Algorithms for elementary path problem : application to kidney exchange

Lucie Pansart

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : Mathématiques et Informatique

Arrêté ministériel : 25 mai 2016

Présentée par

Lucie PANSART

Thèse dirigée par **Hadrien CAMBAZARD**
et codirigée par **Nicolas CATUSSE**, Grenoble INP

préparée au sein du **Laboratoire des Sciences pour la Conception, l'Optimisation et la Production de Grenoble** dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Algorithmes de chemin élémentaire : application aux échanges de reins

Algorithms for elementary path problem: application to kidney exchange

Thèse soutenue publiquement le **18 septembre 2020**,
devant le jury composé de :

Monsieur HADRIEN CAMBAZARD
MAITRE DE CONFERENCES, GRENOBLE INP, Directeur de thèse
Monsieur LOUIS-MARTIN ROUSSEAU
PROFESSEUR, ECOLE POLYTECHNIQUE DE MONTREAL - CANADA, Rapporteur
Monsieur FRANÇOIS CLAUTIAUX
PROFESSEUR DES UNIVERSITES, UNIVERSITE DE BORDEAUX, Rapporteur
Madame ANA VIANA
PROFESSEUR, ECOLE POLYTECHNIQUE DE PORTO - PORTUGAL, Présidente
Monsieur MAXIME OGIER
MAITRE DE CONFERENCES, UNIVERSITE DE LILLE, Examinateur
Monsieur NICOLAS CATUSSE
MAITRE DE CONFERENCES, GRENOBLE INP, Examinateur

Contents

Remerciements

Il est important pour moi de remercier ceux qui ont contribué à faire de cette thèse ce qu'elle est aujourd'hui. Tout d'abord merci à mes rapporteurs, Louis-Martin Rousseau et François Clautiaux, à mes examinateurs, Maxime Ogier et Ana Viana. Merci pour votre temps, vos remarques intéressantes et vos questions pertinentes. Cette thèse n'aurait jamais eu lieu si Gautier Stauffer n'avait pas croisé ma route. Merci d'avoir proposé de me diriger dans cette thèse au sujet passionnant.

Sur cette route surtout il y a eu Hadrien et Nicolas. Travailler avec vous est une expérience incroyable et j'espère que beaucoup d'autres doctorants et stagiaires auront la chance de la vivre. Merci Hadrien pour ton enthousiasme, ton fourmillement d'idées et ta passion. Je te suis reconnaissante d'avoir repris le flambeau de la direction de thèse. Merci Nicolas pour ton écoute, ton acuité et ton assurance communicative. J'ai tellement aimé travailler avec vous et je vous remercie pour tout ce que vous m'avez appris, mais aussi pour votre bienveillance et votre soutien sans faille.

Nicolas, mon ami, mon binôme, tu as fait l'erreur de partir dans le continu, et même pire : le continu en physique ! Je ne t'en veux pas mais je ne désepère pas de te faire ouvrir les yeux un jour sur la beauté de l'optimisation discrète. En attendant, j'espère que tu sais à quel point je te remercie d'être toujours là pour moi. Tu réponds toujours présent. Pour travailler ou pour discuter, depuis 7 ans déjà. Et nos longues discussions sur notre avenir et celui de la recherche ont été pour moi comme des bonbons pendant toutes ces années.

Maintenant moi aussi je suis docteure, comme ma grande soeur ! Merci à toi de m'avoir ouvert le chemin. Merci à mon grand frère de m'avoir montré que "quand on veut, on peut". Merci maman pour ton oreille attentive et nos longues discussions. Pour tous tes conseils qui m'ont guidée dans le brouillard du doute. Merci papa, car, que ce soit pour parler, jouer, boire un verre ou en appui logistique, tu as toujours été disponible, me permettant de conserver mon énergie pour ma recherche. Merci à mes grands-parents. J'aurais aimé que vous soyez encore là pour me voir arriver au bout. Merci

aussi à ma belle-famille pour votre intérêt et votre curiosité sur mon travail, ainsi que pour votre soutien.

Cette thèse aurait été très différente si elle ne s'était pas déroulée au laboratoire G-SCOP. J'ai trouvé là-bas de merveilleux collègues, mais aussi, comme une deuxième famille, de nombreuses amitiés qui dureront toujours. Tous ces collègues et amis ont rendu agréable chaque jour de travail, même lorsque je cherchais sans trouver, même lorsque mon code était bugué, même lorsque l'écriture n'avançait pas...

Il y a ceux qui m'ont simplifié la vie. Merci Marie-Jo, Kevin, Olivier S., Myriam T., Fadila.
Il y a ceux qui m'ont écoutée, soutenue, conseillée. Merci Nadia, Louis, Bernard.
Il y a ceux qui m'ont expliqué, raconté. Merci Pierre, Olivier B.
Il y a ceux qui m'ont fait (beaucoup, beaucoup) rire. Merci Nicolas Béraud, Lisa, Cédric, Margaux, Cléophée, Pascal, Cléa, Victor, Maud, Valentins, Astor, Florence, Florian, Nicolas Bousquet, Parisa, Tom, Sylvain, Élodie.

Et il y a ceux...
Clément tu as été le témoin de mon évolution, depuis mes premiers balbutiements de stagiaire jusqu'à ce diplôme de docteur. Merci pour la douceur de ton amitié.
Enfin, merci du fond du coeur aux soleils de ma vie, qui transforment chaque jour de pluie en un merveilleux Arc-en-Ciel : Alexandre, Aurélie, Laura, Lucas, Lucile, Matthieu.
Et Gricha.

Introduction (en français)

Dans un marché de troc, les participants s'échangent des biens ou des services sans utiliser de moyen de paiement. En général, le troc se passe entre deux personnes, localement et immédiatement, et sans l'intervention d'une organisation étatique. Il y a cependant une exception: les échanges de reins sont organisés par les institutions gouvernementales et peuvent impliquer beaucoup de monde. Les organes ne peuvent pas être vendus, ni échangés d'ailleurs: dans un ***programme d'échange de reins***, ce qui est échangé ce sont les **donneurs**, pas les reins. Les acteurs du marché sont les patients attendant une greffe de rein à cause d'une maladie rénale. Chaque patient est associé à un proche prêt à donner un rein, mais avec qui il est incompatible. De plus, de nouveaux reins peuvent être injectés dans le marché grâce aux donneurs altruistes. Le rôle des programmes d'échange de reins est de déterminer quels échanges doivent être réalisés, dans le but d'optimiser le "bien commun", tout en respectant les contraintes médicales, légales, éthiques et logistiques. En d'autres termes, il doivent résoudre un ***problème d'optimisation combinatoire***.

L'optimisation combinatoire est une branche des mathématiques qui étudie comment trouver un meilleur élément parmi un ensemble, fini et discret, d'éléments. Généralement, cet ensemble ne peut pas être décrit explicitement ; soit parce qu'il difficile de connaître les éléments qu'il contient[1], soit parce que ces éléments sont trop nombreux[2]. Il est donc plutôt défini par une liste de contraintes que ses éléments, les ***solutions réalisables***, doivent respecter. Cette définition alternative ne change pas la structure du problème et des méthodes avancées doivent être utilisées pour gérer l'***explosion combinatoire*** de l'espace de recherche. Les problèmes d'optimisation ne sont pas (seulement) des jeux abstraits pour des mathématiciens qui s'amusent à les résoudre. Ils apparaissent dans beaucoup d'applications réelles rencontrées par la ***recherche opérationnelle*** : ordonnancement, tournées de véhicules, choix d'implantation d'usines, planification de production, conception de réseau, affectation d'équipes... et

[1] Avez-vous déjà essayé de résoudre un sudoku ? Il n'y a qu'une seule bonne solution...
[2] Pour livrer 13 clients, un VRP peut emprunter plus de 6 milliards de trajets différents.

échanges de reins !

Ces dernières années, les programmes d'échange de reins sont devenus un sujet de recherche important pour les médecins et législateurs, mais aussi pour les mathématiciens et informaticiens, confrontés à différentes questions à propos de l'efficacité, de l'équité et du fonctionnement de ces programmes. En Europe, un réseau de chercheurs de tous ces domaines travaillent ensemble pour partager leurs retour d'expériences, établir des bonnes pratiques et développer un programme transnational. Cette action est supporté par COST (European Cooperation in Science and Technology) [32]. Pour le grand public, ce sujet est surprenant, un peu mystérieux[3], mais aussi à la mode, puisqu'il traite des ***algorithmes***. En 2017, la chercheuse française Claire Mathieu a donné un cours[4] sur les algorithmes au Collège de France, un établissement d'enseignement et de recherche prestigieux et ouvert à tous, et son exemple introductif portait justement sur les échanges de reins.

Problématique

La plupart des programmes d'échange de reins autorisent aujourd'hui les donneurs altruistes, donc les échanges de reins ne sont pas seulement les cycles de dons entre pairs patients-donneurs, mais aussi des chaînes de dons initiées par ces donneurs altruistes. De plus, ils contiennent de plus en plus de patients à mesure que la pratique se répand dans les hôpitaux. Le plus grand programme européen enregistre actuellement 250 pairs patients-donneurs, mais on s'attend à des milliers d'entre eux dans les prochaines années. Cette évolution des programmes d'échange de reins est bénéfique pour les patients car elle augmente leur chance de trouver un donneur compatible. D'un point de vue computationnel en revanche, cela rend le ***problème d'échange de reins*** plus difficile à résoudre, car la structure des solutions réalisables (les échanges de reins) devient plus complexe et leur nombre explose. Comment résoudre efficacement le problème d'échange de reins dans ce contexte est au cœur de cette thèse, mais plusieurs autres sujets gravitent autour de cette question. La modélisation du problème nous conduit à considérer des problèmes de packing, de tournées de véhicules et de stables. Avant tout, la résolution du problème nous amène à étudier des problèmes de chemin élémentaire, en particulier dans le cadre de la génération de colonnes.

Contributions

Les contributions de cette thèse concernent quatre sujet différents mais connexes:

[3] Je ne compte plus les rires gênés de mes interlocuteurs lorsqu'ils essayent de déterminer si je suis une trafiquante d'organes.

[4] Disponible en ligne: https://www.college-de-france.fr/site/claire-mathieu

1. Nous proposons une nouvelle formulation pour le problème du stable qui peut être utilisé pour modéliser le problème d'échange de reins. C'est une formulation étendue basée sur une construction des stables par voisinages. Nous caractérisons sont polytope et prouvons qu'il est est meilleur (plus *tight*) que le polytope des cliques. La formulation est donc idéale pour les graphes parfaits, mais nous montrons aussi qu'elle est compacte pour les graphes sans griffe.
2. Nous concevons un algorithme de génération de colonnes pour résoudre le problème d'échange de reins par la *formulation échange* (ou formulation cycle). C'est une formulation exponentielle qui a été beaucoup étudiée pour des programmes ne contenant que des cycles de dons. La NP-difficulté du problème de pricing lorsque les chaînes de dons sont impliquées n'a cependant été prouvée que récemment (en 2016 par Plaut *et al.* [104]). Nous nous concentrons donc sur l'intégration des donneurs altruistes dans des programmes de grande taille et parvenons à fournir un écart de moins de 0.2% entre notre solution et la solution optimale, sur des instances de presque 800 participants. Une partie de ce travail a été publié dans les actes de la conférence MOSIM 2018 [**I**].
3. Nous étudions le problème de chemin élémentaire minimum avec contrainte de taille et adaptons deux programmes dynamiques de la littérature pour le résoudre. Ces adaptations impliquent de tirer parti de la structure du graphe mais aussi du problème lui-même, utilisant la contrainte de taille pour réduire l'espace de recherche. Nos expériences montrent l'intérêt de ces algorithmes pour résoudre le problème, notamment dans une génération de colonnes.
4. Nous présentons de nouvelles stratégies pour le *color coding*, une heuristique randomisée cherchant des sous-graphes, en particulier des chemins élémentaires, dans un graphe. Au lieu d'utiliser la loi uniforme discrète, nous proposons de nouvelles lois de probabilité qui utilisent la structure du graph pour augmenter les chances de trouver une solution optimale. Nous prouvons que cette probabilité est effectivement améliorée pour n'importe quel graphe et atteint 1 dans certaines cas particuliers. En pratique, des graphes de la littérature avec plus de 300 sommets sont toujours résolus à l'optimum. Ce travail a été accepté à la conférence ECAI 2020 [**II**].

Structure de la thèse

Les notions de théorie des graphes et de programmation mathématique nécessaires à la compréhension de cette thèse sont rappelées dans le Chapitre 1 et les notations qui y sont introduites sont valides dans tout le manuscrit. Nous expliquons dans le Chapitre 2 le fonctionnement des programmes d'échange de reins, ainsi que le contexte de leur création et

de leur développement. Une revue de littérature sur les différentes questions mathématiques qui se posent dans ces programmes est ensuite présentée. Finalement, ce chapitre se concentre sur le ***problème d'échange de reins***, en particulier sa définition mathématique et sa modélisation sous forme de graphe. Des équivalences avec d'autres problèmes d'optimisation connus sont aussi proposées, en particulier une réduction avec le problème de tournées de véhicules. Dans le Chapitre 3, nous présentons une vue d'ensemble des différents ***programmes linéaires en nombres entiers*** pour le KEP, en particulier notre nouvelle formulation étendue problème de stable. Le Chapitre 4 se concentre sur l'un d'entre eux, la *formulation échange* et sur la conception d'un algorithme efficace de ***génération de colonnes*** pour la résoudre. Cet algorithme est évalué et comparé avec une formulation compacte. La résolution du problème de pricing, dont la preuve de NP-difficulté est rappelée dans ce chapitre, nécessite de faire face à un ***problème de chemin élémentaire***, qui est le sujet du Chapitre 5. Nous le définissons d'abord formellement avant de présenter un état de l'art sur les problèmes similaires, en particulier lorsqu'ils sont imbriqués dans un schéma de génération de colonnes pour les problèmes de tournées de véhicules. Nous expliquons ensuite comment exploiter la structure du graphe et du problème pour adapter les algorithmes de la littérature et rendons compte de leurs résultats expérimentaux. En particulier, nous adaptons deux programmes dynamiques et l'un d'entre eux, le ***color coding*** est étudié dans le Chaptitre 6. L'étude théorique de la probabilité de trouver un chemin optimal en utilisant cette heuristique randomisée est suivie des résultats expérimentaux sur la fréquence à laquelle cela arrive en pratique.

Chaque chapitre zoome en fait sur une sous-partie du chapitre précédent, ce qui donne une structure *en cascade* à la thèse. Nous recommandons donc fortement au lecteur de suivre le plan proposé, même si chaque chapitre est également pensé pour être indépendant. Remarquez qu'au contraire les cadres expérimentaux sont impactés par les résultats du chapitre d'après, selon une intégration *ascendante*. En effet, les expériences d'un chapitre incluent l'algorithme qui est étudié dans le chapitre suivant, mais alors seulement sa meilleure configuration est prise en compte. Cette dernière est déterminée grâce à l'évaluation expérimentale de l'algorithme, donc dans le chapitre d'après. Le schéma 2 résume la structure de la thèse et comment les chapitres sont liés.

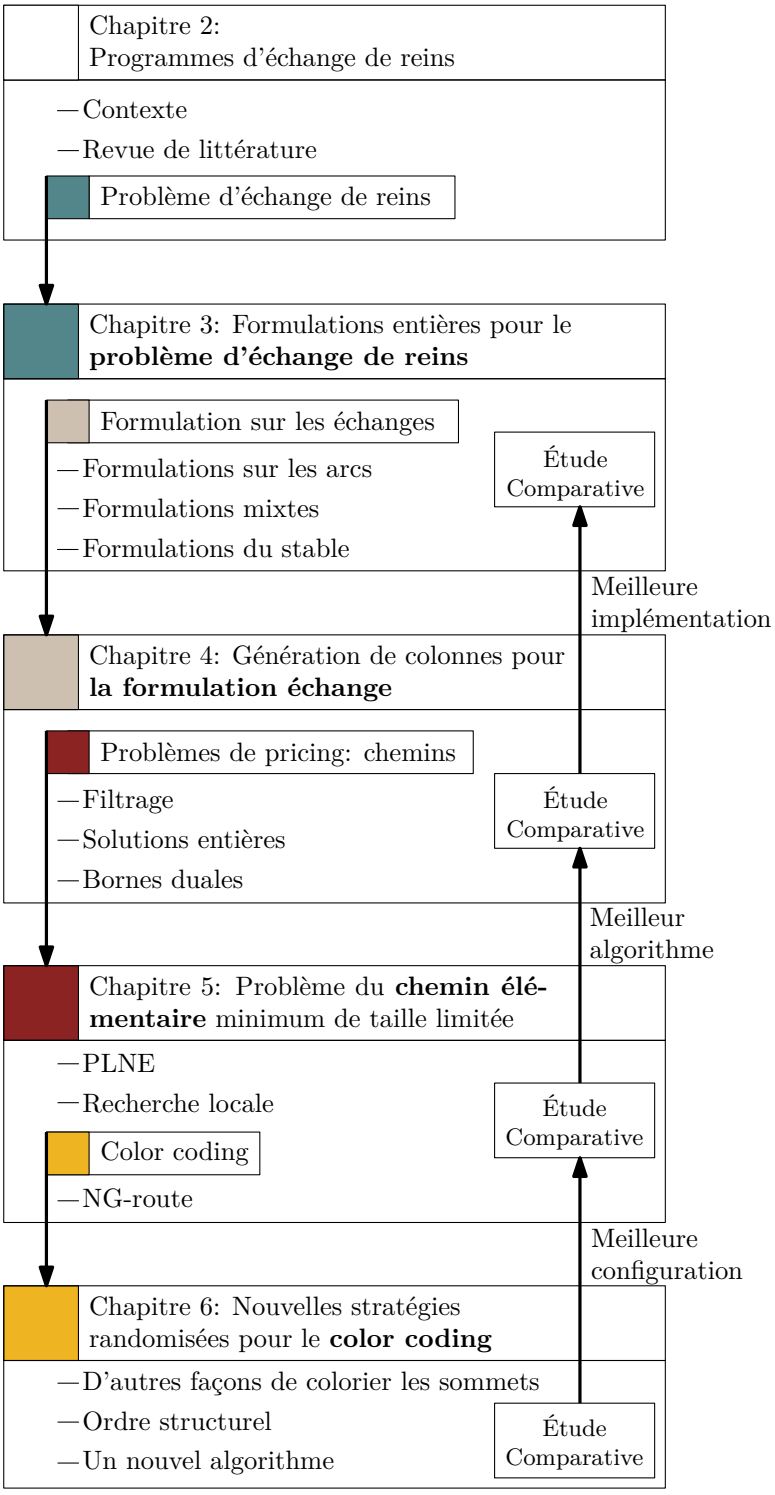


Figure 1 – Structure de la thèse

Introduction

In a barter market, participants trade goods or services without using money or any other medium of exchange. Usually, barter takes place locally, immediately, between two people and without a state organization. One barter market though is an exception: kidney exchanges are organized by countries' institutions themselves and can involve many people. Organs cannot be sold, and actually cannot be exchanged either: in a kidney exchange program, the traded "items" are the **donors**, not the kidneys. Agents of the market are patients waiting for a kidney transplant because of a renal disease. Each patient has a relative ready to donate one kidney, but incompatible. In addition, new items can be injected in the market thanks to altruistic donors. The role of kidney exchange programs is to find out which exchanges should be carried out, in order to maximize the "common good" while respecting medical, ethical, legal and logistical constraints. In other words, they must solve a ***combinatorial optimization problem***.

Combinatorial optimization is a mathematical field that studies how to find an optimal object from a finite and discrete set of objects. Usually this set cannot be explicitly described, either because it is hard to know which objects it contains[5] or because these objects are too many[6]. Instead, it is defined by a list of constraints that the objects, called the ***feasible solutions***, must respect. This different definition does not change the structure of the problem and advanced methods must be applied to deal with the ***combinatorial explosion*** of the search space. Combinatorial optimization problems are not (only) abstract games that mathematicians take pleasure to solve. They appear in many real life applications faced by the ***operations research***: task scheduling, vehicle routing, facility locations, production planning, network design, crew assignment... and kidney exchanges!

In recent years, kidney exchange programs have become an important research topic for physicians and lawmakers, but also for mathematicians and computer scientists, confronted to different questions about their efficiency, fairness and functioning. In Europe, a network of researchers from

[5] Have you ever tried to solve a sudoku? There is only one correct solution...

[6] To visit 13 clients, a traveling salesman can take more than 6 billions different routes.

all these fields are working together to share feedbacks, establish best practices and develop transnational programs in an action supported by COST (European Cooperation in Science and Technology) [32]. For the general public, this subject is surprising, a bit mysterious[7], but also trendy, as it comes to ***algorithms***. In 2017, the french researcher Claire Mathieu gave a lecture[8] on algorithms at the Collège de France, a prestigious education and research establishment open to all, and its introductory example was precisely on kidney exchanges.

Problematic

Most of kidney exchange programs nowadays authorized altruistic donors, so kidney exchanges are not only cycles of donations between patient-donor pairs, but also chains of donations initiated by these altruistic donors. They also involve more and more patients as the usage is spreading in hospitals. The largest program in Europe currently registers 250 patient-donor pairs, but we expect future programs to include thousands of them. This evolution is beneficial for patients as it increases their chance to find a compatible donor. From a computational point of view however, its makes the ***kidney exchange problem*** harder to solve as the structure of feasible solutions, the kidney exchanges, is more complex, and their number explodes. How to efficiently solve the kidney exchange problem in this context is the core question of this thesis, but several other topics gravitate around this question. Modeling the problem leads us to investigate packing problems, vehicle routing problems and stable set problems. Above all, solving the problem drives us to study elementary path problems, in particular in the framework of a column generation.

Contributions

The main contributions of this thesis concern four different, but related, topics:

1. We propose a new formulation for the stable set problem that can be used to model the kidney exchange problem. It is an extended formulation based on a construction of the stable set by neighborhood. We characterize its polytope and prove it is tighter than the clique relaxation polytope. The formulation is thus ideal for perfect graphs, but we also show it is compact for claw-free graphs.
2. We design a column generation algorithm to solve the kidney exchange problem with the *exchange formulation* (or cycle formulation). This formulation is a large-scale model that was deeply studied for programs

[7] I've lost the count of awkward laughters from my interlocutors when they try to figure out if I am an organ trafficker.

[8] Available at https://www.college-de-france.fr/site/claire-mathieu

containing only cycles of donation. However the NP-hardness of the pricing problem when chains of donation are included was proved recently (in 2016 by Plaut *et al.* [104]). We thus focus on integrating altruistic donors in large size kidney exchange programs and manage to get a gap smaller than 0.2% between our solution and the optimal, even for instances with almost 800 participants. A part of this work is published in the proceedings of the MOSIM 2018 conference [**I**].

3. We study the elementary minimum path problem with length constraint and adapt two dynamic programs from the literature to solve it. These adaptations involve to take profit from the graph structure but also from the problem itself, using the length constraint to reduce the search space. Experiments show the benefit of these algorithms to solve the problem, especially in a column generation scheme.
4. We introduce new strategies for the color coding, a randomized heuristic finding subgraphs, in particular elementary paths, in a graph. Instead of using the discrete uniform distribution, we propose new probability distributions that take advantage of the graph structure in order to increase the chance to find an optimal solution. We prove that this probability is indeed improved for any graph and reaches 1 for some particular cases. In practice, some graphs from the literature with more than 300 vertices are always optimally solved. This work has been accepted to the ECAI 2020 conference [**II**].

Outline

Notions of graph theory and mathematical programming which are necessary for the understanding of this thesis are reminded in Chapter 1, and the notations introduced in this chapter are valid throughout all the manuscript. We explain in Chapter 2 the functioning of kidney exchange programs, as well as the context of their creation and development. A literature review on the different mathematical subjects arising in these programs is then presented. Finally, this chapter focuses on the NP-complete ***kidney exchange problem***, in particular its mathematical definition and graph modeling. Equivalences with other famous optimization problems are also provided and we propose a reduction with a vehicle routing problem. In Chapter 3 we make a survey of the different ***integer programming models*** for the kidney exchange problem, including our new extended formulation for the stable set problem. Chapter 4 focuses on one of them, the *exchange formulation*, and the design of an efficient ***column generation*** algorithm to solve it. This algorithm is evaluated and compared with a compact formulation. The proof of NP-hardness of the pricing problem is also recalled in this chapter. Solving it implies to deal with an ***elementary path problem***, which is the subject of Chapter 5. We first define it formally and present the state of the art on similar problems, in particular those embedded in

column generation schemes of vehicle routing problems. Then, we present how to exploit the graph and problem structures to adapt algorithms from the literature and report experiments on these algorithms. In particular, we adapt two dynamic programs and one of them, the ***color coding***, is studied in Chapter 6. The theoretical study of the probability to find an optimal path using this randomized heuristic is followed by computational results on the frequency it happens in practice.

Each chapter actually zooms in on a subpart of the previous chapter, such that the thesis is structured from the top down. Thus, we strongly encourage the reader to follow the proposed plan, even if each chapter is also written to be self-contained. Note that, on the contrary, experimental settings are affected by results of the next chapter, following a bottom up integration. Indeed, the experimentations of a chapter include the algorithm studied in the next chapter, but only its best configuration is taken into account. The latter is determined thanks to the experimental evaluation of the algorithm, so in the next chapter. Figure 2 sums up the outline and how chapters are linked. Complete experimental results can be found on my webpage: https://pagesperso.g-scop.grenoble-inp.fr/~pansartl/.

