilibrium network theory, numerical simulation and empirical data. *Physica A: Statistical Mechanics and its Applications*, 390(8):1481–1485.

[21] Dorigo, M. and Gambardella, L. M. (1997). Ant colonies for the travelling salesman problem. *BioSystems*, 43(2):73–81.

[22] Dueck, G. and Scheuer, T. (1990). Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of computational physics*, 90(1):161–175.

[23] Durillo, J. J. and Nebro, A. J. (2011). jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771.

[24] Dzubera, J. and Whitley, D. (1994). Advanced correlation analysis of operators for the traveling salesman problem. In *Parallel Problem Solving from Nature—PPSN III*, pages 68–77. Springer.

[25] El Yafrani, M. and Ahiod, B. (2015). Cosolver2b: an efficient local search heuristic for the travelling thief problem. In *Computer Systems and Applications (AICCSA), 2015 IEEE/ACS 12th International Conference of*, pages 1–5. IEEE.

[26] El Yafrani, M. and Ahiod, B. (2016). Population-based vs. Single-solution Heuristics for the Travelling Thief Problem. In *Genetic and Evolutionary Computation Conference*, GECCO'16, pages 317–324.

[27] El Yafrani, M. and Ahiod, B. (2017). A local search based approach for solving the travelling thief problem: The pros and cons. *Applied Soft Computing*, 52:795–804.

[28] El Yafrani, M. and Ahiod, B. (2018). Efficiently solving the traveling thief problem using hill climbing and simulated annealing. *Information Sciences*, 432:231 – 244.

[29] El Yafrani, M., Chand, S., Neumann, A., Ahiod, B., and Wagner, M. (2017). Multi-objectiveness in the single-objective traveling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 107–108. ACM.

[30] El Yafrani, M., Martins, M., Wagner, M., Ahiod, B., Delgado, M., and Lüders, R. (2018a). A hyperheuristic approach based on low-level heuristics for the travelling thief problem. *Genetic Programming and Evolvable Machines*, 19(1-2):121–150.

[31] El Yafrani, M., Martins, M. S., Krari, M. E., Wagner, M., Delgado, M. R., Ahiod, B., and Lüders, R. (2018b). A fitness landscape analysis of the travelling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM.

[32] Faulkner, H., Polyakovskiy, S., Schultz, T., and Wagner, M. (2015). Approximate approaches to the traveling thief problem. In *Genetic and Evolutionary Computation Conference*, GECCO'15, pages 385–392. ACM.

[33] Glover, F. (1989). Tabu search - part i. *ORSA J. on Computing*, 1(3):190–206.

[34] Glover, F. (1990). Tabu search - part ii. *ORSA J. on Computing*, 2(1):4–32.

[35] Glover, F. W. and Kochenberger, G. A. (2006). *Handbook of metaheuristics*, volume 57. Springer Science & Business Media.

[36] Goldberg, D. E. and Lingle, R. (1985). Alleles, loci, and the traveling salesman problem. In *Proceedings of the first international conference on genetic algorithms and their applications*, pages 154–159. Lawrence Erlbaum Associates, Publishers.

[37] Graham, R. L. (1966). Bounds for certain multiprocessing anomalies. *Bell Labs Technical Journal*, 45(9):1563–1581.

[38] Guibas, L. and Stolfi, J. (1985). Primitives for the manipulation of general subdivisions and the computation of voronoi. *ACM transactions on graphics (TOG)*, 4(2):74–123.

[39] Hernando, L., Daolio, F., Veerapen, N., and Ochoa, G. (2017). Local optima networks of the permutation flowshop scheduling problem: Makespan vs. total flow time. In *Evolutionary Computation (CEC), 2017 IEEE Congress on*, pages 1964–1971. IEEE.

[40] Ibrahimov, M., Mohais, A., Schellenberg, S., and Michalewicz, Z. (2012a). Evolutionary approaches for supply chain optimisation: part i: single and two-component supply chains. *International Journal of Intelligent Computing and Cybernetics*, 5(4):444–472.

[41] Ibrahimov, M., Mohais, A., Schellenberg, S., and Michalewicz, Z. (2012b). Evolutionary approaches for supply chain optimisation. part ii: multi-silo supply chains. *International Journal of Intelligent Computing and Cybernetics*, 5(4):473–499.

[42] Iori, M. and Martello, S. (2010). Routing problems with loading constraints. *Top*, 18(1):4–27.

[43] Jia, H., Fuh, J. Y., Nee, A. Y., and Zhang, Y. (2007). Integration of genetic algorithm and gantt chart for job shop scheduling in distributed manufacturing systems. *Computers & Industrial Engineering*, 53(2):313–320.

[44] Johnson, D. S. (1973). *Near-optimal bin packing algorithms.* PhD thesis, Massachusetts Institute of Technology.

[45] Kauffman, S. (1993). The structure of rugged fitness landscapes. *The Origins of Order. Self-organization and Selection in Evolution.*

[46] Kauffman, S. and Levin, S. (1987). Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, 128(1):11–45.

[47] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

[48] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA.