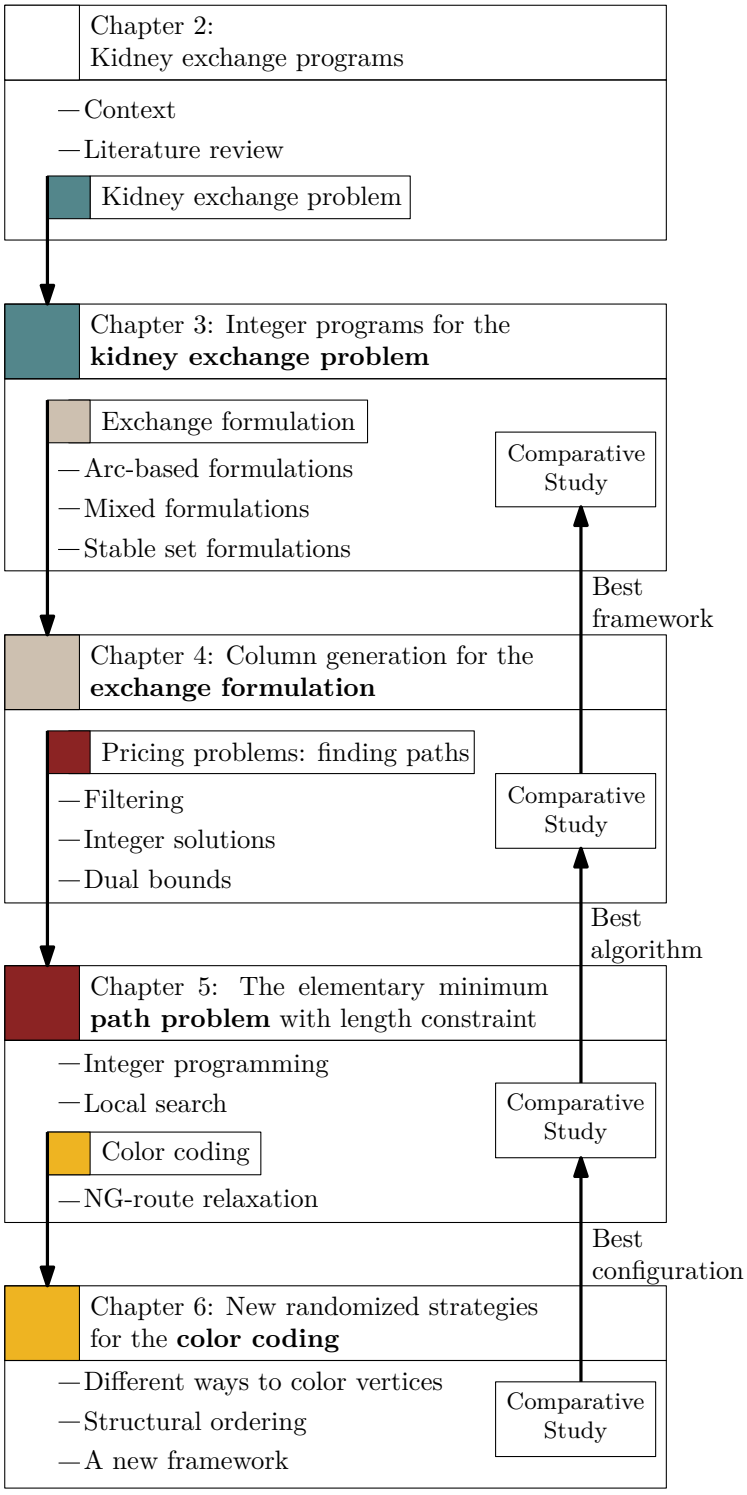


Figure 2 – Structure of the thesis

Chapter 1

Operations research prerequisites

This chapter goes over the basic knowledge in graph theory and linear optimization used in this thesis. It also defines the notations adopted in all our work. You can find an index of the notations and definitions at the end of the manuscrit.

1.1 Graph theory

Directed and undirected graph. A *finite simple graph* $G = (V, E)$ is an ordered pair of a finite *vertex* set V and a finite *edge* set E. The graph can be either *undirected* or *directed*. In the latter case, we can refer to edges also as *arcs* and to the edge set as A. Given an edge $e = (uv)$, we say that u and v are the *endpoints* of e and that e is *incident* to u and v. This also implies that u and v are *adjacent* or *neighbors*. When the graph is directed, we say that u is the *head* and v the *tail* of e and that v is a *successor* of u while u a *predecessor* of v. When the graph is undirected the edge is unordered and $e = (uv) = (vu)$. In this thesis, a **graph** refers to a finite simple graph (directed or not). Given a graph G, we denote by $V(G)$ its set of vertices and by $E(G)$ (or $A(G)$) its set of edges (resp. arcs). In the following, let $G = (V, E)$ be a graph.

Neighborhoods. The **neighborhood** of $u \in V$, denoted by $N_G(u)$, is the set of neighbors of u. The **closed neighborhood** of $u \in V$, denoted by $N_G[u]$, is defined as $\{u\} \cup N_G(u)$. $\delta_G(u)$ denotes the set of incident edges to vertex u. If G is directed, $N_G(u) = N_G^+(u) \cup N_G^-(u)$ where $N_G^+(u)$ is the set of successors of u: $N_G^+(u) := \{v \in V : (uv) \in A\}$ and $N_G^-(u)$ the set of its predecessors $N_G^-(u) := \{v \in V : (vu) \in A\}$. Similarly, $\delta_G(u) = \delta_G^+(u) \cup \delta_G^-(u)$ where $\delta_G^+(u)$, is the set of outgoing edges: $\delta_G^+(u) := \{e \in A : e = (uv)\}$ and

$\delta_G^-(u)$ the set of its incoming edges $\delta_G^-(u) := \{e \in A : e = (vu)\}$. When the context is non ambiguous, we can remove the G index.

Subgraph and induced subgraph. A *subgraph* of G is a graph H such that $V(H) \subseteq V$ and $E(H) \subseteq E$. Given a set $U \subseteq V$, the **subgraph induced** by U, denoted by $G[U]$, is the graph on vertex set U and edge set $E[U]$ having both endpoints in U, $E[U] := \{(uv) \in E : u, v \in U\}$. For any graph H, the graph G is said **H-free** if it does not contain H as an induced subgraph.

Paths and cycles. A *walk* is a sequence of vertices such that each vertex is adjacent to its neighbors in the sequence. A *trail* is a walk without repeated edges. A **simple path** (or **elementary path**, or simply a **path**), is a trail without repeated vertices. The **length** of a path is the number of edges it contains. A *circuit* is a trail such that the first and the last vertices of the sequence are the same. A **cycle** is a circuit without repeated vertices. It can also be defined as a path having the same first and last vertex.

Stable set, clique, coloring. A **stable set** (also called independent set) of G is a set of pairwise non adjacent vertices of V. The **stability number** of G, denoted by $\boldsymbol{\alpha(G)}$, is the maximum cardinality of a stable set of G. A *clique* of G is a set of pairwise adjacent vertices of V. The *clique number* of G, denoted by $\boldsymbol{\omega(G)}$, is the cardinality of the largest clique of G. A *proper vertex coloring* of G is a labeling of the vertices such that two adjacent vertices do not have the same label. The minimal number needed to make a proper vertex coloring of G is called the *chromatic number* and denoted by $\boldsymbol{\chi(G)}$.

Particular graphs. We denote by $K_{1,k}$ the complete bipartite graph, also called a *k-star*, which contains one vertex adjacent to k other non-adjacent vertices. The graph $K_{1,3}$ is called a **claw**.

The class of perfect graphs was defined by Berge in the 1960's in the context of graph coloring. A graph is **perfect** if and only if the chromatic number of every induced subgraph H equals the clique number of H.

Usual notations. Given a set $U \subseteq V$ and any vector $a \in \mathbb{R}^{|V|}$, a_U is the restriction of a to U, that is $(a_v)_{v \in U}$. We denote by $\chi^U \in \{0,1\}^{|V|}$ its characteristic vector, that is $\chi^U(v) = 1 \Leftrightarrow v \in U$.

Given a set S of elements, $\boldsymbol{\mathcal{P}(S)}$ denotes the power set of S, *i.e.*, the set of all subsets of S.

1.2 Quick reminders on complexity

A decision problem is a problem where a question is asked about the values of an input and can be answered only by "yes" (affirmative answer) or "no" (negative answer). A **combinatorial optimization problem** aims at finding an **optimal** object from a finite set of objects with respect to a **criteria**. It is defined by an instance $\mathcal{I}$, a set of feasible solutions $\mathcal{F}$, a function $m : \mathcal{F} \to \mathbb{R}$ and a goal (min or max). For example, in the shortest path problem the instance is any graph $G = (V, E)$, the set of feasible solutions is the set containing all paths of G from a given source $s \in V$ to a destination $t \in V$, the value m of a path is its cost and the goal is to minimize it. The associated decision problem would ask if there exists a path between s and t with a cost of at most M.

Several algorithms can solve a same problem, hence the need to compare and classify them. The efficiency of an algorithm is measured by the number of operations needed to solve any instance of size n, and commonly depicted with the big O notation. The **big $\mathcal{O}$ notation** provides an asymptotic bounding of an algorithm running time: we say that an algorithm runs in $\mathcal{O}(g(n))$ for a given real valued function g if and only if there exists $n_0, M \in \mathbb{R}^+$ such that the number of operations of the algorithm is bounded by $Mg(n)$ for any $n \geq n_0$.

A **polynomial time** algorithm has its running time bounded by a polynomial in the input size n. On the contrary, the running time of an **exponential time** algorithm cannot be bounded by such a polynomial. A problem is said to be belong to **P** if it can be solved with a polynomial time algorithm. The complexity class **NP** gathers all decision problems for which an affirmative answer can be verifiable in polynomial time. By abusing notation, we say that an optimization problem belongs to NP if its corresponding decision problem is in NP. A problem is **NP-hard** if its solution can be transformed in polynomial time into a solution of any NP problem. When a problem is both NP-hard and in NP, it is said to be **NP-complete**. Under the hypothesis that P $\neq$ NP, there does not exist a polynomial time algorithm for a NP-hard problem.

1.3 Linear optimization background

1.3.1 Linear programming problem

A linear programming problem, or **linear program**, aims at finding a vector that maximizes (or minimizes) a linear objective function subject to linear constraints. These constraints are expressed by linear equality and inequality and define a polyhedron. Formally, a linear program can be written $\max\{cx | Ax \leq b, x \in \mathbb{R}^n\}$ where A is a matrix $\mathbb{R}^{n \times m}$, $b \in \mathbb{R}^m$ and c is a **line** vector $\in \mathbb{R}^n$. The linear system $\{Ax \leq b\}$ defines a polyhedron P and the

problem can be stated as $\max\{cx|x \in P, x \in \mathbb{R}^n\}$.

A **feasible solution** is a vector $x \in \mathbb{R}^n$ satisfying all the inequalities $\{Ax \leq b\}$. The polyhedron P is the set of all feasible solutions, also called *domain*, and if $P = \emptyset$ the problem is *infeasible*. Extreme points of the polyhedron P are called **basic (feasible) solutions** and at least one of them maximizes the objective function $f(x) = cx$, *i.e.*, is an **optimal solution**. The optimal solution is denoted by x^* and the optimal value is $z^* = cx^*$. A basic solution contains m **basic variables** and $n - m$ non-basic variables whose values are set to zero. If the problem contains feasible solutions but no optimal ones, it is *unbounded*.

Relaxations. The **relaxation** of a problem is another problem *embedding* the original one: the set of the feasible solutions is a subset of the relaxed problem's domain and the relaxed objective function always gives better (or equal) values. Formally, a relaxation of the linear program $\max\{cx|x \in P, x \in \mathbb{R}^n\}$ is another linear program $\max\{c_R x|x \in P_R, x \in \mathbb{R}^n\}$ such that $P \subseteq P_R$ and $c_R x \geq cx, \forall x \in P$. An optimal solution of the relaxed problem is an upper bound for the original maximization problem.

Lagrangian relaxation. The **Lagrangian relaxation** of a linear program penalizes the violation of some constraints instead of forcing solutions to satisfy them. The domain is split into two sets of constraints and the linear program is stated as $\max\{cx|A_1x_1 \leq b_1, A_2x_2 \leq b_2, x_1 \in \mathbb{R}^{n_1}, x_2 \in \mathbb{R}^{n_2}\}$, where $A_1 \in \mathbb{R}^{n_1 \times m_1}$, $A_2 \in \mathbb{R}^{n_2 \times m_2}$, $b_1 \in \mathbb{R}^{m_1}$, $b_2 \in \mathbb{R}^{m_2}$, $n_1 + n_2 = n$ and $m_1 + m_2 = m$. The Lagrangian relaxation is obtained by moving some constraints in the objective function with some non-negative multipliers $\lambda \in \mathbb{R}^{+m_2}$: $\max\{cx + \lambda(b_2 - A_2x)|A_1x \leq b_1, x \in \mathbb{R}^n\}$.

When constraints of the second set are violated, the objective function is penalized, while it is increased when the constraints are strictly satisfied. This Lagrangian relaxation is usually constructed to be easily computed in order to quickly get an upper bound on the problem.

1.3.2 Integer programming

An **integer linear program**, or simply integer program (**IP**), is a linear program with the additional constraint that all variables must be integer: $\max\{cx|x \in P, x \in \mathbb{Z}^n\}$, where P is a polyhedron. A **mixed-integer linear program** (**MILP**) is a linear program where some of the variables must be integer. The **linear relaxation** of an integer program is the linear program relaxing the integrality constraints $z^*_{LP} = \max\{cx|x \in P, x \in \mathbb{R}^n\}$.

Formulations. Let $X \subseteq \mathbb{Z}^n$ be a set of feasible solutions of a combinatorial problem and $\mathcal{F}$ be an integer program $\mathcal{F} = \max\{cx|x \in Q, x \in \mathbb{Z}^{n+p}\}$, where $p \geq 0$. The projection of Q in the space of X is denoted by $P \subseteq \mathbb{R}^n$. $\mathcal{F}$ is a

formulation for X if $P \cap \mathbb{Z}^n = X$. By abusing notations, we also say that P is a formulation for X. It is an **extended formulation** if the variables are in a higher dimension than X, *i.e.*, $p > 0$. Many formulations exist for a same problem, with different qualities. A formulation P' is **tighter** than another P if $P' \subset P$. A formulation P is **ideal** for X if all the extreme points of P are integer, *i.e.*, $P = conv(X)$. A *valid inequality*, or cut, is an inequality $\pi x \leq \pi_0$ satisfied by every point of X.

In Figure 1.1, P, P' and P^* are formulations for the solution space X (represented by filled circles). P^* is ideal and tighter than P', which is tighter than P.

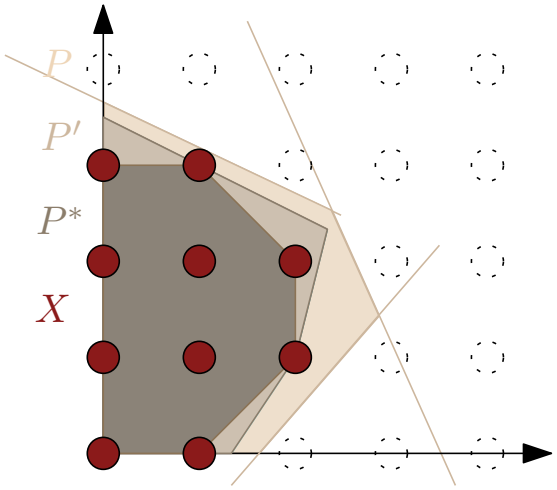


Figure 1.1 – Formulations for the solution space X

1.4 Solving linear programming problems

Linear programming problems are solvable in polynomial time by the *interior points* and *ellipsoid* methods, but usually solved by the simplex algorithm which shows very good results in practice, even though it may take an exponential time for some special instances. On the contrary, integer linear programs are NP-hard and solved by branch-and-bound. This algorithm requires to solve linear relaxations of the problem in an iterative scheme.

1.4.1 Simplex and branch-and-bound

The **simplex algorithm** is based on the fact that at least one optimal solution of a bounded linear programming problem is a basic solution (an extreme point of the corresponding polyhedron). The simplex method moves from a basic solution to another adjacent basic solution which improves the objective function, until it reaches an optimal solution. The movement

between a basic solution to another is called a ***pivot*** and implies that a basic variable is freed and "replaced" by a "better" non-basic variable which enters the basis.

Branch-and-bound is a divide-and-conquer method that can be applied in many optimization contexts. Any optimization problem aims at maximizing (or minimizing) an objective function over a set of possible solutions: $\max\{cx|x \in X\}$. The idea of the branch-and-bound is to divide the solution space X into smaller spaces (branching) and then evaluate the bounds on the objective function for each subdivision (bounding). This branching procedure is repeated for each smaller space, creating an enumeration tree. A node of the enumeration tree represents a subproblem and is pruned if it is proven to be suboptimal. When the optimal solution of a subproblem is found, the node is called a leaf and is not divided more. The branch-and-bound algorithm stops when no node remains to be explored (all of them are leaves or have been pruned), and the optimal solution of the original problem is the best of all optimal solutions found in the leaves.

A **branching rule** divides the solution space X_0 associated to a node into subsets $X_1, ..., X_k$ that cover the original set X_0: $\forall i \neq j \in \{0, ..., n\}$, $X_i \neq X_j$ and $X_0 = \bigcup_{i=1}^{k} X_i$. Many branching rules can be adopted to construct the enumeration tree of an integer program. An *efficient* branching rule should partition (and not only cover) the solution space: $X_i \cap X_j = \emptyset \ \forall i \neq j \in \{1, ..., n\}$. The number of subsets should also be small enough ($k = O(n)$) to avoid a combinatorial explosion.

The node i of the tree aims at solving $\max\{cx|x \in X_i\}$. Actually, it computes an upper bound UB_i and a lower bound LB_i. Let LB^* be the highest lower bound obtained overall. The branch is pruned if the subproblem cannot provide a better solution than the current one ($UB_i < LB^*$) or if it is unfeasible. The node becomes a leaf if the optimal on X_i is found ($UB_i = LB_i^*$).

The standard branch-and-bound solving IP uses a branching on fractional solutions and the linear relaxation to the bounding procedure. At each node the linear relaxation of the subproblem is solved and lower bounds are computed with heuristics. If the optimal solution $\widehat{x}$ of the linear relaxation is integer the node becomes a leaf, otherwise there exists a variable x_i with a fractional value $\widehat{x_i}$. The branching rule is to create two subproblems: one with the restriction $x_i \leq \lfloor \widehat{x_i} \rfloor$ and the other with the restriction $x_i \geq \lceil \widehat{x_i} \rceil$.

1.4.2 Large-scale problems

The efficiency of the branch-and-bound method depends on the quality of the linear relaxation bound computed at each node. Thus, among the many integer programming formulations existing for an optimization problem, the

choice of the tightest one is preferable. Unfortunately, this formulation can have an exponential number of constraints and/or variables in the input of the problem, leading to so-called **large-scale** formulations (in opposition to **compact** formulations). Of course, the linear relaxation is still polynomially solvable in the size of the program, but the linear program itself is exponential in the problem input size. The complete polyhedron may be impossible to know explicitly which makes the standard methods impractical to solve the linear program. When the formulation has an exponential number of constraints a cutting plane method is applied, while the column generation approach is used for programs with an exponential number of variables. These algorithms lie on the fact that an optimal solution can be found even if the polyhedron is described only locally around it and not completely.

The generalized **cutting plane method** can be used to solve any linear program (it can even extend to convex optimization), in particular when the number of constraints is too large to be explicitly set in the model. The technique is iterative: it starts with a relaxation of the problem containing a (small) subset of the numerous constraints and solves this relaxed linear program. A ***separation problem*** checks if the obtained solution is feasible for the original problem. If it is not, a separation oracle can find a ***violated cut***, *i.e.*, a constraint which is not satisfied by the solution. This violated cut is added to the problem and the process is repeated until an optimal solution of the original problem is found. If the separation problem is solvable in polynomial time, then the linear program is solvable in polynomial time. Initially, the cutting plane method was designed by Gomory [57] to solve integer linear programs by iteratively adding cuts to the linear relaxation of the problem until the optimal solution is integer. The cutting plane method can also be embedded in a branch-and-bound algorithm, adding valid inequalities to the linear relaxation in order to strengthen the quality of the bound. This approach is known as the *branch-and-cut*.

The **column generation** approach, depicted in figure 1.2, is based on the fundamental property that there exists an optimal solution which is a basic solution. In such a solution, at most m variables are non-zero (where m is the number of constraints). Consequently, only these basic variables are required to find an optimal solution to the problem. Thus, the linear relaxation can be solved starting from a subset of variables – this problem is called the **restricted master problem** (**RMP**) – and this subset grows as new variables (*i.e.*, new columns) are generated, until it is proven that no more variable is needed to find an optimal solution. Generating a new variable aims at improving the objective value of the current basis in the simplex algorithm. Finding such a column is called the **pricing problem** or *slave problem*. It is an optimization problem where the objective function

is the **reduced cost** of the variable, *i.e.*, the best improvement that could be done on the master problem objective function if this variable enters the basic solution. This reduced cost is computed using the solution value and the dual values of the current RMP. The constraints of the pricing problem emerge from the definition of a variable.

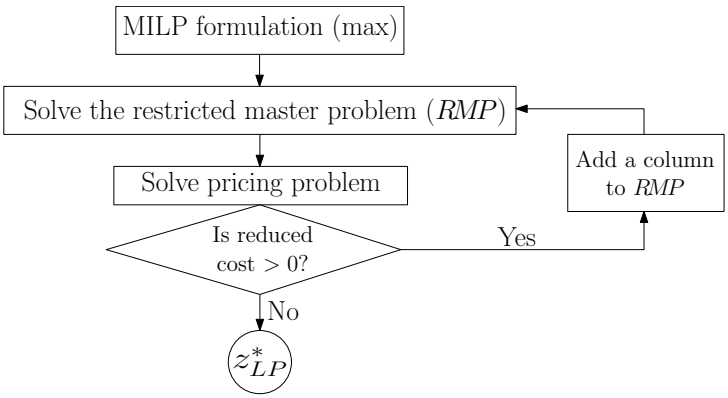


Figure 1.2 – Framework of the column generation

The goal of the pricing problem is to find the variable that maximizes the reduced cost. If the optimal reduced cost is strictly positive, then the variable is added to the set of variables and the process is repeated. Otherwise, every missing variable would only lower the master problem solution, so the optimal solution of the linear relaxation is found.

The column generation is principally applied within a branch-and-bound in order to solve integer programs with a number of variables exponential in the problem input. Such a technique is called *branch-and-price* and uses the column generation to solve the linear relaxation at each node of the branch-and-bound.

In the column generation framework, the number of constraints in the restricted master problem is fixed and all dual values are exactly known. So when both the number of variables and the number of constraints are large, the situation gets worse. Sometimes, the structure of the integer program allows to apply a *branch-and-cut-and-price*, which sequentially adds cuts and columns. This is however possible only when the separation and pricing problems are independent. On the contrary, if constraints are linked to variables, then missing columns imply that their corresponding constraints are also missing. The dual values of these constraints are unknown and the reduced costs calculation incorrect.

In this case, if rows and columns are generated separately, the final solution might be suboptimal or even unfeasible. Indeed, because of the wrong reduced costs, the column generation might miss generating required columns. Moreover, some absent columns can be linked to constraints violated by a solution. Problems with this kind of structure are referred to as

CDR-problems (for problems with **column-dependent-rows**). To guarantee the correctness of the solution of a CDR-problem, columns and rows must be generated simultaneously [91]. In his PhD thesis, Muter developed a general algorithm, called **simultaneous column-and-row generation** [91] and extended by Maher [81], which requires to solve several pricing problems. A big difficulty is to define these subproblems for the considered master problem. Indeed, they must estimate correctly the reduced cost of a variable to generate the required row and columns. For this reason, the design of these pricing problems is a crucial work that might be necessary for each column-dependent-row formulation. More recently, Sadykov and Vanderbeck studied a column-and-row generation for extended formulations [113].

Chapter 2

Kidney exchange programs: literature review and model

Kidney exchange programs and the associated problem studied in this thesis are described in this chapter.

2.1 Context

What is a kidney exchange program? Why is it a matter of interest? How the literature address this subject? We introduce this thesis by answering these questions.

2.1.1 Kidney disease and transplantation

The chronic kidney disease (CKD) causes the gradual loss of the kidney function, until its eventual failure. In the world, around one in ten people suffers from CKD, which was identified as the eleventh most common death cause in 2017 [62]. The prevalence of CKD is growing in proportion and the number of deaths resulting from the disease almost doubled since 1990 [62]. There is no cure for it, but some treatment can slow the evolution of the disease by controlling its causes. Still, every patients will face the final stage of the disease, referred to as the end-stage kidney disease (ESKD), and will need a kidney replacement therapy. Two options are available: a dialysis therapy or a kidney transplant, but the latter is preferable as it is more efficient and has less impact on the patients quality of life [129]. However, the shortage of donors makes this treatment rare and unable to save the numerous patients. In 2017, 90 306 renal transplantations were conducted worldwide while 1.2 million people died of ESKD [62, 120].

In a transplant, the kidney usually comes from a deceased donor and patients must register to the *waiting list* to get one. Patients might wait

several years and many cannot reach the moment when a compatible kidney becomes available for them [59]. This compatibility is based on ABO and HLA compatibilities. ABO compatibilities depend on the blood types (A, B, O and AB) and are demonstrated in Figure 2.1. Being HLA compatible means that the recipient blood does not contain antibodies to the Human Leukocyte Antigens of the donor. A positive crossmatch indicates the presence of such antibodies, which would cause the immune system to attack the transplant[1].

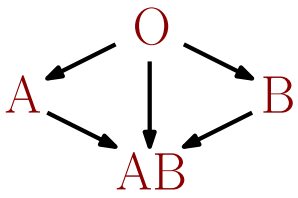


Figure 2.1 – ABO compatibilities. A transplant is ABO compatible if there is an arrow from the donor's blood type to the patient's blood type. Two people sharing the same blood type are also compatible.

Instead of waiting endlessly, a patient can be transplanted from a living donor, considering that a human body has two kidneys but usually needs only one to function. In fact, living donor transplantations were the first renal transplantations to be operated in the early 1950s and they are more successful than deceased ones [93]. Historically, the donor and the recipient needed to be ABO and HLA compatible, but recent research and progresses in immunosuppressive strategies enable incompatible transplants from living donors to be performed. These incompatible transplants are however more costly and more risky than compatible ones [90].

2.1.2 Exchanging kidneys

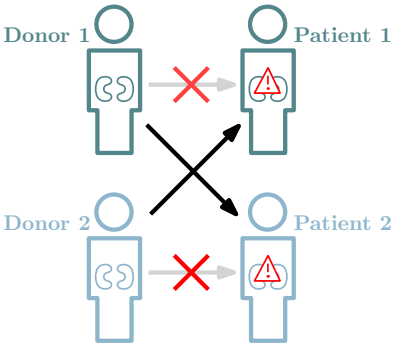
The medical evidences mentioned above lead to the conclusion that the **best type of renal transplantation is from a compatible living donor**. Originally, living donations were authorized only if the patient found itself a donor, and he was transplanted from this specified donor. Moreover, in a lot of countries the donor must be a close relative of the patient. In this context, it is really hard for a patient to find a willing and compatible donor in its entourage. So when a patient finds a willing donor—we say they form a **patient-donor pair**—they can decide to participate in a **kidney exchange program**, or Kidney Paired Donation (**KPD**). In such a program, a patient can swap his donor with another pair: they make an **exchange**. In an exchange, each recipient receives the kidney of another patient's donor,

[1] more info on HLA in kidney transplant can be found at https://web.stanford.edu/dept/HPS/transplant/html/hla.html

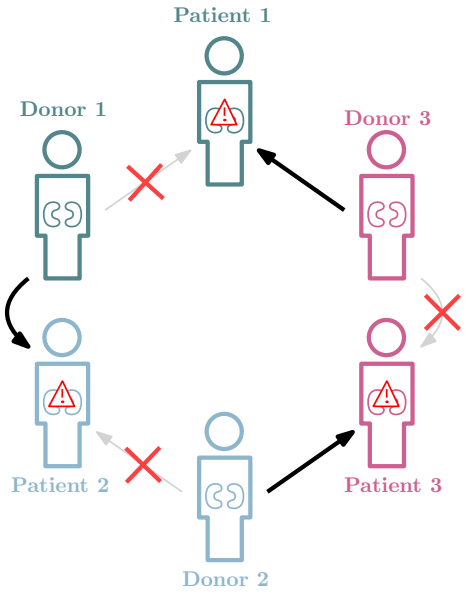
and each donor gives one kidney to another patient. In the initial KPDs, only **cycles of donation** were allowed (see Figure 2.2a and 2.2b). Nowadays, some programs also include **chains of donation** in which a first altruistic donor initiates a *domino donation*. The donor paired with the last patient in the chain either donates a kidney to the waiting list or becomes a *bridge donor*, considered later as an altruistic donor (see Figure 2.2c). Including chains of donation was not an easy decision and many studies were published to evaluate the impact of such a choice [9, 8, 11, 38, 52, 53, 87, 111]. Actually, the very idea of kidney exchanges led to several ethical discussions in the medical field [72, 79, 107, 108, 114]. In particular, the question of organ commerce, which is strictly forbidden in every country, was raised by Menikoff [85, 86]. Whether to keep anonymity or not is another subject still debated [73]. The discussion was however mainly to determine if these programs would be efficient and fair for every patients, some of them being disadvantaged by some allocation strategies (see Section 2.1.4).

A kidney exchange program usually works with match runs every few weeks or months. At each run, an optimization algorithm determines a set of exchanges that should be performed. This algorithm is designed according to the legislative and medical framework of the KPD, which can vary a lot depending on the countries, but three basic constraints are considered in every cases.

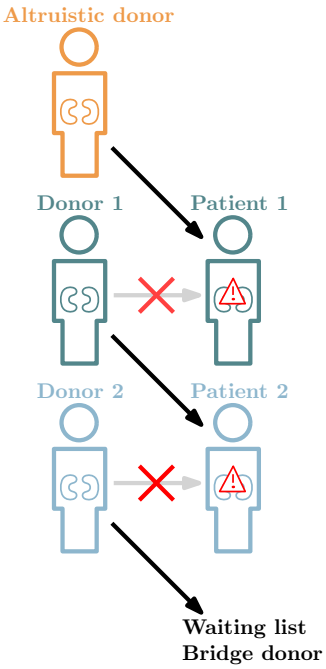
First, and for obvious medical reasons, a donor can donate only one kidney and a patient is candidate to a single transplantation. Thus, each patient-donor pair and each altruistic donor can participate in at most one exchange and we refer to this constraint as the physiological constraint. Secondly, by the very definition of a patient-donor pair, a donor agrees to give its kidney only if its paired patient receives one. This leads to the participation constraint which actually states that the whole pair must be included in an exchange. Finally, it is important to note that last-minute failures can break a possible transplant. Whether it be caused by medical complications or a donor withdrawal, a single breakdown in an exchange cause the whole exchange to fail. Due to this failure risk, it is not an option for a donor to donate a kidney before its paired patient receives one. This means that all the transplants of a cycle of donation must be performed simultaneously. As a cycle involving k pairs requires $2k$ surgeon teams and rooms, this simultaneity condition introduces a length constraint on cycles to avoid logistical difficulties. In the literature, the maximum number of pairs to involve in a cycle is often equal to 3 or 4, even if we can find KPDs performing long cycles, as in the Czech Republic where the successful removing of the simultaneity constraint led to a cycle of length 7 [17]. The failure pressure on chains of donation is far less important, as a patient will receive a kidney before its paired donor gives one. Some programs therefore conduct very long chains, as in the USA where a chain with 68 participants was recorded in 2014 [124]. However, the cancellation of a



(a) Cycle of donation with 2 pairs



(b) Cycle of donation with 3 pairs



(c) Chain of donation

Figure 2.2 – Standard exchanges in a kidney exchange program. Chains and cycles may include more pairs.

transplant still means that the whole remaining patients of the chains will not be transplanted and some programs recommend a length constraint on chains too, in order to limit the number of affected pairs. It seems that no consensus emerge on this topic: some countries only allow small chains (3 or 4), others do not limit their size and still others programs do not allow chains at all. To include the most cases, we consider in our work a limit on the number of patient-donor pairs, but it can be big. In the following, the length constraints denote the fact that lengths of chains and cycles of donation are limited to L and K respectively.

Besides the length constraints, other parameters of a KPD depend on the country: the frequency of match runs, the relationship between KPD and the waiting list, the inclusion of compatible pairs, the inclusion of incompatible pairs agreeing for an incompatible transplantation, the approval for incompatible transplant in exchanges and the possibility for a patient to have multiple donors [20]. These variants however do not affect our model as we will discuss in Section 2.2.1. Similarly, Biro *et al.* [20] identified 19 optimization criteria used by European countries and their management is also explained in Section 2.2.1.

2.1.3 Past and future of kidney exchange programs

The idea of kidney exchange was first mentioned by Rapaport in 1986 [105] and quickly set up in South Korea in 1991 [74, 98]. This idea was promising for this country where the public opinion is hostile on deceased transplantation. The Switzerland was the first European country to perform a kidney exchange in 1999, but the first national kidney exchange program in Europe was created by the Netherlands in 2004 [34]. Since then, a dozen states in Europe have created their own KPD [17]. In the rest of the world, and to the best of our knowledge, such programs exist only in Canada, Australia and the USA. We refer the reader to the survey of Ellison [42] for more details on the development of KPDs.

It is worth to note that several programs exist in the USA and that some of them are very local as they arrange exchanges within a single hospital [42]. On the other hand, European countries try to gather their programs to construct bigger pools of patient-donor pairs. These international programs aim at increasing the chances for a patient to find a match, but involve more complex organizations. A research group called European Network for Collaboration on Kidney Exchange Programs (ENCKEP) gathers policy makers, clinicians, economists, social scientists and optimization experts in order to establish a picture of practices and opportunities concerning this topic in Europe [32]. Transnational exchanges are already practiced in several countries and such collaborations are increasing. The first level of cooperation authorizes foreigners to participate into a national kidney exchange program. Portugal, Italy and Spain are more committed together

as each country includes the patient-donor pairs of the other countries remaining after a match run. Finally, the KPD Scandiatransplant merges the pools of Denmark, Finland, Iceland, Norway, Sweden and Estonia. Since 2016, Austria and the Czech Republic have been sharing their database as well. Cross-border exchanges are increasingly performed making the idea of a European KPD plausible in the next few years. However, the different legislations and variants of kidney exchange programs stand in the way of this project. Another major obstacle is the number of patient-donor pairs to be considered in such a program. The bigger the pool, the higher the chance to match patients, but the harder the underlying optimization problem. Indeed, it is a hard problem that standard optimization algorithms fail to solve when it contains several thousands of pairs. Nowadays, the largest program in Europe involves 250 patients [18], but considering that more than half a million Europeans [71] and even more USA citizens [115, 116] are treated for end stage kidney disease, new efficient techniques must be developed to handle many more candidates for kidney exchange programs.

The development of transnational KPDs should be quick and probably will be as the chronic kidney disease is brought to the forefront by international health organizations [94]. New optimization algorithms must be developed in a short term in order to follow this evolution. In a longer-term vision, we can expect that 3D printed technologies will be able to get rid of the need of donors. Kidneys are already 3D printed but they are simple replicas which can not replace a real kidney, due to the complexity of this organ. Actual artificial kidney transplants will not be possible before years or even decades, making kidney exchange programs invaluable [128].

2.1.4 Mathematical topics in kidney exchange programs

A kidney exchange program is actually a barter market: agents (patients) try to swap their item (donors) with other agents and new items (altruistic donors) can enter the market. The program must solve a clearing problem deciding the exchanges to conduct. The very first KPDs used to select "by hand" these exchanges but the growing number of participants required decision support tools. The first mathematical model of a KPD was proposed in 2003 by Roth *et al.* [111]. In this model, the matches between donors and patient were selected by an allocation algorithm using several criteria in a hierarchical scheme. The same approach was used in various studies and actually applied by most countries in the early years of KPDs [21, 34, 46, 58, 64, 65, 66, 83, 112]. The major drawback of these allocation mechanisms is that they usually enumerate and compare every possible allocations, an operation that becomes inefficient or even impossible for big databases. Optimization algorithms are therefore needed to find the best set of exchanges. Some of them keep the hierarchy of criteria by using lexicographic optimization [40, 55, 56, 83], but the majority gathers the

multiple criteria in a single objective function [20, 50, 82]. In this case each transplant is associated with a weight, or benefit, computed regarding medical, logistical and ethical parameters (ages of participants, blood types,...). The weight of a complete exchange sums the benefit of each contained transplant, sometimes adjusted with parameters of the exchange (*e.g.*, its length). The optimization problem seeking for the set of exchanges maximizing the total benefit is called the **kidney exchange problem** (**KEP**) and is the core of this PhD thesis.

Mathematicians also study the fairness of kidney exchange programs, and more specifically the trade-off between fairness and efficiency. In particular, how to construct the measure of a transplant benefit is a major work. The very definition of fairness is also a crucial topic discussed together with physicians. One of the first fears when the idea of KPD emerged was to disadvantage patients with blood type O, who are already suffering the longest waiting time due to ABO compatibilities (see Figure 2.1) [51, 80, 109, 127, 131]. The same risk arose for highly-sensitized patients, *i.e.*, patients with many kinds of antibodies thus having a positive crossmatch with most donors. These *hard-to-match* patients would be marginalized without adaptation of KPDs as explained by Dickerson *et al.* [40], based on the price of fairness defined by Bertsimas *et al.* and Caragiannis *et al.* [16, 25].

In the field of kidney exchange programs, some works study game theory involved in this market. Ashlagi and Roth studied the rationality for a hospital to participate in a national program without hiding information, in particular when it could transplant them itself [10]. They designed incentive mechanism that was afterward improved to limit the variance of agents' utility by Esfandiari and Kortsarz [43]. Liu *et al.* provide a study on the stability of matchings in kidney exchange programs [78].

2.2 The kidney exchange problem

The kidney exchange problem as defined in the previous section aims at finding the best set of exchanges to conduct in a KPD and can be modeled using standard tools of graph theory. In the following, we consider a kidney exchange program with n participants (patient-donor pairs and altruistic donors) for whom *a priori* compatibilities are known. We also assume that for each transplant a certain level of "desirability" is provided, possibly aggregating several medical parameters from both the donor and the recipient, and that the objective of the problem is to maximize the total benefit of the chosen transplants. Note that a special case of this problem with unitary weight in fact maximizes the **number** of transplants. In general maximizing the weight of exchanges can be conflicting with maximizing the number of transplants. Exchanges include cycles of donation of length at most K and

chains of donation containing at most $L-1$ patient-donor pairs (hence L agents).

2.2.1 Graph models

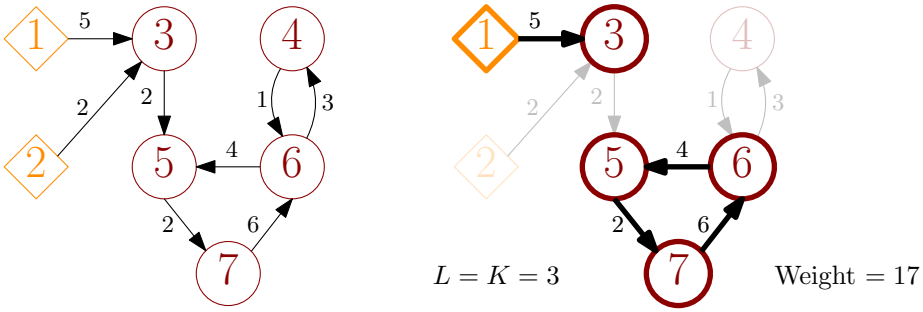
Compatibility graph. We model a kidney exchange program as a directed graph by creating one vertex for each participant and one arc for each possible transplant. Formally, the set P contains one vertex for each patient-donor pairs and the set N one vertex for each altruistic donor. To construct the **compatibility graph** $D = (V = P \cup N, A)$, we add an arc a_{uv} between $u \in V$ and $v \in P$ if the kidney of donor u can be transplanted to patient v. A weight function $w : A \to \mathbb{R}^+$ represents the medical benefit of each possible transplant. Note that determining the weight function is an upstream work and that w is an input in our case. This graph is generally quite sparse as it is rare for a patient and a donor to be compatible. Figure 2.3a shows an example of compatibility graph and differentiates altruistic donors (orange diamonds) from pairs (red circles).

Exchanges. An **exchange** is a subgraph of D which represents either a cycle of donation between pairs or a domino chain initiated by altruistic donors. In the compatibility graph, exchanges are elementary cycles of length at most K, called *valid cycles* and elementary paths starting by a vertex of N and having at most L vertices, *valid paths*. A valid cycle could have several symmetrical representations but they are eliminated by restricting the first vertex of the cycle vector to have the lowest identifier. Thus, a valid cycle c is represented by a unique vector $(v_1, ..., v_{|c|})$ such that $v_1 < v_j \ \forall j \in \{2, ..., |c|\}$.

$\mathcal{C}$ is the set of all valid cycles, $\mathcal{P}$ the set of all valid paths and $\mathcal{E} = \mathcal{C} \cup \mathcal{P}$ the set of all possible exchanges. We refer to the set of vertices (resp. edges) of an exchange e as $V(e)$ (resp. $A(e)$). The weight of an exchange $e \in \mathcal{E}$ is $w(e) := \sum_{a \in A(e)} w_a$. In Figure 2.3a for example, by taking $K = 3$ and $L = 4$, there exists 8 exchanges: two cycles ($e_1 = 5-7-6$; $e_2 = 4-6$) and six paths ($e_3 = 1-3$; $e_4 = 1-3-5$; $e_5 = 1-3-5-7$; $e_6 = 2-3$; $e_7 = 2-3-5$; $e_8 = 2-3-5-7$).

The kidney exchange problem. Two models of the KEP as defined above are equivalent. In the first one, the decision is made for each individual transplant: $\max\{\sum w_a : a \in A$ such that the length constraints, physiological constraint and participation constraint are respected$\}$. In the second vision, exchanges are constructed with respect to the length constraints and participation constraint and chosen such that the physiological constraint is respected: $\max\{\sum w_e : e \in \mathcal{E}$ such that chosen exchanges are disjoint$\}$. This idea to deal with exchanges rather than vertices or arcs gave us the idea to

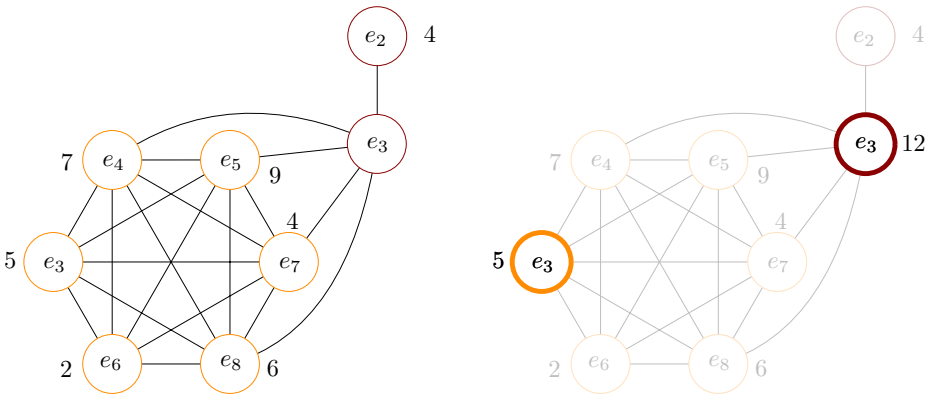
construct the intersection graph of exchanges. Figure 2.3 shows the optimal solution of the KEP in our example.



(a) Model (b) Solution of the KEP

Figure 2.3 – Example of a compatibility graph of a kidney exchange program

Intersection graph. We define $\mathcal{I}(\mathcal{E})$ the *intersection graph of exchanges* of D, as the undirected graph $\mathcal{I}(\mathcal{E}) = (\mathcal{E}, E)$ where each vertex represents an exchange and an edge links two exchanges if they share at least one vertex in D: $E = \{e_1, e_2 \in \mathcal{E}^2 | e_1 \cap e_2 \neq \emptyset\}$. A weight function ω on the vertices of $\mathcal{I}(\mathcal{E})$ provides the medical benefit of each exchange. Note that the construction of this intersection graph is **exponential** for a given input of the kidney exchange problem. A feasible solution of the KEP is a stable set of $\mathcal{I}(\mathcal{E})$. The intersection graph of the exchanges previously described is presented in Figure 2.4a, and the solution enlightened in 2.4b. In these figures, paths are represented in orange and cycles in red.



(a) Model (b) Solution of the KEP

Figure 2.4 – Example of an intersection graph of a kidney exchange program

Property 2.1

If $L \geq K$, then the intersection graph is $K_{1,L+1}$-free.

Proof. Assume $\mathcal{I}(\mathcal{E})$ contains a $K_{1,L+1}$ as an induced subgraph, then an exchange e_0 intersects $L+1$ other exchanges $e_1, ...e_{L+1}$ that do no intersect each other. Each exchange $e_1, ...e_{L+1}$ thus intersects e_0 on a different vertex and e_0 contains at least $L+1$ vertices, which is contradictory with its definition. Note that if $K \geq L$ then $\mathcal{I}(\mathcal{E})$ is $K_{1,K+1}$-free. □

Dealing with problem variants. These models are easy to adapt for most of the different alternative KPDs, in particular via the weight function. Compatible pairs form new vertices in the compatibility graph, and the desirability of transplants with other patients may be more demanding. Allowing incompatible transplants within an exchange only add arcs with positive weight in the graph. Programs with multiple donors for a same patient construct vertices which do not represent a pair anymore, but a patient and all its specified donors. The weight of an arc represents the benefit of the best transplant among all the possible transplants.

Our KEP is a single objective optimization problem but it can accommodate multiple criteria. Most of the criteria existing in KPDs concern a transplant data (*e.g.*, patient age or waiting time) and can be taken into account in the weight function. Some of them however may implement yardsticks for complete exchanges (and not for single transplants). The weight function on exchanges can integrate this kind of criterion and the KEP seen at the exchange level won't change. On the contrary, when the KEP decisions are made on transplants, these criteria must be integrated as constraints and may be hard to dealt with.

2.2.2 Related problems

The kidney exchange problem corresponds to standard and well-known problems described below. Of course as it is NP-complete it is equivalent to every NP-complete problems, but three of them are more closely related.

Set packing problem. Given a set of elements S, the **set packing problem** aims at finding a packing of pairwise disjoint subsets among a family $\mathcal{S} \subseteq \mathcal{P}(S)$ of subsets of these elements. The kidney exchange problem can be modeled as a maximum-weight set packing problem in D. Elements of the problem are the arcs of D and the family of subsets is the exchange set $\mathcal{S} = \mathcal{E}$. These problems are equivalent as the subsets respect the length constraints and participation constraint by construction and the physiological constraint is satisfied with the disjunctive constraint of the set packing problem.

Stable set problem. Given any graph and a weight function on the vertex set, the maximum-weight **stable set problem** seeks a stable set, *i.e.*, a set of vertices pairwise non adjacent, of maximum weight in the graph. By construction, any stable set of $\mathcal{I}(\mathcal{E})$ is a set of exchanges that do not share any patient or donor. Thus, the kidney exchange problem is equivalent to a maximum-weight stable set problem in the intersection graph $\mathcal{I}(\mathcal{E})$.

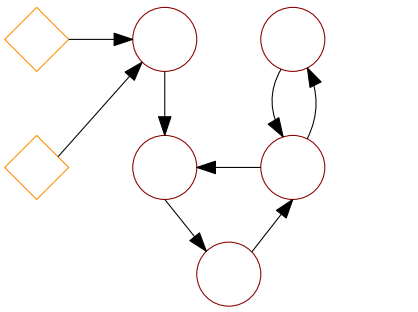
Vehicle routing problem. The goal of the **vehicle routing problem** is to find a set of routes to deliver a set of customers with a fleet of vehicles that minimized the traveled distance. In the standard version the vehicles start and end in a single depot, but the Multi Depot Vehicle Routing Problem (MDVRP) involves multiple depots and several vehicle types. This problem is widely studied and many efficient algorithms exist to solve it, hence our interest for it. By considering exchanges as route, both problems appear quite close. We propose a reduction from the KEP to the VRP, detailed below. This reduction leads to many depots, which means that algorithms dedicated to the VRP do not fit well to solve the KEP.

Formally, given a set of customers C, a set of depots Δ, a demand q_v for each customer $v \in C$, κ vehicle types, a capacity Q_k for each vehicle type k, M_{dk} the number of vehicles of type k at depot $d \in \Delta$, c_k the unit running cost of a vehicle of type k and $d_{uv} \geq 0$ the distance between $u \in C \cup \Delta$ and $v \in C \cup \Delta$, the MDVRP seeks for a set of routes minimizing the total cost. A route is a tour beginning and ending in the same depot and each route is made by a single vehicle. The cost of a route is the unit running cost multiplied by the traveled distance.

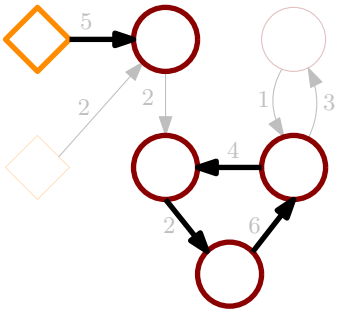
We show how to reduce an instance of KEP to an instance of MDVRP and represent this reduction on an example in Figure 2.5. We define the set of customers C as the set of patient-donor pairs P. The set Δ of depots is composed of Δ_N, Δ_A and Δ_P, where Δ_N contains a depot for each altruistic donor, Δ_A for each arc of $G[P]$ and Δ_P for each patient-donor pair. A unit demand is given to the customer set. Three types of vehicles are used, with a capacity of $L-1$, K and 1 respectively. Each depot of Δ_N, Δ_A and Δ_P contains exactly one vehicle of type 1, 2 and 3 respectively. The unit running cost is -1 and the distance function is defined below, with an associated "type".

$$d_{uv} = \begin{cases} w_{uv} \text{ if } (uv) \in A & \text{agent-agent} \\ \dfrac{w_{uv}}{2} \text{ if } u \in \Delta_A \text{ or } v \in \Delta_A & \text{pair-depot} \\ 0 \text{ if } u \in \Delta_P \text{ or } v \in \Delta_P & \text{self-depot} \\ 0 \text{ if } v \in \Delta_N & \text{agent-altruist} \\ +\infty \text{ otherwise} & \text{no-match} \end{cases}$$

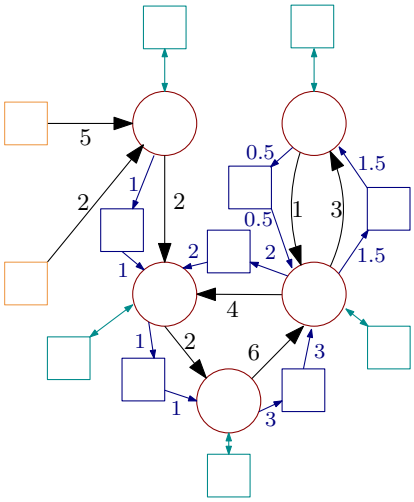
In Figure 2.5, red circles represent donor-pairs and clients, orange diamonds represent altruistic donors and squares represent depots (Δ_N Δ_A



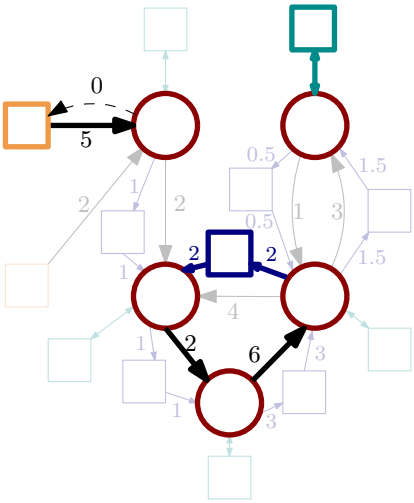
(a) KEP instance



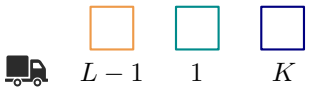
(b) A solution for the KEP



(c) The equivalent MDVRP instance



(d) The equivalent MDVRP solution



(e) Capacity of trucks depending on the depot type

Figure 2.5 – Reduction from KEP to MDVRP on an example

Δ_P). Distances are written above "agent-agent" (brown) and "agent-depot" (blue) edges. Turquoise edges are the "self-depot" edges and have a distance of 0. "No-match" and "agent-altruist" edges are not represented. In the MDVRP instance, numbers are the distance and the unit running cost, not written, equals -1 on each arc.