[49] Langdon, W. B., Poli, R., McPhee, N. F., and Koza, J. R. (2008). *Genetic Programming: An Introduction and Tutorial, with a Survey of Techniques and Applications*, pages 927–1028. Springer Berlin Heidelberg.

[50] Larranaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., and Dizdarevic, S. (1999). Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13(2):129–170.

[51] Larrañaga, P. and Lozano, J. A. (2002). *Estimation of distribution algorithms: A new tool for evolutionary computation*, volume 2. Springer, Netherlands.

[52] Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal, The*, 44(10):2245–2269.

[53] Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516.

[54] López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., and Birattari, M. (2011). The irace package, iterated race for automatic algorithm configuration. Technical report, Citeseer.

[55] López-Ibáñez, M., Paquete, L., and Stützle, T. (2010). Exploratory analysis of stochastic local search algorithms in biobjective optimization. In Bartz-Beielstein, T., Chiarandini, M., Paquete, L., and Preuss, M., editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 209–222. Springer, Berlin, Germany.

[56] Luke, S. and Panait, L. (2002). Lexicographic Parsimony Pressure. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO'02, pages 829–836, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[57] Martello, S. and Toth, P. (1990). *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc.

[58] Martins, M. S., El Yafrani, M., Delgado, M. R., Wagner, M., Ahiod, B., and Lüders, R. (2017). Hseda: a heuristic selection approach based on estimation of distribution algorithm for the travelling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 361–368. ACM.

[59] Mei, Y., Li, X., Salim, F., and Yao, X. (2015). Heuristic evolution with genetic programming for traveling thief problem. In *Proceedings of the 2015 IEEE Congress on Evolutionary Computation, CEC*, pages 2753–2760. IEEE.

[60] Mei, Y., Li, X., and Yao, X. (2014a). Improving efficiency of heuristics for the large scale traveling thief problem. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pages 631–643. Springer.

[61] Mei, Y., Li, X., and Yao, X. (2014b). On investigation of interdependence between sub-problems of the travelling thief problem. *Soft Computing*, pages 1–16.

[62] Michalewicz, Z. and Fogel, D. B. (2013). *How to solve it: modern heuristics*. Springer Science & Business Media.

[63] Michalewicz, Z., Ibrahimov, M., Mohais, A., Schellenberg, S., and Wagner, N. (2009). Applications of evolutionary methods for complex industrial problems.

[64] Muhlenbein, H. (1991). Evolution in time and space-the parallel genetic algorithm. In *Foundations of genetic algorithms*. Citeseer.

[65] Nesmachnow, S. (2014). An overview of metaheuristics: accurate and efficient methods for optimisation. *International Journal of Metaheuristics*, 3(4):320–347.

[66] Ochoa, G., Tomassini, M., Vérel, S., and Darabos, C. (2008). A study of nk landscapes' basins and local optima networks. In *Genetic and Evolutionary Computation Conference*, pages 555–562. ACM.

[67] Ochoa, G., Verel, S., Daolio, F., and Tomassini, M. (2014). Local optima networks: A new model of combinatorial fitness landscapes. In *Recent Advances in the Theory and Application of Fitness Landscapes*, pages 233–262. Springer.

[68] Papadimitriou, C. H. (1977). The euclidean travelling salesman problem is np-complete. *Theoretical computer science*, 4(3):237–244.

[69] Perl, J. and Daskin, M. S. (1985). A warehouse location-routing problem. *Transportation Research Part B: Methodological*, 19(5):381 – 396.

[70] Polyakovskiy, S., Bonyadi, M. R., Wagner, M., Michalewicz, Z., and Neumann, F. (2014). A comprehensive benchmark set and heuristics for the traveling thief problem. In *Genetic and Evolutionary Computation Conference*, pages 477–484. ACM.

[71] Polyakovskiy, S., Makarowsky, A., and M'Hallah, R. (2017). Just-in-time batch scheduling problem with two-dimensional bin packing constraints. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 321–328. ACM.

[72] Polyakovskiy, S. and Neumann, F. (2017). The packing while traveling problem. *European Journal of Operational Research*, 258(2):424–439.

[73] Przybylek, M. R., Wierzbicki, A., and Michalewicz, Z. (2016). Multi-hard problems in uncertain environment. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 381–388. ACM.

[74] Qu, R., Pham, N., Bai, R., and Kendall, G. (2015). Hybridising heuristics within an estimation distribution algorithm for examination timetabling. *Applied Intelligence*, 42(4):679–693.

[75] Reinelt, G. (1991). Tsplib–a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384.

[76] Reinelt, G. (1992). Fast heuristics for large geometric traveling salesman problems. *ORSA Journal on computing*, 4(2):206–217.

[77] Renaud, J., Laporte, G., and Boctor, F. F. (1996). A tabu search heuristic for the multi-depot vehicle routing problem. *Computers & Operations Research*, 23(3):229–235.

[78] Ross, P. (2005). *Hyper-Heuristics*, pages 529–556. Springer US, Boston, MA.

[79] Schellenberg, S., Li, X., and Michalewicz, Z. (2016). Benchmarks for the coal processing and blending problem. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*, pages 1005–1012. ACM.