A solution of the MDVRP visits every patient-donor pair of the program (see Figure 2.5d). Paths of the KEP are equivalent to routes starting from orange depots representing altruistic donors. For cycles, several depots are possible, but only one is chosen as the actual depot of the route. Patient-donor pairs that are not included in the corresponding KEP solution are visited with a truck of capacity 1, leaving from the depot associated to this vertex.

2.2.3 Solving the kidney exchange problem

Different approaches exist to find the set of exchanges maximizing the total weight of transplants in a KPD. We already discussed in Section 2.1.4 the allocation algorithms used at the beginning of kidney exchange programs, but we are interested here in optimization algorithms handling the kidney exchange problem. When it contains only cycles of length 2, the KEP can be solved polynomially via Edmonds' algorithm, but as soon as $K > 2$, the problem is proved to be NP-complete [1, 19]. Consequently, the KEP is often tackled with integer programs and the major ones are surveyed by Mak-Hau [82] and detailed in Chapter 3.

In the first place, formulations used to ignore chains of donation. Roth *et al.* introduced the edge formulation and the cycle formulation in 2007 [110]. Abraham *et al.* proved that the cycle formulation is better than the edge formulation, with respect to the tightness of the linear relaxation [1]. The cycle formulation however requires to compute every possible cycles which is too long in most of the cases. Abraham *et al.* thus developed a column generation approach to solve this integer program [1], which is still nowadays the best way to solve the KEP without altruistic chains.

The cycle formulation can be equivalently applied when including chains of donation, but we will call it the exchange formulation. In Chen *et al.*, every exchanges is computed beforehand [27], but this is not a viable method when the patients pool grows. On the basis of Abraham *et al.* work, branch-and-price algorithms were developed [55, 56, 67, 103] claiming to accommodate well altruistic donors via chains of donation. However some of these algorithms (in [55, 56, 103]) were proven wrong by Plaut *et al.* [104] and Klimentova *et al.* did not test their algorithm with altruistic donors [67]. Actually, Plaut *et al.* proved in 2016 that the pricing algorithm becomes NP-complete in this case. Anderson *et al.* [7] also proposed an integer program with exponentially many variables but did not use a column generation approach. Their formulation, based on the traveling salesman problem, has

an exponential number of constraints too, which are handled by a polynomial separation oracle. Other proposed integer programming formulations are compact [31, 37, 82].

Approximations are often proposed when it comes to NP-hard problems. Still, to the best of our knowledge, only Biro *et al.* [19] and Jia *et al.* [63] considered such algorithms for the standard KEP. Approximations are more common when the problem is not considered to be static and deterministic. Stochastic schemes take into account uncertainty and positive crossmatch failures. Indeed, when the algorithm runs, it is applied on incomplete information. Effective compatibility is tested precisely latter and agents may be pulled out the program. The approach of Manlove and O'Malley to handle this uncertainty is to prioritize small exchanges in a lexicographical multi-objective algorithm [83]. The mainly studied method is however to consider a probability of failure on each transplant [5, 39, 68, 77, 101, 132]. An alternative idea is to consider a KPD as a dynamic, or online, system [6, 12, 123].

2.3 Setting of the thesis

In the variety of parameters, problems and algorithms in kidney exchange programs detailed in this chapter, choices needed to be made for this thesis. Our decisions arise from conclusions drawn from the literature review and might of course be discussed, but we believe they are relevant in the current context of the kidney exchange field. The analysis of kidney exchanges evolution reveals two major issues: the growing size of program pools and the inclusion of altruistic donors via domino chains. Dealing with these two parameters in the KEP is hard enough to elude the system stochasticity and dynamism in a first time. Besides, the arrival of patients or donors may be slow enough to maintain periodical match runs or, should it be necessary, to increase their frequency.

The consequence of having big pools is the need to solve a large-scale problem. We expect the natural exponential formulation (the exchange formulation) to be the most likely to achieve this task in a reasonable amount of time. Indeed, compact formulations are not advantageous compared with exponential ones as they usually do not scale up to large instances[2] and provide poor upper bounds [82]. Moreover, among the several large-scale formulations, including the new one we propose in Section 3.2.2, the exchange formulation is the simplest, an attractive feature to collaborate with other specialists (physicians and lawmakers), and is therefore the one chosen for our work. The inclusion of altruistic donors is rarely implemented

[2]It seems paradoxical that scaling up is a compact formulations weakness, but actually column generation approaches are really effective on exponential formulations. It is often simpler to use large-scale methods (see Section 1.4.2) on these exponential formulations than on compact ones.

in column generation works for this exchange formulation, often treated as a natural extension of existing models. Actually, since the proof by Plaut *et al.* [104] that the pricing problem is NP-hard in this situation, no complete column generation algorithm was developed. We propose to effectively handle these chains of donation, having in general a different length limit than cycles. This brings the subject to elementary paths problems, which is a major part of this thesis. More specifically, we consider the problem of finding elementary paths starting from a given source, having a limited number of vertices and maximizing the total weight of its arcs.

To sum it up, we study the static KEP with limited chains and cycles, using integer program formulations (see Chapter 3), more specially the exchange formulation solved with column generation (see Chapter 4). We also focus on the elementary minimum path problem with length constraint (see Chapter 5 and Chapter 6). This problem is extracted and studied independently from the KEP, even though this background, in particular the sparsity of the KEP instances, influences our research.

Chapter 3

Integer programming formulations for the kidney exchange problem

The purpose of a kidney exchange problem is to find the "best" set of transplants in a barter market of n agents: pairs of incompatible patient-donor and altruistic donors. In this program, only some transplants are possible: each donor (paired or altruistic) is compatible with a small number of patients. Moreover, each possible transplant has an estimated medical benefit, which can be computed by physicians. The "best" set of transplant is the one which maximizes the medical benefit. The solution must respect three basic constraints, whose origins are detailed in Chapter 2:

- physiological constraint: each donor and each patient can participate in at most one transplant
- participation constraint: if a donor from a pair participates in a transplant, then its associated patient must receive a kidney
- length constraints: at most L (resp. K) agents participate in any chain (resp. cycle) of donation

Several integer programming formulations were proposed to model the KEP and this chapter makes a survey on models that take into account altruistic donors. Recall from Section 2.2 that the KEP can be seen either as a packing problem in the compatibility graph D or as a stable set problem in the intersection graph $\mathcal{I}(\mathcal{E})$. Formulations of both problems are thereby presented in separate sections. In addition of existing models, a new extended formulation for the stable set problem having interesting properties in our particular case is studied in Section 3.2.2.

3.1 KEP as a packing

In the compatibility graph, the kidney exchange problem is a maximum-weighted cardinality-constrained cycles and paths packing problem (see Section 2.2.2). As explained in Section 2.2.1, this problem has two different modeling. The most natural one is, given the set of all possible exchanges $\mathcal{E}$, to find the set of exchanges $\mathcal{E}^*$ maximizing the total weight under the physiological constraint: each agent must be chosen in at most one exchange. Integer programs for this model are called **exchange-based formulations**.

3.1.1 Exchange-based formulations

In an exchange-based formulation each exchange is associated with one binary variable indicating if it is chosen or not in the solution. The length constraints and participation constraint are satisfied when the exchanges are constructed. This leads to simple models but having an exponential number of variables, which are usually treated with column generation methods (see Section 1.4.2).

The most studied formulation is the "cycle formulation" from Roth *et al.* and Abraham *et al.* [1, 110]. As we include also elementary paths, we call it the **exchange formulation** (**EF**). Recall that we consider a unique representation of each cycle in $\mathcal{E}$ to exclude symmetry, so our model is equivalent to the disaggregated cycle formulation of Klimentova *et al.* [67]. A unique set of constraints (3.2) is required to model the physiological constraint.

Model EF

$$\forall e \in \mathcal{E},\ x_e = \begin{cases} 1 \text{ if exchange } e \text{ is chosen} \\ 0 \text{ otherwise} \end{cases}$$

$$z^* = \max \sum_{e \in \mathcal{E}} w_e x_e \tag{3.1}$$

$$\sum_{\substack{e \in \mathcal{E}: \\ i \in V(e)}} x_e \leq 1 \qquad \forall i \in V \tag{3.2}$$

$$x_e \in \{0, 1\} \qquad \forall e \in \mathcal{E} \tag{3.3}$$

EF is a large-scale integer program as it contains a variable for each exchange, and the pricing problem turns out to be NP-hard due to the path exchanges as shown by Plaut *et al.* [104]. This was not properly taken into account by previous column generation schemes [56, 103] so we propose a new framework in Chapter 4.

Some other formulations are based on a more disaggregated view of the

problem: instead of choosing directly the exchanges, they determine one by one each transplant of the solution. The decision variables are indexed on the arcs and the effective exchanges to perform are deduced afterwards. Such formulations are said to be **arc-based**.

3.1.2 Arc-based formulations

Initially, arc-based formulations used a binary variable for each arc of D stating if the arc is chosen in a cycle. These variables are not sufficient anymore with the inclusion of paths initiated by altruistic donors. The integer program must still decide if an arc is chosen or not, but, in addition, it has to identify in which sort of exchange it is chosen. Indeed, constraints differ for cycles and paths, in particular length constraints. Therefore, the variables are split into two families of binary variables: u_{ij} for the cycles and y_{ij} for the paths.

Mak-Hau [82] presented two integer programs adding constraints from the traveling salesman problem (TSP) in existing formulations of the KEP. These new constraints had to permit valid paths to be constructed, in particular prevent the path variables to induce subcycles. Mak-Hau decided to add the polynomial-sized subtour elimination constraints coming from the compact Miller-Tucker-Zemlin (MTZ) formulation of the TSP [88]. These constraints require a continuous variable t_i representing the "time stamp" of vertex i in a path, $\forall i \in V$. They also imply that a variable y_{ij} is defined for every pair of vertices, not only arcs.

Recall that the compatibility graph $D = (V = P \cup N, A)$ is composed by two vertex sets: one for patient-donor pairs (P) and one for altruistic donors (N). An arc (ij) represents the possibility to transplant the kidney of donor i to patient j, so vertices of N have no incoming arcs.

MTZ arc formulation. The first arc-based formulation, the edge formulation, proposed in 2007 by Roth *et al.* [110], was split by Mak-Hau to integrate paths in addition to cycles. We call it the **MTZ arc formulation** (**MTZ-AF**). It applies strong cardinality-infeasible-cycle elimination using a set of minimal infeasible paths, defined as $\Pi := \{\pi \subseteq D : \pi \text{ is a path of length } K+1\}$.

Constraints (3.5) and (3.6) are flow constraints for vertices of P modeling the participation constraint. Constraints (3.7) and (3.8) express the physiological constraint. The length constraints are decomposed into elimination constraints (3.9) for cycles and bound constraints (3.10) for paths. There are exponentially many elimination constraints and they require to enumerate all the paths of length $K+1$ or to use a separation oracle. Note that the right-hand side of (3.9) is $K-1$ and not K. Indeed, there is no cycle of length K in π, $\forall \pi \in \Pi$. A valid cycle could involve K vertices of $V(\pi)$, but

Model MTZ-AF

$\forall i,j \in V^2$: $y_{ij} = \begin{cases} 1 \text{ if arc } (ij) \text{ is part of a path} \\ 0 \text{ otherwise} \end{cases}$

$\forall (ij) \in A$: $u_{ij} = \begin{cases} 1 \text{ if arc } (ij) \text{ is part of a cycle} \\ 0 \text{ otherwise} \end{cases}$

$\forall i \in V : t_i =$ time stamp of visiting i in a path

$$\max \sum_{(ij)\in A} w_{ij}y_{ij} + \sum_{(ij)\in A} w_{ij}u_{ij} \tag{3.4}$$

Participation

$$\sum_{j\in N^+(i)} u_{ij} - \sum_{j\in P\cap N^-(i)} u_{ji} = 0 \qquad \forall i \in P \tag{3.5}$$

$$\sum_{j\in N^+(i)} y_{ij} - \sum_{j\in N^-(i)} y_{ji} = 0 \qquad \forall i \in P \tag{3.6}$$

Physiological

$$\sum_{j\in N^+(i)} y_{ij} + \sum_{j\in P\cap N^+(i)} u_{ij} \leq 1 \qquad \forall i \in P \tag{3.7}$$

$$\sum_{j\in N^+(i)} y_{ij} \leq 1 \qquad \forall i \in N \tag{3.8}$$

Length

$$\sum_{(ij)\in A(\pi)} u_{ij} \leq K-1 \qquad \forall \pi \in \Pi \tag{3.9}$$

$$t_i \leq L-1 \qquad \forall i \in V \tag{3.10}$$

$$t_i - t_j + |P|y_{ji} + (|P|+2)\,y_{ij} \leq |P|+1 \qquad \forall i,j \in V^2 \tag{3.11}$$

$$t_i = 0 \qquad \forall i \in N \tag{3.12}$$

$$u_{ij} \in \{0,1\} \qquad \forall (ij) \in A \tag{3.13}$$

$$y_{ij} \in \{0,1\} \qquad \forall i,j \in V^2 \tag{3.14}$$

$$t_i \in \mathbb{R}^+ \qquad \forall i \in V \tag{3.15}$$

then at least one of its arcs would not belong to $A(\pi)$. Thus, at most $K-1$ arcs of π can be selected in a solution.

The MTZ constraints force a path of the solution to visit vertices in increasing order of their time stamps. Indeed, (3.11) state that if arc (ij) is in the solution, then $t_j = t_i + 1$. The time stamp of altruistic donors is set to zero by (3.12) as they are either first in a path or not chosen at all.

MTZ extended arc formulation. Mak-Hau also split the extended edge formulation of Constantino *et al.* [31] to construct the **MTZ extended arc formulation** (**MTZ-EAF**). The principle is to clone D into $|P|$ copies and to impose that each copy contains at most one cycle in the solution.

Thus, the u variables are also indexed by the copy of D in which the cycle appears. Cloning the graph enables an independent treatment for each cycle.

Model MTZ-EAF

$$\forall i,j \in V^2\text{: } y_{ij} = \begin{cases} 1 \text{ if arc } (ij) \text{ is part of a path} \\ 0 \text{ otherwise} \end{cases}$$

$$\forall (ij) \in A,\ \forall k \in [\![1;|P|]\!]\text{: } u_{ij}^k = \begin{cases} 1 \text{ if arc } (ij) \text{ is part of a cycle} \\ \qquad\qquad \text{in the } k^{th} \text{ copy of } D \\ 0 \text{ otherwise} \end{cases}$$

$$\forall i \in V : t_i = \text{time stamp of visiting } i \text{ in a path}$$

$$\max \sum_{(ij)\in A} w_{ij} y_{ij} + \sum_{k=1}^{|P|} \sum_{(ij)\in A} w_{ij} u_{ij}^k \tag{3.16}$$

Participation

$$\sum_{j\in N^+(i)} u_{ij}^k - \sum_{j\in P\cap N^-(i)} u_{ji}^k = 0 \qquad \forall i \in P, \forall k \in [\![1;|P|]\!] \tag{3.17}$$

$$\sum_{j\in N^+(i)} y_{ij} - \sum_{j\in N^-(i)} y_{ji} = 0 \qquad \forall i \in P \tag{3.18}$$

Physiological

$$\sum_{j\in N^+(i)} y_{ij} + \sum_{k=1}^{|P|} \sum_{j\in P\cap N^+(i)} u_{ij}^k \leq 1 \qquad \forall i \in P \tag{3.19}$$

$$\sum_{j\in N^+(i)} y_{ij} \leq 1 \qquad \forall i \in N \tag{3.20}$$

Length

$$\sum_{(ij)\in A} u_{ij}^k \leq K \qquad \forall k \in [\![1;|P|]\!] \tag{3.21}$$

$$t_i \leq L - 1 \qquad \forall i \in V \tag{3.22}$$

$$\sum_{j\in P\cap N^+(i)} u_{ij}^k - \sum_{j\in P\cap N^+(i)} u_{kj}^k \leq 0 \qquad \forall i \in P, \forall k \in [\![1;i-1]\!] \tag{3.23}$$

$$\sum_{j\in P\cap N^+(i)} u_{ij}^k = 0 \qquad \forall i \in P, \forall k \in [\![i+1;|P|]\!] \tag{3.24}$$

$$t_i - t_j + |P| y_{ji} + (|P|+2)\, y_{ij} \leq |P| + 1 \qquad \forall i,j \in V^2 \tag{3.25}$$

$$t_i = 0 \qquad \forall i \in N \tag{3.26}$$

$$u_{ij}^k \in \{0,1\} \qquad \forall (ij) \in A,\ \forall k \in [\![1;|P|]\!] \tag{3.27}$$

$$y_{ij} \in \{0,1\} \qquad \forall i,j \in V^2 \tag{3.28}$$

$$t_i \in \mathbb{R}^+ \qquad \forall i \in V \tag{3.29}$$

Constraints (3.17) and (3.18) are flow constraints for vertices of P mod-

eling the participation constraint. Constraints (3.19) and (3.20) express the physiological constraint. The length constraints are decomposed into bound constraints (3.21) for cycles and (3.22) for paths. Constraints (3.23) and (3.24) deal with the symmetry elimination by restricting the index k of a cycle to be the lowest index of its vertices. More precisely, the k^{th} cycle is either empty or contains an arc (kj) and some arcs (ij) with $i > k$. Constraints (3.24) impose that an arc (ij) cannot belong to a cycle with an identifier k greater than i, since i should have been the identifier of this cycle. Together with constraints (3.23), they impose that if an arc (kj) is not taken in the k^{th} cycle, then this cycle contains no arcs.

The MTZ constraints force a path of the solution to visit vertices in increasing order of their time stamps. Indeed, (3.25) state that if arc (ij) is in the solution, then $t_j = t_i + 1$. The time stamp of altruistic donors is set to zero by (3.26) as they are either first in a path or not chosen at all.

Position-indexed formulation HPIEF. Dickerson *et al.* proposed several position-indexed formulations [37], including the purely arc-based **HPIEF**. As the MTZ-EAF, it uses $|P|$ copies of D, but it also indexes variables with positions. Let $\mathcal{P}_{ij}^k$ be the set of positions at which arc (ij) can be selected in a cycle of the k^{th} copy of D, such that the "first" vertex of this cycle (in its unique representation) is k.

$$\forall (ij) \in A, \forall k \in [\![1, |P|]\!],\ \mathcal{P}_{ij}^k = \begin{cases} \{1\} & i = k \\ \{2, ..., K-1\} & i > k \text{ and } j > k \\ \{2, ..., K\} & j = k \end{cases} \tag{3.30}$$

Similarly, let $\mathcal{P}_{ij}$ be the set of positions at which arc (ij) can be chosen in a path of D. A vertex can be at the first position of a path if and only if it represents an altruistic donor.

$$\forall (ij) \in A,\ \mathcal{P}_{ij} = \begin{cases} \{1\} & i \in N \\ \{2, ..., L\} & i \in P \end{cases} \tag{3.31}$$

Constraints (3.33) and (3.34) are flow constraints for vertices of P modeling the participation constraint. Constraints (3.35) and (3.36) express the physiological constraint. The length constraints are integrated in the variable definition.

Model HPIEF

$$\forall (ij) \in A, p \in \mathcal{P}_{ij}\text{: } y_{ijp} = \begin{cases} 1 \text{ if arc } (ij) \text{ is selected at position } p \text{ in a path} \\ 0 \text{ otherwise} \end{cases}$$

$\forall (ij) \in A, \forall k \in [\![1; |P|]\!], p \in \mathcal{P}^k_{ij}$:

$$u^k_{ijp} = \begin{cases} 1 \text{ if arc } (ij) \text{ is selected at position } p \\ \qquad \text{in a cycle of the } k^{th} \text{ copy of } D \\ 0 \text{ otherwise} \end{cases}$$

$$\max \sum_{k=1}^{|P|} \sum_{(ij)\in A} \sum_{p \in \mathcal{P}^k_{ij}} w_{ij} u^k_{ijp} + \sum_{(ij)\in A} \sum_{p \in \mathcal{P}_{ij}} w_{ij} y_{ijp} \tag{3.32}$$

Participation

$$\sum_{\substack{j \in N^-(i): \\ p \in \mathcal{P}^k_{ji}}} u^k_{jip} - \sum_{\substack{j \in N^+(i): \\ p+1 \in \mathcal{P}^k_{ij}}} u^k_{ij(p+1)} = 0 \qquad \begin{matrix} \forall k \in [\![1; |P|]\!] \\ \forall i \in [\![k+1; |P|]\!] \\ \forall p \in [\![1; K-1]\!] \end{matrix} \tag{3.33}$$

$$\sum_{\substack{j \in N^-(i): \\ p \in \mathcal{P}_{ji}}} y_{jip} - \sum_{j \in N^+(i)} y_{ij(p+1)} \geq 0 \qquad \begin{matrix} \forall i \in P \\ \forall p \in [\![1; L-1]\!] \end{matrix} \tag{3.34}$$

Physiological

$$\sum_{k=1}^{|P|} \sum_{i \in N^-(j)} \sum_{p \in \mathcal{P}^k_{ij}} u^k_{ijp} + \sum_{i \in N^-(j)} \sum_{p \in \mathcal{P}_{ij}} y_{ijp} \leq 1 \qquad \forall j \in P \tag{3.35}$$

$$\sum_{j \in N^+(i)} y_{ij1} \leq 1 \qquad \forall i \in N \tag{3.36}$$

$$u^k_{ijp} \in \{0,1\} \qquad \begin{matrix} \forall (ij) \in A \\ \forall k \in [\![1; |P|]\!] \\ \forall p \in \mathcal{P}^k_{ij} \end{matrix} \tag{3.37}$$

$$y_{ijp} \in \{0,1\} \qquad \begin{matrix} \forall (ij) \in A \\ \forall p \in \mathcal{P}_{ij} \end{matrix} \tag{3.38}$$

In an arc-based formulation, constraints must manage the elimination of possible subtours to avoid non elementary paths or cycles. Moreover, two sets of constraints are required for the KEP, as cycles and paths must be treated separately. We found in the literature several ways to model these subtour elimination constraints, mainly adapted from the traveling salesman problem. They are summed up in Table 3.1a, with the number of variables and constraints in Table 3.1b.

Using copies of the graph avoids the strong cardinality-infeasible-cycle elimination of model MTZ-AF, which are exponentially many, but intro-

	Cycles		Paths
Formulation	Subtour	Symmetry	Subtour
MTZ-AF	Enumeration		MTZ (Time stamps)
MTZ-EAF	Copies	Constraints	MTZ (Time stamps)
HPIEF	Copies	Positions	Positions

(a) Subtour elimination and symmetry management

	Variables	Constraints
MTZ-AF	$O\left(\|V\|^2\right)$	$O\left(\|V\|^K\right)$
MTZ-EAF	$O\left(\|V\|^3\right)$	$O\left(\|V\|^2\right)$
HPIEF	$O\left(\|V\|^4\right)$	$O\left(K\|V\|^2\right)$

(b) Number of variables and constraints

Table 3.1 – Arc-based formulations review

duces numerous variables. The HPIEF formulation uses positions for cycles to achieve symmetry elimination, replacing the additional constraints (3.23) and (3.24) of model MTZ-EAF. To sum up, arc-based formulations require to use subtour elimination constraints which are either weak, as the MTZ suffering from big-M inequalities, or exponentially many. Some of them are compact for the KEP (MTZ-EAF and HPIEF), but they still use $|P|$ copies of the compatibility graph and at least $|A|$ variables for each of these copies. On top of that, they are also complex to understand and implement. An idea proposed in several papers is to mix the arc-based and exchange-based formulations, with the purpose to take advantage of both.

3.1.3 Mixed formulations

We identified two mixed formulations. Both use an exchange-based approach for cycles and an arc-based approach for paths. This is relevant as usually K is quite small (3 or 4) and allows the computation of all the cycles beforehand (forming the set $\mathcal{C}$ of all valid cycles). Thus, the following formulations use a binary variable for each valid cycle and binary variables for arcs.

PC-TSP based formulation. Anderson *et al.* proposed a formulation inspired from the prize-collecting traveling salesman problem (PC-TSP) [7] which also uses the idea of cloning the graph. One path can be constructed in each copy of the graph.

Constraints (3.40) and (3.41) express the physiological constraint. Constraints (3.42) are flow constraints for vertices of P modeling the participation constraint. The length constraints are only required for paths and

Model PC-TSP

$$\forall (ij) \in A, \forall l \in [\![1;|N|]\!]\colon\ y_{ij}^l = \begin{cases} 1 \text{ if arc } (ij) \text{ is selected in a path} \\ \qquad\qquad\qquad \text{in the } l^{th} \text{ copy of } D \\ 0 \text{ otherwise} \end{cases}$$

$$\forall c \in \mathcal{C}, x_c = \begin{cases} 1 \text{ if cycle } c \text{ is selected} \\ 0 \text{ otherwise} \end{cases}$$

$$\max \sum_{c \in \mathcal{C}} w_c x_c \ + \sum_{(ij)\in A} \sum_{l=1}^{|N|} w_{ij} y_{ij}^l \tag{3.39}$$

$$\sum_{\substack{c \in \mathcal{C}: \\ i \in V(c)}} x_c \ + \sum_{j \in N^-(i)} \sum_{l=1}^{|N|} y_{ji}^l \leq 1 \qquad \forall i \in P \tag{3.40}$$

$$\sum_{j \in N^+(i)} \sum_{l=1}^{|N|} y_{ij}^l \leq 1 \qquad \forall i \in N \tag{3.41}$$

$$\sum_{j \in N^-(i)} y_{ji}^l \ - \sum_{j \in N^+(i)} y_{ij}^l \geq 0 \qquad \forall i \in P, \forall l \in [\![1;|N|]\!] \tag{3.42}$$

$$\sum_{(ij)\in A} y_{ij}^l \leq L \qquad \forall l \in [\![1;|N|]\!] \tag{3.43}$$

$$\sum_{j \in \delta^-(S)} y_{ji}^l \ - \sum_{j \in N^-(i)} y_{ji}^l \geq 0 \qquad \forall S \subseteq P,\ \forall i \in P, \forall l \in [\![1;|N|]\!] \tag{3.44}$$

$$y_{ij}^l \in \{0,1\} \qquad \forall (ij) \in A \tag{3.45}$$

$$x_c \in \{0,1\} \qquad \forall c \in \mathcal{C} \tag{3.46}$$

defined by (3.43). The set of constraints (3.44) is an adaptation of cut set inequalities of the traveling salesman problem.

Position-indexed chain-edge formulation. The **PICEF** formulation [37] is a mixed version of the position-indexed formulation seen in Section 3.1.2. Recall that $\mathcal{P}_{ij}$ is the set of positions at which arc (ij) can be chosen in a path of D.

Constraints (3.48) and (3.49) express the physiological constraint. Constraints (3.50) are flow constraints for vertices of P modeling the participation constraint. The length constraints are only required for paths and are integrated in the variable definition.

Model PICEF

$$\forall (ij) \in A, p \in \mathcal{P}_{ij}\colon\ y_{ijp} = \begin{cases} 1 \text{ if arc } (ij) \text{ is selected at position } p \text{ in a path} \\ 0 \text{ otherwise} \end{cases}$$

$$\forall c \in \mathcal{C}, x_c = \begin{cases} 1 \text{ if cycle } c \text{ is selected} \\ 0 \text{ otherwise} \end{cases}$$

$$\max \sum_{c \in \mathcal{C}} w_c x_c \ + \sum_{(ij) \in A} \sum_{p \in \mathcal{P}_{ij}} w_{ij} y_{ijp} \tag{3.47}$$

$$\sum_{\substack{c \in \mathcal{C}: \\ i \in V(c)}} x_c \ + \sum_{j \in N^-(i)} \sum_{p \in \mathcal{P}_{ji}} y_{jip} \leq 1 \qquad \forall i \in P \tag{3.48}$$

$$\sum_{j \in N^+(i)} y_{ij1} \leq 1 \qquad \forall i \in N \tag{3.49}$$

$$\sum_{\substack{j \in N^-(i): \\ p \in \mathcal{P}_{ji}}} y_{jip} \ - \sum_{j \in N^+(i)} y_{ij(p+1)} \geq 0 \qquad \forall i \in P, \forall p \in [\![1; L-1]\!] \tag{3.50}$$

$$y_{ijp} \in \{0,1\} \qquad \forall (ij) \in A, \forall p \in \mathcal{P}_{ij} \tag{3.51}$$

$$x_c \in \{0,1\} \qquad \forall c \in \mathcal{C} \tag{3.52}$$

Mixed formulations, even though they are a bit simpler than arc based-based formulations, suffer from the same drawbacks: scaling issues and linear relaxation weakness. Actually, according to Mak-Hau, mixed formulations are weaker than the MTZ-based formulations [82]. Moreover, these arc-based or mixed formulations often have poor linear relaxations, in particular compared to the exchange formulation. Indeed, EF dominates every other formulation presented above (see Appendix A). The exchange formulation is thus promising for the KEP, despite its exponential number of variables. This exponentiality is due to the fact that variables are indexed on exchanges, just like stable-set formulations.

3.2 Stable-set formulations

As explained in Section 2.2.2, any formulation of the maximum-weight stable set problem (MWSSP) on the intersection graph $\mathcal{I}(\mathcal{E})$ can be used to solve the kidney exchange problem. The specificities of the KEP are involved in the construction of $\mathcal{I}(\mathcal{E})$, but not in the stable set problem which is a standard problem. For this reason, the context of the (weighted) stable set problem in this section in wider than the KEP and concerns any simple undirected graph $G = (V, E)$. We first recall key notions on this problem,

then we present and study a new formulation.

3.2.1 Stable set polytopes

A natural formulation, called the *edge formulation* (SS-EF) of the MWSSP is given below:

Model SS-EF

$$\forall v \in V,\ y_v = \begin{cases} 1 \text{ if vertex } v \text{ is selected in the solution} \\ 0 \text{ otherwise} \end{cases}$$

$$z^* = \max \sum_{v \in V} w_v y_v \tag{3.53}$$

$$y_{v_1} + y_{v_2} \leq 1 \qquad \forall (v_1 v_2) \in E \tag{3.54}$$

$$y_e \in \{0,1\} \qquad \forall v \in V \tag{3.55}$$

Constraints (3.54) state that the endpoints of an edge cannot both belong to the solution. In the KEP context, that means that exchanges sharing a common agent (donor or patient) cannot be both performed. SS-EF is known to be weak in general but is a classical and simple formulation.

The polytope defined by the linear relaxation of SS-EF is called the *fractional stable set polytope* of G ($FSTAB(G)$). The **stable set polytope** of G, denoted by $\boldsymbol{STAB(G)}$, is the convex hull of the characteristic vectors of the stable sets of G:

$$\begin{aligned} STAB(G) &:= conv\{\chi^S : S \text{ is a stable set of } G\} \\ &= \{y \in \mathbb{R}^{|V|} : y = \sum_{\substack{S \text{ stable} \\ \text{set of } G}} \lambda_S \chi^S \text{ for some } \lambda_S \geq 0, \\ &\qquad \forall S \text{ stable set of } G \text{ s.t. } \sum_{\substack{S \text{ stable} \\ \text{set of } G}} \lambda_S = 1\} \end{aligned}$$

Many valid inequalities are known and studied for $STAB(G)$ [76], including the **clique inequalities**. They state that for all clique K of G and for all $y \in STAB(G)$, $y(K) := \sum_{v \in K} y_v \leq 1$. The *clique relaxation polytope* of G, denoted by $QSTAB(G)$, is the polytope defined by clique constraints and non-negativity constraints:

$$QSTAB(G) = \{y \in \mathbb{R}^{|V|} : y(K) \leq 1,\ \forall \text{ clique } K \text{ of } G, y \geq 0\}$$

Note that $STAB(G) \subseteq QSTAB(G)$ and the equality holds for perfect graphs. Actually, it is a characterization of this class of graph, due to Chvátal [30] and Padberg [96] (independently): G is perfect $\Leftrightarrow QSTAB(G) = STAB(G)$.

3.2.2 A new formulation for the stable set problem

Recall that if $L \geq K$, then the intersection graph $\mathcal{I}(\mathcal{E})$ has the property to be $K_{1,L+1}$-free (see Section 2.2.1). The formulation proposed below, called **stable neighborhood formulation** (**SNF**) is stronger on this class of graph, as it will be proved after. For any vertex $u \in V$, we denote by G_u the subgraph of G induced by $N[u]$ and $\mathcal{S}_u$ the set of all stable sets of G_u.

Model SNF

$$\forall v \in V\colon\ y_v = \begin{cases} 1 \text{ if vertex } v \text{ is selected in the solution} \\ 0 \text{ otherwise} \end{cases}$$

$$\forall v \in V, \forall S \in S_v\colon\ x_{vS} = \begin{cases} 1 \text{ if stable set } S \text{ is a subset of the solution} \\ 0 \text{ otherwise} \end{cases}$$

$$\max \sum_{v \in V} y_v \tag{3.56}$$

$$\sum_{S \in S_v} x_{vS} = 1 \qquad \forall v \in V \tag{3.57}$$

$$\sum_{\substack{S \in S_u \\ v \in S}} x_{uS} = y_v \qquad \forall v \in V\ \forall u \in N[v] \tag{3.58}$$

$$x_{vS} \in \{0,1\} \qquad \forall v \in V\ \forall S \in S_v \tag{3.59}$$

$$y_v \in \{0,1\} \qquad \forall v \in V \tag{3.60}$$

In the weighted case, let $w : V \to \mathbb{R}$ be a weight function on vertices of G, the objective (3.56) can be replaced by $\max \sum_{v \in V} w_v y_v$. Equations (3.57) state that we must select exactly one stable set in the closed neighborhood of any vertex and equations (3.58) impose *consistency*: if a vertex v is taken, it must also be taken in the stable sets chosen by all its closed neighbors. The idea of the formulation is to describe stable sets of G by looking at their intersection with the neighborhood of any vertex: if S is a stable set of G, then $S \cap N[u]$ is a stable set of G_u. Conversely, one can recombine stable sets in S_u into a single stable set of G if they are consistent. When the stability number $\alpha(G_v)$ is not bounded (for at least one vertex $v \in V$), the formulation is not polynomial.

Let Q be the polyhedron associated with the SNF linear relaxation and P be the projection of Q on the y variables:

$$Q := \{(y,x) : \sum_{S \in S_v} x_{vS} = 1,\ y_v = \sum_{\substack{S \in S_u \\ v \in S}} x_{uS},\ 0 \leq x_{vS} \leq 1,\ 0 \leq y_v \leq 1,$$
$$\forall v \in V,\ \forall u \in N[v], \forall S \in S_v\}$$

$$P := Proj_y(Q) = \{y \in \mathbb{R}^{|V|} : \exists x : (y,x) \in Q\}$$

Property 3.1

$STAB(G) \subseteq P \subseteq QSTAB(G)$

Proof. We show first that for all stable set S of G, there exists an integer solution $(y, x) \in Q : y = \chi^S$, and then that $P \subseteq QSTAB(G)$. This means that we do not miss any stable set and that no integer solution y that is not a stable set of G can be extended to a feasible solution $(y, x) \in Q$.

The first part is trivial. Given a stable set S, we define for all $v \in V$ the stable set $S'_v := S \cap N[v]$ and construct (y, x) as: $y = \chi^S$ and, $\forall v \in V$, $x_{vS'_v} = 1$. By definition, (y, x) satisfies constraints (3.57) and (3.58):

For the second part, let $y \in P$. By definition, we have $y \geq 0$, so we only need to check that it satisfies clique inequalities. Let K be a clique of G and let $u^* \in K$:

$$\sum_{v \in K} y_v = \sum_{v \in K} \sum_{\substack{S \in S_{u^*} \\ v \in S}} x_{u^*S} \overset{(a)}{=} \sum_{\substack{S \in S_{u^*} \\ S \cap K \neq \emptyset}} x_{u^*S} \leq \sum_{S \in S_{u^*}} x_{u^*S} = 1$$

(a) is true because K is a clique so a stable set S contains at most one vertex from K, and thus S is counted only once in the sum. Therefore, we have $y(K) \leq 1$ for all clique K of G. □

Property (3.1) proves that SNF is an extended formulation for the stable set problem, but also that it is ideal for perfect graphs. If G is a $K_{1,k}$-free graph, then $\alpha(G_u) \leq k$ for each vertex $u \in V$ and there are at most $\mathcal{O}(|V|^k)$ stable sets G_u for each vertex $u \in V$, *i.e.*, $\forall u \in V\ |S_u| = \mathcal{O}(|V|^k)$. In this case, there is a polynomial number of constraints (3.58), so SNF is a compact formulation for $K_{1,k}$-free graphs. In particular, SNF is compact for intersection graphs in the kidney exchange problem as they are $K_{1,L+1}$-free. Moreover, SNF is a compact ideal formulation for the stable set problem in claw-free perfect graphs.

We showed that our formulation is not ideal in general, but tighter than the clique relaxation. We know in fact that SNF satisfies every inequalities that are contained inside the closed neighborhood of every vertex, but no other inequalities.

Property 3.2

For any graph $G = (V, E)$, $y \in P \Leftrightarrow \forall u \in V :\ y_{[N[u]]} \in STAB(G_u)$

Proof. We show first that $y \in P \Rightarrow \forall u \in V : y_{[N[u]]} \in STAB(G_u)$, which means that for any solution $(y, x) \in Q$, y is in the intersection of the convex combinations of stable sets in G_u, for all $u \in V$. Then, we show that for any point contained in the intersection of the convex combinations of stable sets in G_u, for all $u \in V$, there exists a solution $(y, x) \in Q$.

$\Rightarrow$
Let $(y, x) \in Q$. For $u \in V$:
By (3.58), we have $y_v = \sum_{\substack{S \in S_u \\ v \in S}} x_{uS} = \sum_{S \in S_u} x_{uS} \chi^S(v), \ \forall v \in N[u]$. It follows that $y_{[N[u]]} = \sum_{S \in S_u} x_{uS} \chi^S_{N[u]}$, that is $y_{[N[u]]}$ is a convex combination of stable sets of $N[u]$ (with multipliers $x_{uS} \geq 0 : \sum_{S \in S_u} x_{uS} = 1$). Hence $y_{[N[u]]} \in STAB(G_u)$.
$\Leftarrow$
Let $y \in \mathbb{R}^{|V|}$: $y_{[N[u]]} \in STAB(G_u), \ \forall u \in V$. By definition, we have that $\forall u \in V$: $\exists \lambda_u \in [0,1]^{|\mathcal{S}_u|}$ such that $\sum_{S \in S_u} \lambda^S_u = 1$ and $y_{[N[u]]} = \sum_{S \in S_u} \lambda^S_u \chi^S_{N[u]}$.
Taking $x_{uS} = \lambda^S_u$ yields the result. □

We already mentioned that many inequalities are studied for the stable set problem. Their validity in the stable neighborhood formulation can be deduced from Proposition 3.2. In particular, the odd wheel inequalities [28], the antiweb-wheel inequalities [29] and the clique constraints (already shown by proposition 3.1) are valid for SNF while the odd cycle inequalities [95], rank inequalities [30], web and antiweb inequalities [121] and grille inequalities [24] are not.

In the context of kidney exchange programs, recall that the stable set problem must be solved on the intersection graph $\mathcal{I}(\mathcal{E})$ which may be impossible to compute integrally when dealing with large scale instances. Even if the stable neighborhood formulation is compact for $\mathcal{I}(\mathcal{E})$, it still is exponential for the KEP it aims at solving. Indeed, variables (and constraints) of SNF are indexed on vertices of $\mathcal{I}(\mathcal{E})$, *i.e.*, exchanges of the compatibility graph D. As explained in Section 1.4.2, a standard column generation framework would fail to correctly solve its linear relaxation, because columns and rows are dependent. That means a complex column-and-row generation scheme is necessary to be able to exploit the interesting properties of SNF.

3.3 Experimental comparison

Exponential formulations solved with column generation can be very advantageous over compact formulations as their linear relaxation is often tighter. This advantage can however be limited by the pricing problem solving, especially when it is NP-hard. To evaluate the computation benefit of each kind of formulation, we compare the exchange formulation with the model MTZ-EAF, which is considered as the best compact formulation for the KEP by Mak-Hau [82]. Our column generation framework solving the exchange formulation, called CG-dyn, is described in the next chapter.

3.3.1 Instances

	Frequency (in %)	
Parameter	Donors	Patients
Type O	34.5	65.1
Type A	45.8	20
Type B	19.7	12.4
Type AB	0	2.5
Spouse	48.97	
Female	40.90	
Low PRA		21.6
Medium PRA		16
High PRA		62.4

	Probability of + crossmatch (%)	
PRA	Spousal*	Others
Low	62.5	50
Medium	85	80
High	98.5	98

Table 3.2 – Parameters used for the KEP graph generator

* the probability of a negative crossmatch is reduced of 75%

The web application www.dcs.gla.ac.uk/~jamest/kidney-webapp/#/generator provides a generator based on Saidman *et al.* [114]. Dickerson *et al.* [40] studied kidney exchange programs to extract real data on the demographic and proposed a set up for the different medical parameters of the generator. We use their results, detailed in Table 3.2, to generate realistic pools of patients and donors. Recall that a positive crossmatch between a donor and a patient means they are incompatible. The average density of graphs generated with this configuration is about 5%.

We created a set of benchmark instances, called KBR, made up of 27 different classes of KEP instances depending on the values of three parameters: the number of pairs ($|P| \in \{50, 100, 250\}$), the percentage of altruistic donors ($p_{|N|} \in \{5, 10, 25\}\%$) and the length limit for paths ($L \in \{4, 7, 13\}$). The size limit for cycle K is always 3.

3.3.2 Exponential versus compact formulations

We run on 135 instances (5 in each class of KBR) our algorithm CG-dyn to solve EF and the compact formulation MTZ-EAF within 2000 seconds[1].

Table 3.3 shows different results depending on the size of the instances for both algorithms. CG-dyn outperforms MTZ-EAF as soon as $|P| = 250$, providing feasible solutions of excellent quality, while MTZ-EAF cannot solve most instances of that size in the time given and provides large gaps for these instances. In average, CG-dyn is better on medium instances but

[1] Complete results available on my webpage: https://pagesperso.g-scop.grenoble-inp.fr/~pansartl/.

	# solved (# opt)		Average time (s)		Average gap (%)	
$\|P\|$	CG-dyn	MTZ-EAF	CG-dyn	MTZ-EAF	CG-dyn	MTZ-EAF
50	45 (44)	45 (45)	1.6	**0.41**	< 0.01	**0**
100	**45** (32)	38 (38)	**10**	435.1*	**0.15**	1.34
250	**45** (17)	9 (9)	**483.7**	1714**	**0.25**	29.29

Table 3.3 – Number of instances for which the algorithm finishes before the time limit, average running time and average gap between upper and lower bound for 135 instances

* including the 7 instances (out of 45) reaching the time limit

** including the 36 instances (out of 45) reaching the time limit

it actually depends on the path length limit, so Table 3.4 shows results for instances with $|P| = 100$ depending on L. The larger L is, the faster MTZ-EAF. This is also true for larger instances: the 9 instances of size 250 that are solved by MTZ-EAF in the time limit all have $L = 12$. Even though the compact formulation is stronger for greater L, it is not sufficient to always get a good solution in the time allocated. The compact formulation also proves to be quicker on very small instances ($|P| = 50$).

	# solved (# opt)		Average time (s)		Average gap (%)	
L	CG-dyn	MTZ-EAF	CG-dyn	MTZ-EAF	CG-dyn	MTZ-EAF
4	**15** (10)	10 (10)	**1.21**	868.3*	**0.13**	2.83
7	**15** (8)	13 (13)	**2.17**	433.5**	**0.3**	1.2
13	15 (14)	15 (15)	26.6	**3.61**	0.01	**0**

Table 3.4 – Number of instances for which the algorithm finishes before the time limit, average running time and average gap between upper and lower bound for the 15 instances with $|P| = 100$

* including the 5 instances (out of 15) reaching the time limit

** including the 2 instances (out of 15) reaching the time limit

We also compared the linear relaxations of the two models and observe that, as expected from their theoretical comparaisons, EF is tighter than MTZ-EAF. Table 3.5 shows the gap between the linear relaxations and the optimal value when it is know (instances solved): $\frac{z^*_{LR} - OPT}{z^*_{LR}}$, as well as the ratio $\frac{\text{MTZ-EAF}}{\text{EF}}$. The bigger ratio for open instances suggests that the gaps in this case may be even more distant.

	# instances	Gap EF	Gap MTZ-EAF	Ratio
Solved	105	**0.07%**	5.12%	1.06
Open	30	–	–	1.11

Table 3.5 – Quality of the linear relaxation

3.4 Conclusion: solving the KEP with integer programming

Basis	Formulation	Number of variables	Number of constraints	Section
Arc	MTZ-AF	Polynomial	Exponential	3.1.2
	MTZ-EAF	Polynomial		3.1.2
	HPIEF	Polynomial		3.1.2
Exchange	EF	Exponential	Polynomial	3.1.1
	SSF	Exponential*		3.2
	SNF	Exponential*		3.2.2
Mixed	PC-TSP	Exponential	Exponential	3.1.3
	PICEF	Exponential	Polynomial	3.1.3

Table 3.6 – Overview of the main formulations for the KEP
* exponential for an input of the KEP, but polynomial in $\mathcal{I}(\mathcal{E})$

We presented in this chapter several formulations from the kidney exchange problem literature. We also introduced a new formulation for the stable set problem, which defines a polytope tighter than the clique relaxation polytope. In particular, this formulation, based on a construction of the stable set by neighborhood, is ideal and compact for perfect and $K_{1,k}$-free graph. Table 3.6 sums up all the formulations studied in this chapter, also detailing their size (exponential or polynomial) with respect to the input $(D; K; L; w)$.

As explained in Chapter 2, kidney exchange programs are expected to significantly increase in the next few years. Compact formulations may seem interesting to exploit, but they actually also suffer from the instance size growth. To deal with the kidney exchange problem in this context, solvers must use techniques recalled in Section 1.4.2. As the exchange formulation is tighter and shows better experimental results for large graphs, we focus on the design of a column generation framework to solve its linear relaxation, detailed in the next chapter. Other avenues for research are not addressed in this thesis but are promising for future work. In particular, we would like

to develop a column-and-row generation for our new stable set formulation SNF, which implies to identify the different pricing subproblems. Moreover, theoretical comparisons between some formulations including paths remain to be done.

Chapter 4

Column Generation for the Exchange Formulation

The exchange formulation, introduced by Roth *et al.* and Abraham *et al.* for cycles only [1, 110], is a large-scale integer linear program modeling the kidney exchange problem (see Chapter 3, in particular Section 3.1.1). The input is the weighted compatibility graph D and the length limits L and K. In this graph, the exchange set $\mathcal{E}$ contains all exchanges (paths and cycles) respecting the length constraints and participation constraint. A decision variable is associated with each exchange of $\mathcal{E}$ and a unique set of constraints (3.2) is required to model the physiological constraint. This formulation grows exponentially with K and L so $\mathcal{E}$ is too big for a standard branch-and-bound (see Section 1.4). This is the reason why a column generation is applied to solve its linear relaxation EFL, presented below.

Model EFL

$$\forall e \in \mathcal{E},\ x_e = \begin{cases} 1 \text{ if exchange } e \text{ is chosen} \\ 0 \text{ otherwise} \end{cases}$$

$$z^*_{LP} = \max \sum_{e \in \mathcal{E}} w_e x_e \tag{4.1}$$

$$\sum_{\substack{e \in \mathcal{E}: \\ v \in V(e)}} x_e \leq 1 \qquad \forall v \in V \tag{4.2}$$

$$x_e \geq 0 \qquad \forall e \in \mathcal{E} \tag{4.3}$$

In the column generation algorithm, the exchanges are not computed beforehand and $\mathcal{E}$ is unknown. Instead, a subset of exchanges $\mathcal{E}'$ is iteratively constructed, by solving alternatively the restricted master problem (RMP)—

the restriction of EFL on the set $\mathcal{E}'$—and the pricing problem. This pricing problem decides if an exchange should be included to $\mathcal{E}'$ and relies on the dual information provided by the master problem. The following problem is the dual of EFL and α_v are the dual variable associated with constraints (4.2). When the RMP is solved, it computes dual values α_v for each vertex v.

Model EFL-Dual

$$\min \sum_{v \in V} \alpha_v \tag{4.4}$$

$$\sum_{v \in V(e)} \alpha_v \geq w_e \qquad \forall e \in \mathcal{E} \tag{4.5}$$

$$\alpha_v \geq 0 \qquad \forall v \in V \tag{4.6}$$

The solution of the pricing problem either augments $\mathcal{E}'$ with a new exchange, *i.e.*, for which a variable is added to RMP, or proves that the optimal solution of the current RMP is the optimal solution of EFL. In this latter case, the algorithm stops and returns z^*_{LP}. This pricing step is crucial in the column generation and is discussed in the next section.

4.1 Pricing problem(s)

The pricing problem of the exchange formulation aims at finding a new exchange with a positive reduced cost or proving that none exists. The reduced cost of an exchange e is given by $rc_e = w_e - \sum_{v \in V(e)} \alpha_v$. It is important to note that these reduced costs are in $\mathbb{R}$ and thus can be positive or negative. As there are two kinds of exchanges, we can decompose this pricing problem into two subproblems:

- The cycle pricing problem: find a cycle of length at most K of positive reduced cost, or prove none exists
- The path pricing problem: find an elementary path of length at most L starting by an altruistic donor and with a positive reduced cost, or prove none exists.

To latter cast the path pricing problem as a **minimization** problem, we define a new weight function associating with each arc the **opposite** of its estimated reduced cost: $\forall (uv) \in A, c_{uv} = -w_{uv} + \alpha_v$. We also construct a new directed graph containing an artificial source s linked to each altruistic donor $D' = (V \cup \{s\}, A')$ where $A' = A \cup \{(su)\ \forall u \in N\}$. The function c is extended to these new arcs: $\forall u \in N, c_{sv} = \alpha_v$. A valid path contains

$L+1$ vertices (including the source) and L arcs in D'. For an exchange e, $c_e = \sum_{(uv)\in A'(e)} c_{uv} = \sum_{v\in V(e)} \alpha_v - w_e = -rc_e$.

In this section the "weight" of an arc (uv) (resp. an exchange e) refers to c_{uv} (resp. c_e), and both D and D' are weighted with this function c. As $c_e = -rc_e$, the pricing problems aim at finding exchanges of **negative weight**.

4.1.1 The cycle pricing problem

Deciding whether a weighted and directed graph contains an elementary cycle of negative weight and limited length can be done in polynomial time with a Bellman-Ford algorithm. The Bellman-Ford algorithm was first designed to compute shortest paths from a source to all vertices in a weighted directed graph [15, 47, 89, 118]. If the graph contains a negative cycle, then the algorithm returns this cycle instead of shortest paths, as they are unbounded. The Bellman-Ford algorithm can thus be used to detect such a cycle instead of computing the shortest paths. To meet the pricing problem objective, some modifications are required, leading to the **modified Bellman-Ford** algorithm 1, used in previous works on column generation for the KEP [37, 102]:

- The algorithm applies $|P|$ Bellman-Ford algorithms, all considering a different patient-donor pair as the source in the algorithm, in fact the "source" of a potential cycle. Actually, since a negative cycle will necessarily visit a vertex with an outgoing negative arc, we can use only the vertices of $P_s = \{u \in P : \exists v \in P | (uv) \in A \text{ and } c_{uv} < 0\}$ as sources.
- Each Bellman-Ford algorithm is limited to K steps to report only cycles of length at most K.

Note that Algorithm 1 is not applied on D' but on D as the source cannot be in a cycle. We can also ignore the set N of vertices representing altruistic donors.

4.1.2 The path pricing problem

Determining if the graph D' contains an elementary path of negative weight, visiting at most L arcs and starting by vertex s is an NP-complete problem. The proof, based on a reduction from the directed Hamiltonian path problem, was recently given by Plaut *et al.* [104] and is quickly presented below.