[80] Silva, S. and Almeida, J. (2005). Gplab-a genetic programming toolbox for matlab. In *Proceedings of the Nordic MATLAB Conference (NMC-2003)*, pages 273–278.

[81] Soubeiga, E. (2003). *Development and Application of Hyperheuristics to Personeel Scheduling*. Phd thesis, School of Computer Science and Information Technology, University of Nottingham.

[82] Stolk, J., Mann, I., Mohais, A., and Michalewicz, Z. (2013). Combining vehicle routing and packing for optimal delivery schedules of water tanks. *OR Insight*, 26(3):167–190.

[83] Tavares, J., Pereira, F. B., and Costa, E. (2008). Multidimensional knapsack problem: A fitness landscape analysis. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(3):604–616.

[84] Uludağ, G., Kiraz, B., Etaner-Uyar, A. Ş., and Özcan, E. (2012). *A Framework to Hybridize PBIL and a Hyper-heuristic for Dynamic Environments*, pages 358–367. Springer, Berlin, Heidelberg.

[85] Verel, S., Ochoa, G., and Tomassini, M. (2008). The connectivity of nk landscapes' basins: A network analysis. In *Artificial Life XI: 11th International Conference on the Simulation and Synthesis of Living Systems*, pages 648–655.

[86] Verel, S., Ochoa, G., and Tomassini, M. (2011). Local optima networks of nk landscapes with neutrality. *IEEE Transactions on Evolutionary Computation*, 15(6):783–797.

[87] Wagner, M. (2016). Stealing Items More Efficiently with Ants: A Swarm Intelligence Approach to the Travelling Thief Problem. In *10th Int. Conference on Swarm Intelligence, ANTS 2016*, pages 273–281, Brussels, Belgium. Springer.

[88] Wang, S. and Sarker, B. R. (2006). Optimal models for a multi-stage supply chain system controlled by kanban under just-in-time philosophy. *European Journal of Operational Research*, 172(1):179–200.

[89] Watson, J.-P. (2010). An introduction to fitness landscape analysis and cost models for local search. In *Handbook of metaheuristics*, pages 599–623. Springer.

[90] Watson, J.-P., Whitley, L. D., and Howe, A. E. (2005). Linking search space structure, run-time dynamics, and problem difficulty: A step toward demystifying tabu search. *J. Artif. Intell. Res.(JAIR)*, 24:221–261.

[91] Whitley, D., Hains, D., and Howe, A. (2009). Tunneling between optima: partition crossover for the traveling salesman problem. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 915–922. ACM.

[92] Whitley, D., Hains, D., and Howe, A. (2010). A hybrid genetic algorithm for the traveling salesman problem using generalized partition crossover. In *Parallel Problem Solving from Nature, PPSN XI*, pages 566–575. Springer.

[93] Whitley, D., Starkweather, T., and Shaner, D. (1991). *The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination.* Citeseer.

Résumé

Plusieurs problèmes d'optimisation réels sont composés de multiples sous-problèmes en interaction. Cependant, peu de chercheurs se penchent sur la résolution de ces problèmes en utilisant les métaheuristiques et le calcul évolutionnaire. Dans cette thèse, nous nous concentrons sur l'étude de problèmes avec multiple sous-problèmes interdépendants. Pour atteindre notre objectif, nous commençons par proposer des problèmes inspirés par la logistique et la gestion de la chaîne d'approvisionnement. Ensuite, nous fournissons des définitions formelles des différents types de dépendances dans les problèmes multi-composants. Par la suite, deux études de cas sont réalisées. La première se focalise sur le problème du voyageur voleur, qui est un modèle de référence introduit pour aider les chercheurs à étudier l'interdépendance dans les problèmes du monde réel. Le problème est présenté, analysé sous différentes perspectives, et des solutions heuristiques, métaheuristiques et hyper-heuristiques sont proposées pour y faire face. La deuxième étude de cas est un problème inspiré de l'optimisation des chaînes d'approvisionnement réelles. Nous introduisons le problème dans le but de proposer un problème de référence plus réaliste qui illustre une classe différente de dépendance entre composantes.

Mots-clés: Interdépendance, Problèmes multi-composants, Métaheuristiques, Optimisation Combinatoire

Abstract

Many real-world optimisation problems are composed of multiple interacting sub-problems. However, few researchers look into tackling these problems using metaheuristics and evolutionary computation. In this thesis, we focus on studying problems with multiple interdependent sub-problems. To achieve our goal, we start by providing examples of these problems that are inspired from logistics and supply chain management. Then, we provide formal definitions of different types of dependencies in multi-component problems. Afterwards, two case studies are carried out. The first centers on the travelling thief problem, a benchmark model introduced to help researchers investigate the interdependence in real-world problems. The problem is introduced, analysed from different perspectives, and heuristic, metaheuristic, and hyperheuristic solutions are proposed to tackle it. The second case study is a problem inspired by real-world supply chain optimisation. We introduce the problem with the aim of proposing a more realistic benchmark problem that embedes a different class of dependency between components.

Keywords: Interdependence, Multi-component problems, Metaheuristics, Combinatorial optimisation.

Année Universitaire : 2017-2018