Proof of NP-completeness. Given some digraph $H = (V_H, A_H)$, the directed Hamiltonian path problem is to decide whether a Hamiltonian path,

Algorithm 1 Modified Bellman-Ford

Input: $D = (V, A)$ weighted with function c and $K \in \mathbb{N}$
$P_s = \{u \in P : \exists v \in P | (uv) \in A, c_{uv} < 0\}$ ▷ potential sources
Output: Either:
a negative cycle using at most K arcs in D,
or there is no such cycle in D.

```
for all s ∈ P_s do                                  ▷ initialization
    for all v ∈ V do
        d[v] = +∞                                   ▷ table of distances from s to v
        p[v] = ∅                                    ▷ table of parents of each vertex v
    d[s] = 0
    for 1 to K do                                   ▷ iterations of BF limited to K
        for all a = (uv) ∈ A  do
            if d[u] + c(u,v) < d[v] then
                d[v] = d[u] + c(u,v)
                p[v] = u
            if d[s] < 0 then                        ▷ found a negative cycle using s
                e = {s}                             ▷ construction of the cycle...
                v = p[s]
                while v != s do                     ▷ ...backtracking parents chain
                    e = v ∪ e
                    v = p[v]
                return e
return No positive cycle in D
```

i.e., a path visiting each vertex exactly once, exists in H. Given this input H, we can construct an input $(D' = (V', A'), L, c)$ of the path pricing problem: $V' = V_H \cup \{s\}$, $A' = A_H \cup \{(sv)\ \forall v \in V_H\}$, $L = |V_H|$ and

$$c_{uv} = \begin{cases} L-2 & \text{if } u = s \\ -1 & \text{if } (uv) \in A_H \end{cases}$$

This reduction is illustrated in Figure 4.1 where the fat arrow means all arcs exist between s and the set of vertices and a waved arrow means any arc (with any orientation) of H exists. If $(v_1, ..., v_L)$ is a Hamiltonian path in H, then $(s, v_1, ..., v_L)$ is an elementary path of D' containing $|V_H| \leq L$ arcs and of total weight -1. Reciprocally, given a solution p to the path pricing problem in D', we can write $p = (s, u_1, ..., u_l)$. As $c_p < 0$ and $c_{su_1} = L - 2$ then p contains exactly L arcs to have a negative weight. Since p is an elementary path, it visits every vertices of D' and $(u_1, ..., u_l)$ is an Hamiltonian path in H.

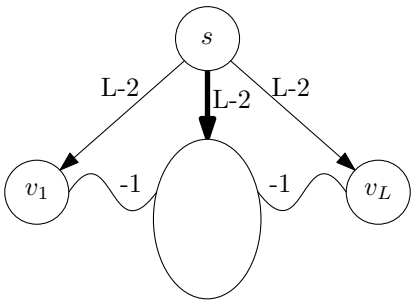


Figure 4.1 – Construction of the reduction from Hamiltonian path problem to path pricing problem

The hardness of the path pricing problem is the main obstacle to overcome when solving the KEP with altruistic donors, as the pricing problem has to be solved a large number of times in general. Another difficulty of this column generation framework is that two separate problems have to be solved in the pricing step. The next section sets out how these different problems are handled in our implementation.

4.1.3 Solving in practice.

In the pricing step, the modified Bellman-Ford is called first, since the cycle pricing is easy to solve. If a cycle of negative weight is found, the pricing step stops here and a new iteration of the column generation begins. Otherwise, the path pricing problem must be solved.

This decision problem is actually handled with algorithms solving—in heuristic, exact or relaxed manner—the associated optimization problem: the **elementary minimum path problem with length con-**

straint (EMPPLC). **Heuristics** provide a feasible solution (giving an upper bound) of EMPPLC and if this solution has a negative weight, then the path is added to the set of exchanges $\mathcal{E}'$ of RMP. **Relaxations** produce lower bounds for EMPPLC, thus if the solution found by a relaxation has a non-negative weight, then the optimality proof of z^*_{LP} needed to end the column generation is done. Otherwise, an **exact algorithm** is required to make this proof. These lower bounds can also be used to compute upper bounds on the master problem solution z^*_{LP} by duality, as detailed in Section 4.3. In this thesis, we study four algorithms for the EMPPLC:

- a local search heuristic designed to quickly find columns.
- the heuristic called color coding [4], deeply studied in Chapter 6.
- the NG-route relaxation [13] providing lower bound for EMPPLC as explained above, but also adding non elementary paths to speed up the column generation convergence.
- a mixed-integer linear program which can be used to compute the final proof of pricing optimality to end the column generation.

These algorithms are studied in Chapter 5 and combined in a column generation scheme presented in Section 4.5. They are called only when the modified Bellman-Ford asserted that every cycle has a positive or nil weight. Thus, an optimal solution of EMPPLC is a path with the minimum weight over **all the exchanges**, and not only overall paths. Accordingly, when the i^{th} pricing step solves the EMPPLC, the optimal value of the pricing is $c^{*i} = \min_{e \in \mathcal{E}} \{ \sum_{v \in V(e)} \alpha^i_v - w_e \}$. In the same way, a relaxation of EMPPLC, with value $c^{r_i} \leq c^{*i}$, gives a lower bound on the minimum weight of an **exchange**, not only of a path.

The difficulty of the pricing step makes worthwhile the development of strategies to speed up the column generation and the KEP solving in general. We focus on two levers: reducing the size of the instance (Section 4.2) and avoiding the implementation of a complete branch-and-price (Section 4.4). Some methods detailed in these sections require to compute upper bounds on the linear relaxation value z^*_{LP}, an issue addressed in Section 4.3.

4.2 Filtering

Reducing the size of a KEP instance involves removing arcs and vertices. Elements of the graph can be, and should be, removed if they are not part of an optimal solution, *i.e.*, they are not *interesting* for the problem. The interest of an arc or a vertex is measured according to two criteria: its feasibility and its optimality.

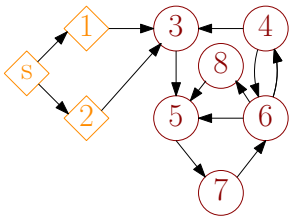
4.2.1 Feasibility

A preprocessing can filter arcs and vertices based on a feasibility criteria, as described in Algorithm 2. The Floyd-Warshall algorithm, or Johnson's for sparse graphs, is used to compute the **distance function** $d : V' \times V' \Rightarrow \mathbb{N} \cup \{+\infty\}$ where $d(u, v)$ is the length of a shortest path, with respect to the **number** of arcs, between u and v. An arc is then removed if its tail is too far from the source (it cannot belong to a path of less than L arcs) and too far from the head of the arc (it cannot belong to a cycle of less than K arcs). Finally, all the isolated vertices are deleted. An example is depicted in Figure 4.2.

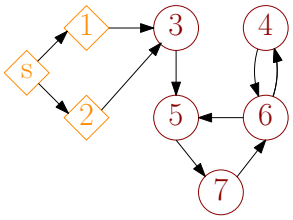
Algorithm 2 Preprocessing

Input: $D' = (V', A')$ and $K, L \in \mathbb{N}$

Floyd-Warshall or Johnson's algorithm $\rightarrow d(u, v)$ ▷ distance function
for all $(uv) \in A$ **do** ▷ initialization
  if $d(s, u) \geq L$ and $d(v, u) \geq K$ **then**
    remove (uv) from A and A'
remove isolated vertices



(a) Before preprocessing



(b) After preprocessing

arc (uv)	$d(s, u)$	$d(v, u)$
$(3, 5)$	2	4
~~$(4, 3)$~~	~~5~~	~~4~~
$(4, 6)$	5	1
$(5, 7)$	2	2
$(6, 4)$	4	1
$(6, 5)$	4	2
$(7, 6)$	3	2
~~$(8, 5)$~~	~~5~~	~~3~~
~~$(6, 8)$~~	~~4~~	~~3~~

(c) Table of distances. Strikethrough lines represent arcs to remove.

Figure 4.2 – Example of preprocessing 2 on a compatibility graph, with $L = 4$ and $K = 3$.

This algorithm is applied as a preprocessing for any instance to remove all the unnecessary arcs and vertices. It can be interesting to also apply it later, and several times, in the column generation. Indeed, as discussed in

the next section, arcs may be removed because of their suboptimality, changing the graph structure and the distances. Algorithm 2 is however quite costly ($O(|V|^3)$ operations for Floyd-Warshall and $O(|V|^2 log(|V|) + |V||A|)$ for Johnson's) and thereby must not be overused.

4.2.2 Optimality

An arc can also be removed when it is proven to be suboptimal, *i.e.*, it cannot belong to an optimal solution. Such a proof requires to have a lower bound and an upper bound of the solution, but also, for each arc, a bound on the benefit of taking it in the solution. The marginal cost of an arc measures this benefit and if it is too small regarding what is already known about the optimal solution, the arc can be removed. Each iteration of the column generation reports primal and dual information that can be used to define a filtering rule detecting some of these suboptimal arcs.

Marginal cost of an arc. Informally, the **marginal cost** λ^*_{uv} of an arc (uv) is a lower bound on the decrease of the upper bound value when including this arc in the solution. Its calculation derives from the duality theory. Consider the linear program EFL_{uv} imposing to take arc (uv) with constraint (4.7):

Model EFL_{uv}

$$z^*_{LP} = \max \sum_{e \in \mathcal{E}} w_e x_e \tag{4.1}$$

$$\sum_{\substack{e \in \mathcal{E}: \\ v \in V(e)}} x_e \leq 1 \qquad \forall v \in V \tag{4.2}$$

$$\sum_{\substack{e \in \mathcal{E}: \\ (uv) \in A(e)}} x_e = 1 \tag{4.7}$$

$$x_e \geq 0 \qquad \forall e \in \mathcal{E} \tag{4.3}$$

Let λ_{uv} be the dual variable associated with constraint (4.7). The dual problem of EFL_{uv} is defined as follows:

Model EFL$_{uv}$-Dual

$$\min \sum_{v \in V} \alpha_v + \lambda_{uv} \tag{4.8}$$

$$\sum_{v \in V(e)} \alpha_v + \lambda_{uv} \geq w_e \qquad \forall e \in \mathcal{E} : (uv) \in A(e) \tag{4.9}$$

$$\sum_{v \in V(e)} \alpha_v \geq w_e \qquad \forall e \in \mathcal{E} : (uv) \notin A(e) \tag{4.10}$$

$$\alpha_v \geq 0 \qquad \forall v \in V \tag{4.11}$$

$$\lambda_{uv} \in \mathbb{R} \tag{4.12}$$

Solving a linear program for each arc to compute its marginal cost is intractable, but the pricing step might provide this marginal cost, or a upper bound on it, without additional effort. Indeed, with the linear relaxation of the primal problem, we can build a dual feasible solution by setting $\lambda^*_{uv} = \max_{\substack{e \in \mathcal{E} \\ (uv) \in A(e)}} \{w_e - \sum_{v \in V(e)} \alpha_v\}$.

Filtering rule. Assume $\underline{z^i}$ and $\overline{z^i}$ are lower and upper bounds of the restricted master problem at the i^{th} iteration. The marginal cost, or a upper bound, λ^i_{uv} is also known for each arc (uv). Note that we can replace constraint (4.7) by $\sum_{\substack{e \in \mathcal{E}: \\ (uv) \in A(e)}} x_e \geq 1$, the equality being held by constraints (4.2). This enlightens the fact that $\lambda_{uv} \leq 0$ so taking arc (uv) actually decreases the value of the upper bound. If this decrease is too important so that the objective value becomes lower than the best lower bound, we know arc (uv) is suboptimal. Formally, if $\overline{z^i} + \lambda^i_{uv} < \underline{z^i}$ then arc (uv) is removed since no cycle or path using it can provide a better solution than the best known at step i.

This filtering procedure requires to get a an upper bound on the optimal value. The better this upper bound is, the most arcs and vertices can be removed. Primal and dual relation offers the opportunity to compute such bounds.

4.3 Dual bounds

An upper bound of z^*_{LP} can be computed for free each time a lower bound of the elementary minimum path problem with length constraint is found during the pricing step. The construction, and proof, of this bound results from an auxiliary linear program, called EFL$_\beta$, adding the following redundant

constraint to EFL:

$$\sum_{e\in\mathcal{E}} x_e \leq \frac{|V|}{2} \tag{4.13}$$

4.3.1 Construction

This constraint is valid for every solution of the KEP as it stipulates that the number of exchanges is at most half of the number of agents, or, in other words, that each exchange contains at least two agents.

Model EFL$_\beta$

$$z^*_{LP} = \max \sum_{e\in\mathcal{E}} w_e x_e \tag{4.1}$$

$$\sum_{\substack{e\in\mathcal{E}:\\ v\in V(e)}} x_e \leq 1 \qquad \forall v \in V \tag{4.2}$$

$$\sum_{e\in\mathcal{E}} x_e \leq \frac{|V|}{2} \tag{4.13}$$

$$x_e \geq 0 \qquad \forall e \in \mathcal{E} \tag{4.3}$$

Let β be the dual variable associated with constraint (4.13). The dual problem of EFL$_\beta$ is defined as follows:

Model EFL$_\beta$-Dual

$$\min \sum_{v\in V} \alpha_v + \frac{|V|}{2}\beta \tag{4.14}$$

$$\sum_{v\in V(e)} \alpha_v + \beta \geq w_e \qquad \forall e \in \mathcal{E} \tag{4.15}$$

$$\alpha_v \geq 0 \qquad \forall v \in V \tag{4.16}$$

$$\beta \geq 0 \tag{4.17}$$

From the weak duality theorem, we know that the value of any feasible solution (α, β) of EFL$_\beta$-Dual is an upper bound for z^*_{LP}. Assume the column generation is in its i^{th} iteration. On the one hand, the restricted master problem provides the optimal value of the current RMP z^{*i}_{LP} and the vector of dual values α^i. From the strong duality theorem, we know that $\sum_{v\in V} \alpha^i_v = z^{*i}_{LP}$. On the other hand, if the pricing step computes an optimal solution of EMPPLC, it provides $c^{*i} = \min_{e\in\mathcal{E}}\{\sum_{v\in V(e)} \alpha^i_v - w_e\}$. Assume $c^{*i} < 0$ so that at least one improving column has been found. As a result,

$(z^{*i}_{LP}, -c^{*i})$ is a feasible solution of EFL$_\beta$-Dual and a dual upper bound UB^i_D on z^*_{LP} can be computed:

$$UB^i_D = z^{*i}_{LP} - \frac{|V|}{2}c^{*i} \tag{4.18}$$

However as discussed in Section 4.1.3, EMPPLC is a hard problem and the pricing step might fail to get c^{*i} and compute instead a lower bound for EMPPLC: $c^{r_i} \leq c^{*i}$. This lower bound can come from any relaxation of EMPPLC (*e.g.*, the NG-route or the linear relaxation of the mixed-integer program) and still allows to compute an upper bound UB^i_r on z^*_{LP} which respects $z^* \leq UB^i_D \leq UB^i_r$:

$$UB^i_r = z^{*i}_{LP} - \frac{|V|}{2}c^{r_i} \tag{4.19}$$

Note that when the pricing problem ends up with $c^{*i} = 0$, this means that no exchange is interesting to add and the column generation stops with the optimal solution of EFL. This is clearly visible with these bounds. Indeed, z^{*i}_{LP} is a lower bound on z^*_{LP} as RMP is a restriction of EFL. So when $c^{*i} = 0$, we have: $z^*_{LP} \leq UB^i_D = z^{*i}_{LP} \leq z^*_{LP}$ so $z^*_{LP} = z^{*i}_{LP}$. The same reasoning holds for $c^{r_i} = 0$.

The upper bound can be equivalently seen as the Lagrangian relaxation of constraint (4.13). Its quality depends on this constraint, in particular on the value $\frac{|V|}{2}$, which could be strengthen by reasoning on feasible solutions. The computation of this bound is important in order to perform filtering (Section 4.2.2), but also to qualify feasible solutions found by non-exact algorithms. Such algorithms also rely on the fact that, at the end of the column generation, the current set of generated exchanges $\mathcal{E}'$, a fractional solution $\overline{x}$ of value $\overline{z}$ and an upper bound UB on z^* are returned.

4.3.2 End of the column generation

Ideally, the column generation ends with the optimality proof of the linear relaxation and $\overline{x}$ is optimal for EFL: $UB = \overline{z} = z^*_{LP}$. This optimality is however not guaranteed in two situations, but a (weaker) upper bound can still be computed. First, if the solution contains non elementary paths then $\overline{x}$ is the solution of a relaxed EFL. In this case, $\overline{z}$ is an upper bound anyway but the solution is not valid: we say that a solution of the KEP (fractional or integer) is *valid* if it contains only elementary exchanges. Second, if the column generation stopped because of a limit on the running time, $\overline{z}$ and z^*_{LP} are incomparable. UB is then computed as explained in the previous section.

If the solution $\overline{x}$ is optimal for EFL, but also integer and valid, *i.e.*, is feasible, then it is the optimal solution of EF and the kidney exchange problem is optimally solved: $\overline{z} = z^*_{LP} = z^*$. Otherwise, a feasible solution

can be found via heuristics. In general this solution is not optimal, but its quality can be assessed against UB. In practice the optimality is often reached, and when it is not, the gap with the upper bound is very small (see Section 4.6). The next section describes two of these heuristics.

4.4 Finding an integer solution

The common technique to find an integer solution of a linear program is the branch-and-bound described in Section 1.4. Using a column generation in this framework changes the name to branch-and-price but does not change the principle: at each node of the search tree, the linear relaxation must be computed. Considering that the linear relaxation of the KEP is hard to get due to the hardness of the pricing problem, it would be opportune to solve it only once.

4.4.1 Integer programming

The first method simply solves the integer program EF restricted on the subset of valid exchanges found by the column generation. If $\mathcal{E}'$ contained invalid exchanges, they are removed. This technique can however be too costly in some cases, as it requires to solve an integer program. Contrarily, the approximation presented in the next section is a polynomial-time algorithm.

4.4.2 Iterative rounding approximation

Assume $\overline{x}$ is a valid fractional solution returned by the column generation (non-elementary paths, if any, are removed of the solution). The algorithm 3 describes an approximation based on iterative rounding, a method developed for many combinatorial problems by Lau, Ravi and Singh [75]. It selects an exchange of maximum weight and keep it in the integer solution while removing all the exchanges intersecting it.

Recall that $\forall e \in \mathcal{E}'\ \overline{x_e} \in [0,1]$. For any solution x, we define $\forall v \in V$, $x(v) = \sum_{\substack{e \in \mathcal{E}' \\ v \in V(e)}} x_e$, and say that a vertex v is *saturated* if $x(v) = 1$.

An example of execution is pictured in Figure 4.3. Four exchanges are involved in the fractional solution 4.3a: $e_1 = 5-7-6$ (in gold); $e_2 = 4-6$ (in turquoise); $e_3 = 1-3$ (in beige) and $e_4 = 2-3$ (in brown). Their fractional value does not matter for this instance[1]. In the first iteration 4.3b, e_1 is kept as it has a weight of 12, so e_2 is removed. In the second (and last) iteration 4.3c, the maximum exchange with a fractional value is e_3, hence the suppression of e_4.

[1] Actually, this example is very naive as the fractional solution is worse than the integer solution, no matter x values.

Algorithm 3 Iterative rounding

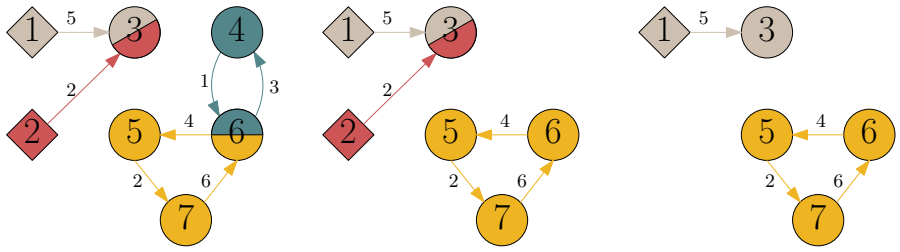
Input: $\overline{x} : \mathcal{E}' \to [0,1]$ a valid solution
Output: $x : \mathcal{E}' \to \{0,1\}$

$x = \overline{x}$
$S = \{e \in \mathcal{E}' : 1 > x_e > 0\}$ ▷ exchanges with fractional value
while $S \neq \emptyset$ **do**
  $e_{max} = \arg\max_{e \in S} w_e$
  $x_{e_{max}} = 1$
  $S = S \setminus \{e_{max}\}$
  for all $u \in V(e_{max})$ **do**
    for all $e \in S$ such that $u \in V(e)$ and $e \neq e_{max}$ **do**
      $x_e = 0$
      $S = S \setminus \{e\}$
  $S = \texttt{saturate}(x)$ ▷ see Algorithm 4
return x

Algorithm 4 Saturate

Input: $x \in \mathcal{E}' \to [0,1]$
Output: S: the set of exchanges with fractional value

for all $x \in \mathcal{E}' : x_e > 0$ **do**
  Increase x_e until one of its vertex is saturated
return $S = \{e \in \mathcal{E}' : 1 > x_e > 0\}$



(a) Fractional solution (b) (c) Integer solution

Figure 4.3 – Execution of Algorithm 3 on an example

At each loop of the iterative rounding, the value of the solution is decreased of at most $(L-1)w_{e_{max}}$ (assuming $L \geq K$). Indeed, each exchange weighs at most $w_{e_{max}}$ and at each vertex of e_{max}, exchanges totalizing a value of at most 1 are removed. Thus, removed exchanges are worth at most $Lw_{e_{max}}$ but an exchange of weight $w_{e_{max}}$ is kept. When the iterative rounding algorithm 3 is applied on the optimal linear solution, it returns a feasible solution with a guaranteed quality.

Property 4.1

Given a valid fractional solution $\overline{x}$, Algorithm 3 returns a feasible solution x for the KEP in polynomial time.
Moreover, assume that $L \geq K$ and $\overline{x}$ is the optimal solution of EFL, this solution x has a value z_{apx} such that: $\frac{z^*}{L} \leq z_{apx} \leq z^*$.

Proof. Let $\overline{x}$ be a valid fractional solution of EFL. The first part of the property is due to the following statements:

- The algorithm ends: at each loop, an exchange e_{max} and all the exchanges intersecting e_{max} are removed from the set S so the loop finishes.
- The algorithm runs in polynomial time: in the worst case, the loop visits every exchange of the support of the linear relaxation (exchange e such that $\overline{x_e} > 0$). Since there is a polynomial (in $|V|$) number of constraints, this support has $O(|V|)$ elements.
- The algorithm returns an integer solution: this is the condition of ending ($S = \emptyset \Leftrightarrow x_e = 1$ or $0\ \forall e \in \mathcal{E}'$).
- The algorithm returns a feasible solution: in the output solution x, each vertex v is involved in at most one exchange with value 1 and then $x(v) \leq 1$. Moreover each exchange is elementary since $\overline{x}$ is valid.

Now assume that $\overline{x}$ is the optimal solution of EFL and $L \geq K$. Let us prove that $\frac{z^*}{L} \leq z_{apx} \leq z^*$. Recall that $z^*_{LP} = \sum_{e \in \mathcal{E}'} w_e \overline{x_e}$. We denote by $\overline{\mathcal{E}}$ the support of solution $\overline{x}$, *i.e.*, $\overline{\mathcal{E}} = \{e \in \mathcal{E}' : \overline{x_e} > 0\}$. Algorithm 3 returns a

solution x of value $z_{apx} = \sum_{e \in \xi} w_e$, where ξ is the set of exchanges from the initial solution $\overline{x}$ kept by the algorithm. For each exchange $e_m \in \xi$, $\mathcal{E}_{e_m}$ denotes the set of exchanges of $\overline{\mathcal{E}}$ intersecting e_m: $\mathcal{E}_{e_m} := \{e \in \overline{\mathcal{E}} : e \cap e_m \neq \emptyset\}$. $\forall u \in V(e_m)$, $\mathcal{E}_{e_m}(u)$ is the set of exchanges in $\mathcal{E}_{e_m}$ containing u. Note that $\mathcal{E}_{e_m} = \bigcup_{u \in V(e_m)} \mathcal{E}_{e_m}(u)$. As $z^*_{LP} = \sum_{e \in \xi} w_e \overline{x_e} + \sum_{e \notin \xi} w_e \overline{x_e}$, we have:

$$
\begin{aligned}
z^*_{LP} &\leq \sum_{e_m \in \xi} \left(w_{e_m} \overline{x_{e_m}} + \sum_{e \in \mathcal{E}_{e_m}} w_e \overline{x_e} \right) \\
&\leq \sum_{e_m \in \xi} \left(w_{e_m} \overline{x_{e_m}} + \sum_{u \in V(e_m)} \sum_{e \in \mathcal{E}_{e_m}(u)} w_e \overline{x_e} \right)
\end{aligned}
$$

Given an exchange $e_m \in \xi$, we have:

1. e_m was chosen with a maximal weight so $\forall e \in \mathcal{E}_{e_m}\ w_e \leq w_{e_m}$
2. $\forall u \in V(e_m)\ \overline{x(u)} \leq 1$
3. e_m contains at most L vertices: $V|(e_m)| \leq L$

Thus,

$$
\begin{aligned}
z^*_{LP} &\leq \sum_{e_m \in \xi} \left(w_{e_m} \overline{x_{e_m}} + \sum_{u \in V(e_m)} w_{e_m} \cdot \sum_{e \in \mathcal{E}_{e_m}(u)} \overline{x_e} \right) && \text{(from 1.)} \\
&\leq \sum_{e_m \in \xi} \left(w_{e_m} \overline{x_{e_m}} + \sum_{u \in V(e_m)} (1 - \overline{x_{e_m}})\, w_{e_m} \right) && \text{(from 2.)} \\
&\leq \sum_{e_m \in \xi} (w_{e_m} \overline{x_{e_m}} + L(1 - \overline{x_{e_m}}) w_{e_m}) && \text{(from 3.)} \\
&\leq \sum_{e_m \in \xi} (L w_{e_m} \overline{x_{e_m}} + L(1 - \overline{x_{e_m}}) w_{e_m}) = \sum_{e_m \in \xi} (L w_{e_m}) = L z_{apx}
\end{aligned}
$$

Finally $z^* \leq z^*_{LP} \leq L z_{apx}$ and, z_{apx} being the value of a feasible solution, $z_{apx} \leq z^*$, hence the result $\frac{z^*}{L} \leq z_{apx} \leq z^*$. □

Both proposed techniques provide a feasible solution that may be non optimal. Its quality can be assessed against the best upper bound of the problem. When the gap between them is strictly smaller than 1, the integer solution is proven to be optimal. The quality of this upper bound depends on how it is computed, as explained in Section 4.3.2.

4.5 Complete algorithm

Combining the features introduced in this chapter, we designed an algorithm to solve the kidney exchange problem with altruistic donors. The choice of

the methods was led by the numerous experiments conducted in a preliminary phase. In particular, many configurations of the EMPPLC algorithms were compared and computational results can be found in Chapter 5.

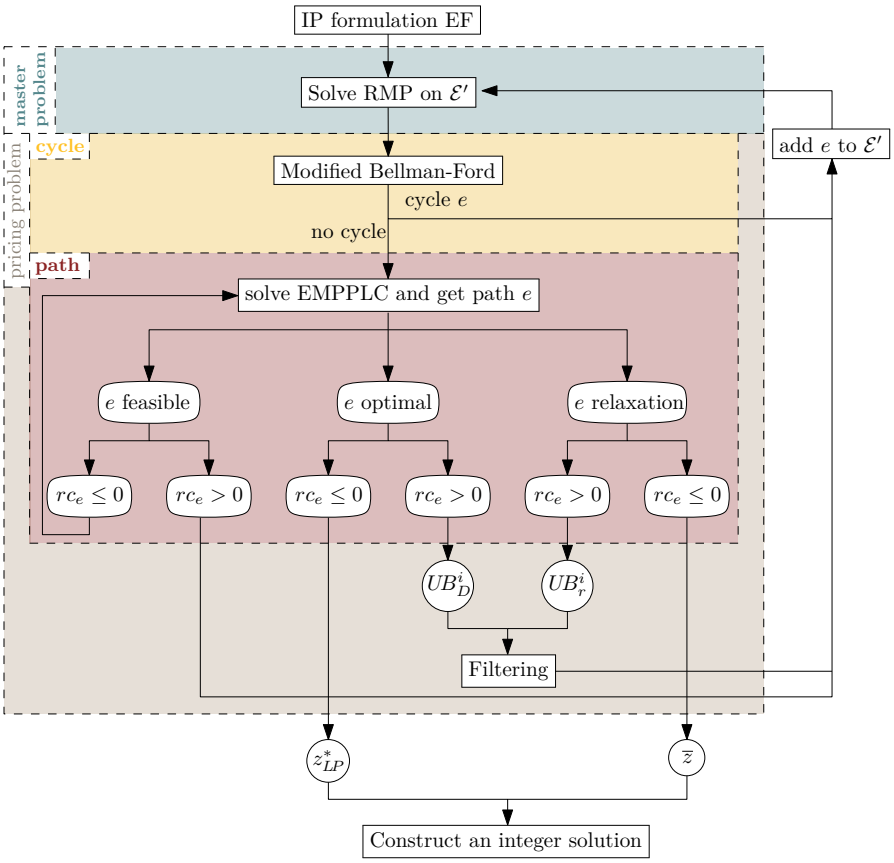


Figure 4.4 – General column generation scheme for the KEP

4.5.1 General approach

The general column generation approach solving the KEP is depicted in Figure 4.4. The scheme is simple: the pricing step stops as soon as an exchange of positive reduced cost is found (even if it does not have the best reduced cost) and the pricing problem is decomposed into two pricing subproblems as explained in Section 4.1. When a cycle of positive reduced cost is found with the Bellman-Ford algorithm, a new iteration of the column generation begins. Otherwise, the path pricing problem must be solved and this is done by considering the elementary minimum path problem with length constraint. Depending on the algorithm applied, the solution can be feasible, relaxed or optimal for the EMPPLC, leading to different actions to continue the column generation. In particular, when the solution is feasible

with a negative reduced cost, the path pricing problem must be solved again. Indeed, no improving exchange was found and nothing can be concluded about the state of the column generation. Of course, another algorithm should be called to get another solution.

When the solution is optimal, or a relaxation, and its reduced cost positive, an upper bound on the linear relaxation z^*_{LP} can be computed (see Section 4.3). With this bound, and other dual information, arcs and vertices can be filtered with the rules described in Section 4.2. At the end, when no more columns can improve the objective value, the column generation stops. If the final solution is fractional, the integer program or the approximation of Section 4.4 can be used to get an integer solution.

4.5.2 Our implementation

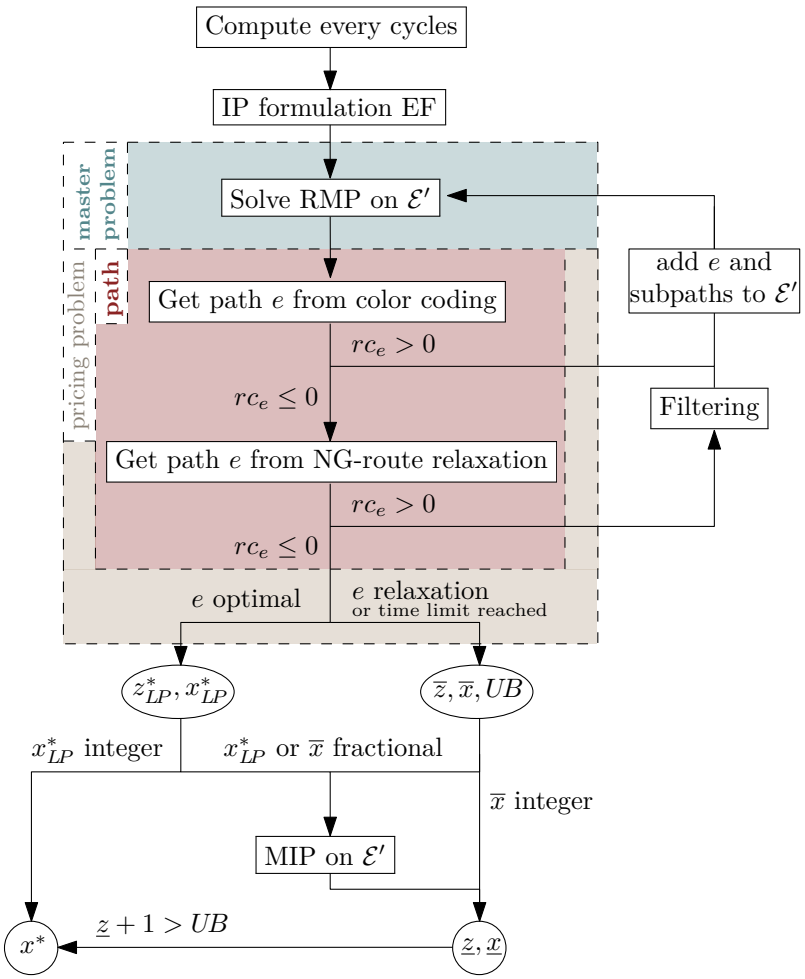


Figure 4.5 – Our algorithm solving the kidney exchange problem: CG-dyn

In this thesis, we consider only $K = 3$ and preliminary experimental results reveal that in this case, computing cycles beforehand is more efficient. These results, summarized in Section 4.6.1, also demonstrate that adding all the subpaths of the EMPPLC solution speeds up the column generation. Concretely, when the path $(s, v_1, ..., v_l)$ is added to $\mathcal{E}'$, we also add the paths $(s, v_1, ..., v_{l-1})$, $(s, v_1, ..., v_{l-2})$, and so on until (s, v_1, v_2).

Conclusions of Chapter 5 led us to use only the two dynamic programs (color coding and NG-route) to solve the EMPPLC of the path pricing step. The color coding is limited to 1 second at each iteration of the column generation. Thus, if the color coding fails to find an improving path within this time limit, the NG-route relaxation is called. After the column generation, the final integer solution is computed with the integer program exchange formulation restricted on $\mathcal{E}'$ as it runs very quickly, especially compared to the column generation running time. The complete framework of our implementation called **CG-dyn** is shown in Figure 4.5.

4.6 Experimentations

We developed several sets of tests to assess the performance of the new algorithm that we propose in this chapter. Section 4.6.1 presents the results of a first phase, which, in order to design the best framework for the column generation, evaluates the effect of different parameters on its performance (*e.g.*, adding subpaths). A second testing phase compares the performance of our algorithm CG-dyn to a more naive column generation which only uses the local search and the time-stage formulation to solve the EMPPLC. We implemented this other column generation algorithm, called CG-tsf, to figure out the importance of solving this subproblem and results are analyzed in Section 4.6.2[2]. These experiments were conducted on instances generated as explained in the Section 3.3.1. In particular, the number of pairs varies from 50 to 250, $L \in \{4, 7, 13\}$ and $K = 3$.

4.6.1 Column generation configuration

A first testing phase was conducted to evaluate the interest of integrating all the cycles beforehand or to run the Bellman-Ford algorithm during the pricing step. The benefit of adding subpaths of an EMPPLC solution was also demonstrated in this phase. We run, within a time limit of one hour, our column generation framework with these two parameters varying on 27 instances: 1 in each instance class. Table 4.1 shows the average computation time for the four possible configurations, depending on L and $|P|$.

[2]Complete results available on my webpage: https://pagesperso.g-scop.grenoble-inp.fr/~pansartl/.

	L			$\|P\|$			
	4	7	13	50	100	250	total
No subpaths + BF	4.8	18.1	830.1*	1.7	10.6	840.8*	284.4
No subpaths + no BF	1.9	11.6	884.4*	1.8	9.8	886.3*	299.3
Subpaths + BF	4.2	15.5	462.6	**1.6**	8.8	471.9	160.8
Subpaths + no BF	**1.8**	**10.2**	**435.8**	**1.6**	**7.9**	**438.2**	**149.2**

Table 4.1 – Average time (in seconds) to run the column generation on 27 instances, depending on L or $|P|$.

* one instance reached the time limit of one hour.

Adding subpaths is clearly an advantage in the column generation, a result which is consistent with the theoretical analysis. Indeed, this strategy may increase the execution time of the RMP in each iteration, but in our model, the master problem is not the limiting factor of the column generation performance. On the contrary, it strongly depends on the efficiency of the pricing problem, and more particularly the EMPPLC. Yet, adding subpaths can reduce the number of iterations and therefore the number of times the EMPPLC must be solved.

Computing all the cycles before the column generation so that they form the initial set $\mathcal{E}'$ is interesting in this setting, in particular for larger graphs. As even larger instances are expected to be solved, we choose to generate all the cycles, thus to not apply the Bellman-Ford algorithm, in our advanced experiments (see Figure 4.5). These results were observed for $K = 3$ and different conclusions could emerged with greater values of K.

4.6.2 Performance of the pricing step

The efficiency of dynamic programming approaches to solve the elementary minimum path problem with length constraint was demonstrated in Chapter 5. To find out if this is still the case when its embedded in a column generation, we implemented a naive column generation called CG-tsf, which handles the pricing step with the local search heuristic and the time-stage formulation. Every other parameter of the column generation framework is the same for both algorithms, in particular they generate all cycles in the master problem and add every subpath in $\mathcal{E}'$. They are applied on 135 instances (5 in each class of KBR), each within a time limit of 2000 seconds.

Table 4.2 shows that our algorithm always finds the linear relaxation EFL while CG-tsf reaches the time limit for 15 instances (those with $|P| = 250$ and $L = 13$). Consequently, our column generation algorithm finds the optimal solution of the integer program exchange formulation for more instances. For both algorithms, this integer solution in mostly found because the linear

	CG-dyn	CG-tsf
# z^*_{LP} computed	135	120
Average gap between UB and LB	0.13 %	3.02 %
# z^* computed (gap = 0)	93	84

Table 4.2 – Results for the two different CG schemes on 135 instances after 2000 seconds of running time

L	CG-dyn	CG-tsf
4	1.6	8.9
7	10	34.3
13	15.6	673.1

Table 4.3 – Average running time (in seconds), depending on L, for the 120 instances solved in the time limit

relaxation is integer (and valid), and in this case there is no need to call the integer program after the column generation. Moreover, considering the 120 instances for which both methods find the linear relaxation in 2000 seconds, the running time is significantly smaller for CG-dyn than CG-tsf (see Table 4.3). These results support the efficiency of the dynamic programs to solve the path pricing problem in a column generation for the KEP.

4.6.3 Scaling up

In Section 3.3.2, we saw that the compact formulation MTZ-EAF failed to solve instances with only 250 pairs in 2000 seconds. Our algorithm could handle these instances in a reasonable amount of time giving solution of very good quality. As kidney exchange programs are growing, we experiment CG-dyn on larger instances, generated as described in Section 3.3.1, but with different parameters. In particular, it seems reasonable to consider that most of the altruistic donors are already included in kidney exchange programs, unlike patients for whom such programs are very different than the standard procedure. Thus, the proportion of altruistic donors would probably be low in future large programs. In the end, we apply CG-dyn on 20 instances, divided in 4 classes ($L \in \{4, 7\}$, $|P| \in \{500, 750\}$, $p_{|N|} = 1\%$).

Results, summarized in Table 4.4, show that the quality of the solution is still very high. Every instance was solved in less than half an hour and in average rather quickly. We also tried to solve instances with $L = 13$ or $|P| = 1000$, but encountered memory issues. However, our implementation does not profit from the fact that instances are quite sparse, so a more efficient implementation should overcome these memory errors. Moreover,

while solving the final integer programming corresponds in average to less than 1% of the running time on realistic instances, for these large instances it represents more than 5%, sometimes almost 15%. It is likely that this proportion will increase with the size of instances, making necessary the development of new algorithms to find feasible solutions.

	$\|P\| = 500$		$\|P\| = 750$	
	$L = 4$	$L = 7$	$L = 4$	$L = 7$
Average gap between UB and LB	0.05%	0.18%	0.06%	0.23%
Average running time (seconds)	5.6	123.6	23.1	1823.5

Table 4.4 – Results of CG-dyn on 40 large instances

4.7 Conclusion: a new column generation scheme for the exchange formulation

In this chapter, we designed a complete column generation framework to solve the kidney exchange problem including altruistic donors. Due to the hardness of the pricing problem in this case, an extension of previous column generation schemes containing only cycles was not possible. Using the study of the elementary minimum path problem with length constraint, which is detailed in the next chapter, we proposed an algorithm showing excellent results on realistic instances and promising for larger instances. Indeed, we believe that the memory issues encountered for instances with $|P| = 1000$ can be avoided with an implementation of the column generation using algorithms and data structures adapted to large and sparse instances. For example, the preprocessing step computing the distances between each pair of vertices could be performed with a Johnson's algorithm instead of Floyd-Warshall. Other avenues of research can be explored in future work, in particular the development of algorithms to get feasible solutions during and after the column generation. Besides providing an integer solution, working on these feasible solutions will strengthen the filtering in both ways. Not only it will raise the best lower bound, but it will also strengthen the valid inequality (4.13) used to compute the dual upper bound.

Chapter 5

Solving the elementary minimum path problem with length constraint

PREVIOUSLY in this thesis, we explained why the pricing problem of the column generation for the exchange formulation involves a path problem that we call elementary minimum path problem with length constraint (EMPPLC). This problem can be encountered in other domains, so this chapter is designed to be self-contained and focuses on EMPPLC in a general context. The formal definition of the problem, its characterization in the literature and solving methods are presented for any directed graph $G = (V, A)$ with a cost function c on arcs. Experiments are however developed for a column generation framework, on instances of the KEP.

5.1 Dealing with a path problem

EMPPLC belongs to the well-known family of paths problems. It is a special case of the elementary shortest path problem with resource constraints. However its specificity—the length constraint—can be exploited to strengthen existing algorithms.

5.1.1 Definition

Let $G = (V, A)$ be a digraph such that the set of vertices V includes a source s. Each arc $(ij) \in A$ has a cost c_{ij}. Let L be the limit on the length (number of arcs) of a path. An **(l, i)-path** $p = (s, i_1, ..., i_l = i)$ is an elementary path of length l starting from the source s and ending in i. We denote by $A(p)$ (resp. $V(p)$) the set of arcs (resp. vertices) of p and by $c_p = \sum_{(ij) \in A(p)} c_{ij}$ its

cost. The objective of the **elementary minimum path problem with length constraint** (**EMPPLC**) is to find p^* an elementary (l, i)-path of minimum cost c^* such that $l \leq L$.

Path problems. Interest for elementary shortest path problems mainly arose from vehicle routing applications solved by column generation. The elementarity constraint is often relaxed to get a simpler problem, which can be relevant in many applications as the vehicle can go twice to the same place. In the standard case, one just wants to solve the *shortest path problem* (SPP), which can be done with a Bellman-Ford algorithm. In the presence of resource constraints, such as time windows or vehicle capacities, the problem becomes harder. In 1988, Desrochers [36] proposed an extension of the Bellman-Ford algorithm for the *shortest path problem with resource constraints* (SPPRC). However, Feillet *et al.* [44] argued that relaxing the elementarity constraint can leads to bounds of poor quality, thus proposed to extend Desrochers' algorithm in order to solve the NP-hard **elementary shortest path problem with resource constraints** (**ESPPRC**). In the ESPPRC, paths have limited resources (instead of having a maximum length as in EMPPLC). Let R be the number of resource types and $g_{ij}^r \geq 0$ the consumption of resource r along the arc (ij). Each vertex $i \in V$ constrains the path to reach it with a resource consumption belonging to $[a_i^r, b_i^r]$ for each resource r. The objective is to find a path of minimum cost c such that every resource constraint is satisfied. By considering a single resource with a unit consumption and by setting the bound on this resource consumption to the length limit, ESPPRC describes EMPPLC. Formally let $R = 1$ and, $\forall i \in V : a_i = 0$ and $b_i = L$. In addition we set $g_{ij} = 1, \forall (ij) \in A$ and observe that EMPPLC is a special case of ESPPRC.

A small note about the kidney exchange problem. In the KEP context and previous chapters, the elementary minimum path problem with length constraint is applied on D', *i.e.*, the compatibility graph with an extra source. The cost of arcs changes at each iteration of the column generation and represents the opposite of the estimated reduced cost of an arc. The elementarity constraint cannot be relaxed for a feasible solution, but it can be relaxed in the RMP, "temporarily", in order to speed up the column generation. Actually, compatibility graphs are generally sparse, unlike graphs of vehicle routing problems which are usually complete, and relaxed solutions may be elementary anyway.

5.1.2 Exploiting the length constraint

As we look for a path of length at most L, we can use this information to reinforce shortest path algorithms. We compute with Floyd-Warshall algorithm the distance function $d : V \times V \to \mathbb{N} \cup \{+\infty\}$ where $d(i, j)$ is the

shortest path between i and j, with respect to the **number** of arcs. We define the **extended neighborhood** $\gamma(i)$ of vertex i as the set of vertices that may appear in any path starting at the source of length at most L including i: $\gamma(i) := \{j \in V : d(s,i) + d(i,j) \leq L \text{ or } d(s,j) + d(j,i) \leq L\}$. Note that if $i \in \gamma(j)$ then $j \in \gamma(i)$. We define the **extended predecessors** $\gamma^-(i)$ of vertex i as the sets of vertices that can reach i in a path of length at most L: $\gamma^-(i) := \{j \in V : d(s,j) + d(j,i) \leq L\}$.

This distance function is used first to perform a **preprocessing** on the graph G by removing every arc (and vertex) that is too far from the source to be contained in a path of length L (the same as algorithm 2 in Section 4.2.1, without the condition on K). The sets of extended neighbors and predecessors also take part in the algorithms presented in this chapter.

5.1.3 Dynamic programs

Numerous approaches solving paths problems, including Desrochers' and Feillet', are based on dynamic programing and labeling algorithms following Held and Karp results [60]. Recall that $N^-(i)$ and $N^+(i)$ are the sets of predecessors and successors of $i \in V$. The optimal solution of EMPPLC can be computed via the recursive function:

$$f^*(S,i) = \min_{j \in N^-(i) \cap S} \{f^*(S \setminus \{i\}, j) + c_{ji}\} \tag{5.1}$$

where $f^*(S,i)$ is the minimal cost of a $(|S|-1,i)$-path, visiting all vertices of S and ending in $i \in S$. This dynamic program has a space complexity of $\mathcal{O}(|V|2^{|V|})$, a time complexity of $\mathcal{O}(|A|2^{|V|})$ and does not scale up to large instances.

Relaxations A common approach to overcome scaling issues is to relax the problem in order to deal with a smaller search space. Relaxing a minimization problem aims at quickly providing lower bounds. If we relax the constraint of elementarity, the search space is strongly reduced since the visited vertices are not remembered anymore. The problem to solve becomes the minimum path problem with length constraint, a special case of the **shortest path problem with resource constraints** and can be solved by a dynamic program running in $\mathcal{O}(L|A|)$:

$$f^*(l,i) = \min_{j \in N^-(i)} \{f^*(l-1,j) + c_{ji}\}$$

where $f^*(l,i)$ is the minimal cost of a (l,i)-path.

If, on top op that, we also relax the resource constraints, the problem falls to the **shortest path problem**. When the graph contains cycles of negative weight, the problem is unbounded. Otherwise the solution is a shortest path using at most $|V|-1$ arcs that can be found in $\mathcal{O}(|V||A|)$

by the Bellman-Ford algorithm. It is actually based on the same recursive formula, the length indexation being present to avoid infinite loops. In a directed acyclic graph, one can instead uses the following recursive formula of time complexity $\mathcal{O}(|A|)$:

$$f^*(i) = \min_{j \in N^-(i)} \{f^*(j) + c_{ji}\}$$

where $f^*(i)$ is the minimal cost of a path from s to i.

In this thesis, we study a more complex relaxation due to Baldacci *et al.* [13] called the NG-route relaxation. We present their algorithm and how we reinforce it for our problem in section 5.5.

Restrictions Another, and opposite, idea to reduce the search space in dynamic programming is to restrict the problem. More constrained problems will provide feasible solutions and thus upper bounds. We quickly introduce the color coding algorithm proposed by Alon *et al.* [4] and our contributions in Section 5.4, but a full analysis is provided in Chapter 6.

5.1.4 Heuristic, relaxation and exact approaches

When dealing with the EMPPLC, like for any problem, the objective is generally to find the optimal solution. Exact algorithms are designed for this, but as EMPPLC is NP-hard, they might take a long time to get it. On the other hand, heuristics are meant to quickly produce feasible solutions and upper bounds, while relaxations provide lower bounds. And sometimes, these non-exact algorithms are sufficient, in particular when the problem is embedded in a column generation (see Section 4.1.3).

We present four algorithms to address these three goals. The linear program of Section 5.2 can meet two of these objectives: optimality via the integer program and lower bounds via the linear relaxation. To compute feasible solutions, a local search heuristic is proposed in Section 5.3. We focus however on dynamic programming approaches, as the previous section showed how they can find any kind of solution (exact, relaxed or feasible). Sections 5.4 and 5.5 detail the NG-route relaxation and the color coding restriction, adapted from the literature to improve their performance for the EMPPLC. The color coding is fully analyzed in Chapter 6.

5.2 Time-stage formulation

We model the EMPPLC with the **time-stage formulation** (**TSF**), adapted from the formulation of Fox, Gavish and Graves [48] for the traveling salesman problem. A dummy sink vertex t is added, as well as an arc for each vertex to t with cost 0. $G' = (V' = V \cup \{t\}, A' = A \cup \{it \ \forall i \in V\})$.

Model TSF

$$\forall (ij) \in A',\ l \in \{1,...,L+1\}\colon\ y_{ij}^{l} = \begin{cases} 1 \text{ if } (ij) \text{ is the } l^{th} \text{ arc in the path} \\ 0 \text{ otherwise} \end{cases}$$

$$c^{*} = \min \sum_{l \in \{1,...,L+1\}} \sum_{(ij) \in A'} c_{ij} y_{ij}^{l} \tag{5.2}$$

$$s.t. \quad \sum_{j \in N^{+}(s)} y_{sj}^{1} = 1 \tag{5.3}$$

$$\sum_{l \in \{2,...,L+1\}} \sum_{i \in N^{-}(t)} y_{it}^{l} = 1 \tag{5.4}$$

$$\sum_{i \in N^{-}(j)} y_{ij}^{l} = \sum_{i \in N^{+}(j)} y_{ji}^{l+1} \qquad \forall j \in V', \forall l \in \{1,...,L\} \tag{5.5}$$

$$\sum_{(ij) \in A'} y_{ij}^{l} \leq 1 \qquad \forall l \in \{1,...,L+1\} \tag{5.6}$$

$$\sum_{l \in \{1,...,L+1\}} \sum_{j \in N^{+}(i)} y_{ij}^{l} \leq 1 \qquad \forall i \in V' \tag{5.7}$$

$$y_{ij}^{l} \in \{0,1\} \qquad \forall (ij) \in A', l \in \{1,...,L+1\} \tag{5.8}$$

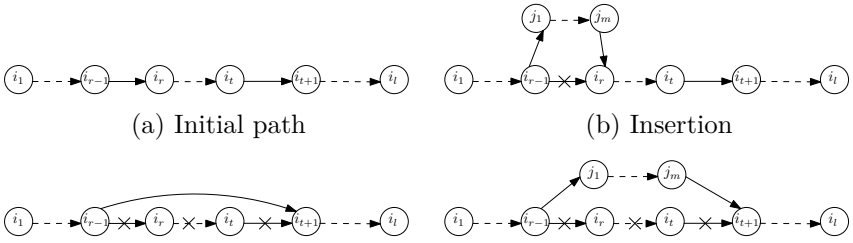
Constraints (5.3) and (5.4) ensure that the first chosen arc leaves the source and that the sink is reached. Constraints (5.5) are flow constraints guaranteeing that if a vertex is reached by the l^{th} arc then it is left with the $(l+1)^{th}$. Constraints (5.6) impose that only one arc is taken at stage l and constraints (5.7) forbid a vertex to be taken more than once.

The integer program TSF is used to compute the optimal solutions of the different EMPPLC instances. It is also solved within a time limit of 1 second, given the best lower and upper bounds **TSF-lb** and **TSF-ub**.

5.3 Local search heuristic

We developed a simple heuristic to quickly find feasible solutions. It is based on local search: the algorithm moves from the current solution to a better solution in its neighborhood. A first path is constructed with a random search.

Neighborhood. Given $p = (s, i_1, ..., i_l)$ a path of length at most L starting from s. A neighbor p' of p is obtained by applying one of the three following movements (see figure 5.1):



(a) Initial path (b) Insertion

(c) Suppression (d) Exchange

Figure 5.1 – Movements of the local search.
Dashed arrows represent subpaths, plain arrows are edges and crossed arrows are removed in the movement.

- `insertion` at position r.
  Given any elementary path $(j_1, ..., j_m)$ disjoint from p, if it is arc-disjoint from p and if $(i_{r-1}j_1) \in A$, $(j_m i_r) \in A$ and $l + m \leq L$ then $p' = (s, i_1, ..., i_{r-1}, j_1, ...j_m, i_r, ..., i_l)$ is a neighbor of p.

- `suppression` of $(i_r, ..., i_t) \subseteq V(p)$.
  If $(i_{r-1}i_{t+1}) \in A$, then $p' = (s, i_1, ..., i_{r-1}, i_{t+1}, ..., i_l)$ is a neighbor of p.

- `exchange` of $(i_r, ..., i_t) \subseteq V(p)$.
  Given any elementary path $(j_1, ..., j_m)$, if it is arc-disjoint from p and if $(i_{r-1}j_1) \in A$, $(j_m i_{t+1}) \in A$ and $l - (t - r) + m \leq L$ then $p' = (s, i_1, ..., i_{r-1}, j_1, ..., j_m, i_{t+1}, ..., i_l)$ is a neighbor of p.

We limit the number of inserted, removed or exchanged vertices to 3. The local search randomly browses the neighborhood of the current solution and moves to p' if $c_{p'} < c_p$.

Termination criteria. The local search could stop when the current solution has no improving neighbor and is a local minimum. As the criteria of improvement is given by a strict inequality, no solutions will be visited twice and the algorithm actually ends. However, as explained in Section 5.6.1, we rather return the minimum path found in one second (solution **LM**) or the first path of negative cost (solution **LF**). If no path of negative cost is found in the time limit, then LF is equal to LM.

5.4 The color coding restriction

Alon *et al.* [4] proposed in 1995 a randomized dynamic programming algorithm to find simple paths of a given length, called **color coding**. The idea is to randomly color the graph and then to remember visited colors instead of visited vertices in the dynamic program. When the available number of

colors is smaller than the number of vertices, the search space of the dynamic program is reduced, but the optimal path may not be found as several vertices share the same color. We quickly present the original algorithm and an overview of the improvements we propose, but our complete work on this algorithm is detailed in Chapter 6.

5.4.1 Description of the algorithm

Given $C \geq L$ colors and $S \subseteq V$ a set of vertices. A **coloring** of S is a tuple $c \in \{1, ..., C\}^{|S|}$. Let c_i be the color of vertex $i \in S$. A graph is colored when a color is assigned to each vertex of V. The color coding algorithm follows two phases:

1. randomly color each vertex[1] $i \in V \setminus \{s\}$ with a color $c_i \in \{1, 2, ..., C\}$ with a certain probability.
2. find a shortest colorful path from the source using at most L colors with dynamic programing. $f^*(\mathcal{C}, i)$ is the minimal cost of a path from s to i, using $|\mathcal{C}| - 1$ arcs and visiting vertices of each color of $\mathcal{C}$.

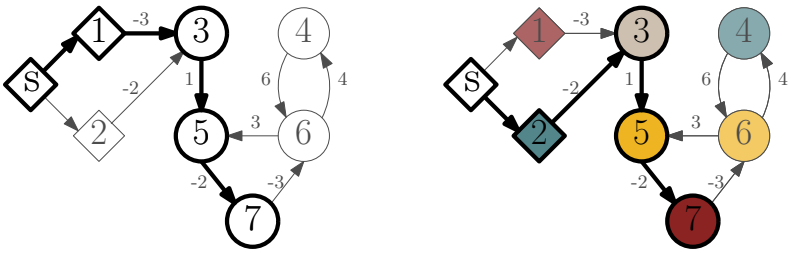
$$f^*(\mathcal{C}, i) = \min_{\substack{j \in N^-(i) \\ c_j \in \mathcal{C} \setminus \{c_i\}}} \{f^*(\mathcal{C} \setminus \{c_i\}, j) + c_{ji}\} \tag{5.9}$$

The memory of this dynamic program is reduced as there are C distinct vertex identifiers instead of $|V|$. The space complexity is indeed $\mathcal{O}(2^C|V|)$ and the time complexity $\mathcal{O}(|A|2^C)$. However its solution is only a feasible solution of EMPPLC, without guarantee on its quality. In order to get an optimal path p^* by color coding, step 1 must, by chance, color p^* with different colors. In this case, p^* would be colorful and be returned by step 2. Figure 5.2 illustrates the case where the coloring does not permit to find the optimal solution (path 1-3-5-7 of cost -4) and instead returns a solution of cost -3.

In the initial color coding algorithm, the color of each vertex is drawn according to a discrete uniform distribution. In this case, the probability that path p^* is colorful, denoted by ρ, is quite low. To reach a high probability of finding an optimal path, steps 1 and 2 are repeated several times The probability that exactly t trials are needed to make p^* colorful is $1 - (1 - \rho)^t$. To guarantee a failure probability of at most $\epsilon \in [0, 1]$, the color coding steps should be repeated at least $\frac{ln(\epsilon)}{ln(1-\rho)}$ times.

Instead of giving a fixed number of trials, our color coding implementation runs during 1 second and the best colorful path is returned, a solution denoted by **CM**. We also evaluate the quality of **CF**, the first solution of negative cost found by the color coding. If no path of negative cost is found, then CF is equal to CM.

[1] The source is not colored as it is taken in any solution.



(a) The optimal path (b) The color coding solution

Figure 5.2 – Example of a color coding trial: the best colorful path is not the optimal solution because 1 and 7 have the same color.

5.4.2 A different color coding framework

In addition to the standard color coding which colors vertices with a uniform distribution, we propose a new strategy that aims at spreading the colors in extended neighborhoods. To do so, it tries to make extended neighborhoods colorful by relying on three main ideas. First, a preprocessing step (called `la` ordering) applies a local search to create an ordering in which extended neighbors are gathered. Secondly, the vertices are colored by intervals of size C in this given order such that each interval is colorful: this is the `spread` coloring strategy. Finally, the ordering is shifted so that the intervals are made up of different extended neighbors each iteration. The whole algorithm is denoted by `lass` (for `la` ordering + `shifted-spread`) while the standard color coding is called `unif`. We refer the reader to Chapter 6 for every detail on our algorithm.

5.5 The NG-route relaxation

A few years ago, Baldacci *et al.* [13] proposed a new relaxation of the elementary shortest path problem with resource constraints called **NG-route**. The memory of a path is relaxed so that the search space of the dynamic program is reduced. In practice, a path constructed in the NG-route relaxation, called an NG-path, can forget that it went through some vertices and may visit them several times and be non elementary. However, if it turns out that the NG-path is elementary, then it is an optimal solution of the ESPPRC. We describe the adaptation of this algorithm for the EMPPLC special case.

Note that solving the EMPPLC with the NG-route relaxation in a column generation scheme can lead to the introduction of non elementary columns in the RMP. In this case, the column generation does not solve the linear relaxation of the master problem, but a relaxation of this linear problem. In the KEP context, the solution obtained with non-elementary

paths is thus an upper bound on z^*_{LP}, but it can be used similarly, for example to assess the quality of a feasible solution. Of course if only elementary paths were added by the NG-route relaxation, this upper bound is actually z^*_{LP}. As compatibility graphs are rather sparse, we expect that the elementarity will be often satisfied by NG-paths.

5.5.1 Description of the algorithm

In this relaxation, each vertex i has a "memory", also named **NG-set**, denoted by $\eta_i \subseteq V$ and such that $i \in \eta_i$. If an NG-path goes through i, it can remember only vertices of η_i. As it is true for each vertex, an NG-path only remembers the vertices appearing in every NG-set. The forward and backward dynamic programs constructing the NG-path of minimum cost with respect to these NG-sets are described below.

Forward dynamic program Each path $p = (s, i_1, ..., i_l)$ is associated with a set $\Pi(p) = \left\{ i_r : i_r \in \bigcap_{t=r+1}^{l} \eta_{i_t}, r = 1, ..., l-1 \right\} \bigcup \{i_l\}$ (see Figure 5.3).

A forward **NG-path** (Π, l, i), is a non necessarily elementary path $p = (s, i_1, ..., i_l = i)$ starting from s, ending in i, using l arcs and such that $\Pi = \Pi(p)$. Π represents the memory of p, since no vertex of Π can be used to extend p. p is constructed by adding i to a smaller NG-path that belongs to the set $\Psi^-(\Pi, l, i)$:

$$\Psi^-(\Pi, l, i) = \{ (\Pi', l-1, j) \text{ ng-paths s.t. : } j \in N^-(i), \Pi = (\Pi' \cap \eta_i) \cup \{i\}, \Pi' \subseteq \eta_j, j \in \Pi', i \notin \Pi' \}$$

$f^*(\Pi, l, i)$ is the minimal cost of an NG-path (Π, l, i) and can be computed with the following recursive formula:

$$f^*(\Pi, l, i) = \min_{(\Pi', l-1, j) \in \Psi^-(\Pi, l, i)} \{ f^*(\Pi', l-1, j) + c_{ji} \} \tag{5.10}$$

i	η_i
0	$\{0\}$
1	$\{0, 1\}$
2	$\{0, 1, 2\}$
3	$\{0, 1, 2, 3\}$
4	$\{1, 2, 4\}$

i in Π $(p = (0, 1, 2, 3, 4))$?		
no:	$0 \notin \eta_1 \cap \eta_2 \cap \eta_3 \cap \eta_4$	$\Rightarrow \Pi(p) = \{1, 2, 4\}$
yes:	$1 \in \eta_2 \cap \eta_3 \cap \eta_4$	
yes:	$2 \in \eta_3 \cap \eta_4$	
no:	$3 \notin \eta_4$	
yes:	by definition	

Figure 5.3 – Memory construction of a forward NG-path $p = (0, 1, 2, 3, 4)$

Backward dynamic program Each backward-path $p = (i_{L-l}, ..., i_L)$ is associated with $\Pi^{-1}(p) = \left\{ i_r : i_r \in \bigcap_{t=L-l}^{r-1} \eta_{i_t}, r = L-l+1, ..., L \right\} \bigcup \{i_{L-l}\}$ (see Figure 5.4).

A backward NG-path (Π, l, i) is a non necessarily elementary path $p = (i = i_{L-l}, ..., i_L)$ starting from i, using l arcs and such that $\Pi = \Pi^{-1}(p)$. An NG-path $p' = (j = i_{L-l+1}, ..., i_L)$ that can reach p by adding (at the beginning) the vertex $i = i_{L-l}$ belongs to the set Ψ:

$$\Psi(\Pi, l, i) = \{ (\Pi', l-1, j) \text{ backward ng-paths s.t. : } j \in N^+(i), \Pi = (\Pi' \cap \eta_i) \cup \{i\}, \Pi' \subseteq \eta_j, j \in \Pi', i \notin \Pi' \}$$

$b^*(\Pi, l, i)$ is the minimal cost of a backward ng-path (Π, l, i) and can be computed with the following recursive formula:

$$b^*(\Pi, l, i) = \min_{(\Pi', l-1, j) \in \Psi(\Pi, l, i)} \{ b^*(\Pi', l-1, j) + c_{ij} \} \tag{5.11}$$

i	η_i
9	$\{9\}$
8	$\{8\}$
7	$\{7\}$
6	$\{6, 7\}$
5	$\{5, 7\}$

i in Π $(p = (5,6,7,8,9))$?	
no:	$9 \notin \eta_8 \cap \eta_7 \cap \eta_6 \cap \eta_5$
no:	$8 \notin \eta_7 \cap \eta_6 \cap \eta_5$
yes:	$7 \in \eta_6 \cap \eta_5$
no:	$6 \notin \eta_5$
yes:	by definition

$\Rightarrow \Pi(p) = \{5, 7\}$

Figure 5.4 – Memory construction of a backward NG-path $p = (5, 6, 7, 8, 9)$

Filtering. States of a dynamic program can be pruned if they are proven to be suboptimal. This filtering makes dynamic programming approaches very efficient when bounds can be easily computed. Assume we have UB_p an upper bound on the EMPPLC solution value. Given any lower bound $\underline{b}(l, i)$ on the cost of a path starting from i and using at most l arcs, we can cut off the forward dynamic program every state (Π, l, i) such that $f^*(\Pi, l, i) + \underline{b}(L-l, i) \geq UB_p$. Indeed, the best NG-path that can be constructed from the NG-path of state (Π, l, i) costs at least $f^*(\Pi, l, i) + \underline{b}(L-l, i)$. If this cost is greater than the upper bound, this state is suboptimal. In the same way, given any lower bound $\underline{f}(l, i)$ on the cost of a path starting from s, ending in i and using at most l arcs, each state (Π, l, i) of the backward dynamic program such that $b^*(\Pi, l, i) + \underline{f}(L-l, i) \geq UB_p$ can be removed.

When the NG-route relaxation is used during the pricing step of a column generation algorithm, its aim is to find a path of negative cost. In this case, zero is a natural upper bound on the best NG-path to find and $UB_p = 0$.

The lower bounds $\underline{b}$ and $\underline{f}$ are called **completion bounds** and should be computed very fast. In the following, we explain how these bounds can be obtained in an iterative scheme of the NG-route relaxation.

5.5.2 NG-route configurations

As for the color coding, different configurations of the dynamic program are possible. The first type of improvement is a general technique used in dynamic programming, while the second one is specific to the EMPPLC as it uses the extended neighborhoods.

Decremental State-Space Relaxation. Pecin *et al.* [100] proposed to use the Decremental State-Space Relaxation (DSSR) technique of Righini and Salani's [106]. It is an iterative algorithm in which the search space is even more relaxed than in the pure NG-route relaxation. At each iteration k, each vertex i is associated with a set μ_i^k that takes the role of the NG-set η_i in the dynamic program. At the first iteration the subsets μ_i^0 are empty sets. When the solution p_k of iteration k is not elementary and does not respect the chosen criterion, vertices are added to the sets μ_i^k and a new iteration begins. There are three main possible criteria in the DSSR NG-route algorithm.

- predefined: original NG-sets η_i are computed and the DSSR continues until p_k is either elementary or a feasible NG-path with respect to these sets. Vertices are added to sets μ_i^k only if they belong to η_i.
- limited: the DSSR continues until p_k is elementary or the sizes of sets μ_i^k exceed a given limit.
- unlimited: the DSSR continues until p_k is elementary.

Filtering in DSSR. When applying a DSSR, we can alternate the forward and backward dynamic programs and use the information calculated at a previous iteration to compute the lower bounds needed to apply the filtering rule described above. We note $f_k^*(\Pi, l, i)$ and $b_k^*(\Pi, l, i)$ the values of the recursive formula at iteration k and define $\underline{f}_k(l, i) = \min_{l' \leq l, \Pi} f_k^*(\Pi, l, i)$ and $\underline{b}_k(l, i) = \min_{l' \leq l, \Pi} b_k^*(\Pi, l, i)$. Then, $\underline{f}_k(l, i)$ (resp. $\underline{b}_k(l, i)$) can be used as lower bounds in the next backward (resp. forward) dynamic program.

Choice of the NG-sets. Without descent or with a predefined one, NG-sets are the heart of this algorithm since they determine the quality of the solution as well as the computation efficiency. When η_i is empty for every vertex, there is absolutely no constraint on the elementarity of the path and the NG-route relaxation solves the SPPRC. When $\eta_i = V$ for every vertex,

the NG-route is not a relaxation anymore and the solution is necessarily elementary. Any other choice requires to make a decision about the composition of these sets and usually their construction is random. We propose to take into account the extended neighborhoods to construct NG-sets in order to increase the chance to obtain an elementary path without increasing too much the computation times.

It is sufficient for a vertex to "remember" in η_i only its extended predecessors since they are the only vertices that can appear in a path reaching i. However $\gamma^-(i)$ can still be too big to permit reasonable computation time and here again choices have to be made. We fix therefore a limit Λ to the size of an NG-set. When the EMPPLC is embedded in a column generation framework, vertices are associated with a dual value which usually represents the interest for the vertex to appear in the solution. We propose to sort according to this dual value the sets $\gamma^-(i)$ and to keep only the Λ first vertices in the ng-set η_i.

5.6 Experiments

5.6.1 Protocol

Experiments are conducted to profile the different algorithms for a column generation scheme. Algorithms configurations and instances choices are led by this context.

Choice of the instances. Experiments are conducted on several EMPPLC instances generated from the pricing step of KEP instances. Pools of patients and donors are created using an online[2] Saidman-based generator [114] with realistic parameters, leading to sparse graphs. The generation of the KBR benchmark is fully detailed in Section 3.3.1, but note that there are 27 different classes of instances. In particular, the number of patients varies between 50 and 250, $K = 3$ and $L \in \{4, 7, 13\}$. In this chapter, only one KEP instance in each class is considered. The exchange formulation is solved by column generation on these instances and EMPPLC instances are extracted from the first, last and middle iterations of the pricing problem. Note that the 54 instances generated from the first and middle iterations contain a solution of negative weight while the optimal value for the 27 last iterations is zero. The first 54 instances are grouped in E-KBR$^-$, the 27 others in E-KBR^0. The purpose of generating instances with such a procedure is to get dual values at different stages of the column generation.

Choice of the performance indicators. The quality of the solutions returned by the different algorithm is evaluated using its gap to the optimal

[2]available at http://www.dcs.gla.ac.uk/~jamest/kidney-webapp/#/generator

solution, as well as the number of solutions that are actually optimal. However, in a column generation framework it is not important to find optimal solution, but to find a solution with the same sign as the optimal solution. Indeed, if a path with a negative cost is found, optimal or not, it is a new column to add to the restricted master problem. Similarly, if a relaxation produces a solution of positive (or zero) cost, the optimality proof of the linear relaxation is done. Thus, we also compute the number of solutions having the same sign than the optimal solution.

Given an EMPPLC instance, whose optimal solution OPT costs c^*, we thereby compute three performance indicators for a solution x of value c:

- Gap to opt $\frac{|c-c^*|}{c^*}$
- Optimal found $c = c^*$
- Same sign $\begin{cases} c^* \geq 0 \text{ and } c \geq 0 \\ c^* < 0 \text{ and } c < 0 \end{cases}$

Note that the gap cannot be computed for instances of E-KBR^0 as their optimal value is 0.

Choice of the algorithm settings. In a column generation, methods providing feasible solutions are designed to quickly find new columns to add, *i.e.*, find a path with a negative reduced cost. For this reason, heuristics either stop when the first solution of negative cost is found, or run within a small time limit. On the contrary, relaxations must find good solutions to allow filtering and computation of good bounds. Therefore, the NG-route relaxation is not limited in time while the color coding and local search algorithms are set to return the first solution of negative cost and the best solution after 1 second of running time. Note that the color coding actually ends after at least one trial was completely executed, so the effective running time may exceed this time limit. The integer program is used to compute the optimal solution, but also best upper and lower bounds within 1 second. All in all, seven solution types are reported:

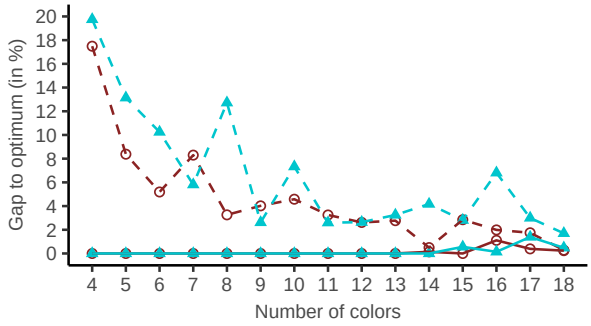
- 5 upper bounds
  - CF: first solution of color coding
  - CM: best solution of color coding in 1 second
  - LF: first solution of local search
  - LM: best solution of local search in 1 second
  - TSF-ub: best feasible solution of the integer program in 1 second
- 2 lower bounds
  - NG: best NG-route of the instance
  - TSF-lb: best lower bound of the integer program in 1 second

Note that solutions CF and LF are not reported for instances of E-KBR^0 as their optimal value is 0 so no feasible solution of negative cost can be found.

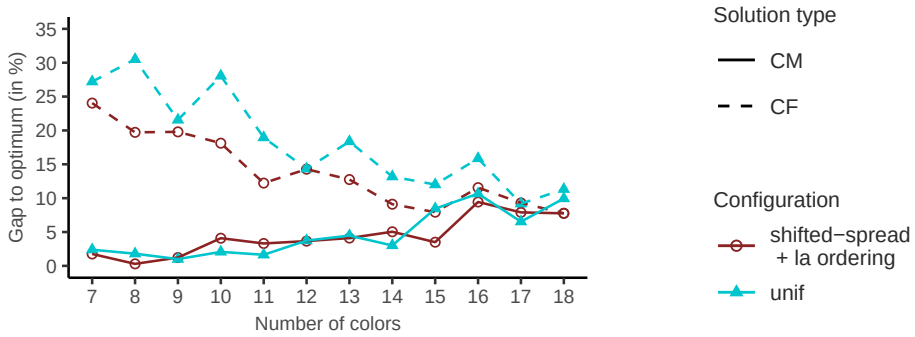
5.6.2 Preliminary results

As the NG-route and the color coding have different configurations and a customizable memory size, we have conducted preliminary experiments to compare the different configurations of color coding and NG-route. The "best" configurations are then kept for the global computational results, reported afterward.

(a) $L = 4$ (18 instances)



(b) $L = 7$ (18 instances)



(c) $L = 13$ (18 instances)

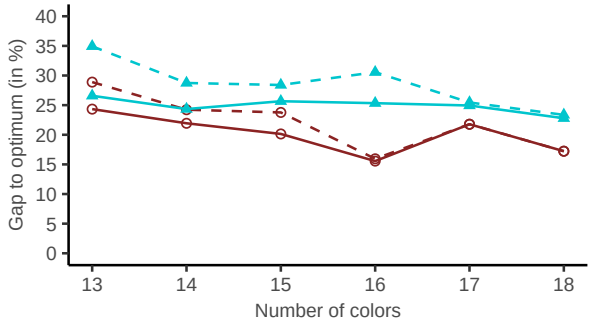


Figure 5.5 – Gap between CF or CM and OPT depending on the configuration and the number of colors, for the E-KBR$^-$ benchmark

Color coding. We conducted experiments on the two configurations of color coding (`unif` and `lass`) and different number of colors in order to identify the most promising algorithm to embed in our column generation scheme. The experimental results support the theoretical results of Chapter 6 as well as our expectations. In particular, we observe that our method `lass` outperforms the standard color coding algorithm.

Figures 5.5 show the gap $\frac{|c-c^*|}{c^*}$ between the solution OPT of value c^* and the solutions CF and CM for the E-KBR$^-$ benchmark. The 54 instances are grouped in three sets of 18 instances, according to the parameter L. The first negative solution CF for the `lass` strategy is almost always better than for the standard color coding `unif`. Its quality increases with the number of colors and this is due to the fact that a trial of color coding is more efficient with more colors, both for `lass` and `unif` configurations. However, an important condition for the color coding to return good solutions is to make many trials. As increasing C also increases the running time of one trial, it leads to poorer solutions in the same time limit, hence the growing gap for CM. When C is too big, the color coding can exceed the time limit (see Figure 5.6) because an iteration runs during more than one second. In this case the color coding actually stops after a single iteration and CF = CM. From this analysis, the best compromise seems to run the color coding with a `lass` configuration and a small number of colors (we take $C = L + 1$).

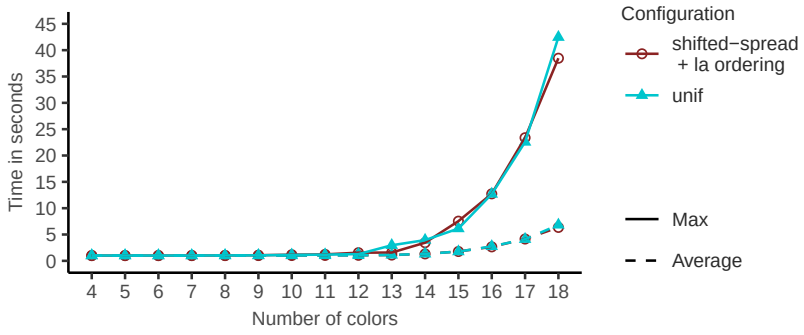


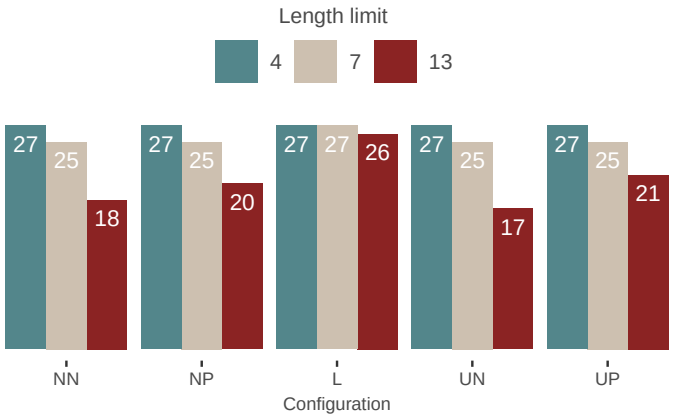
Figure 5.6 – Average and maximum color coding running times depending on the number of colors and the configuration on E-KBR$^-$ instances

NG-route. Five versions of the NG-route are implemented and tested, voluntarily omitting the unlimited DSSR as it provides no control on the computation time and memory space. Table 5.1 sums up the different version and how they construct the NG-set and which DSSR is applied, if any. We tested different size limits for the NG-set (5 to 13), but it appears that they make no difference on the solution quality. On the other hand, increasing this size limit deteriorates the computation time, in particular for configurations without descent. Thus, only results for a size of 5 are kept.

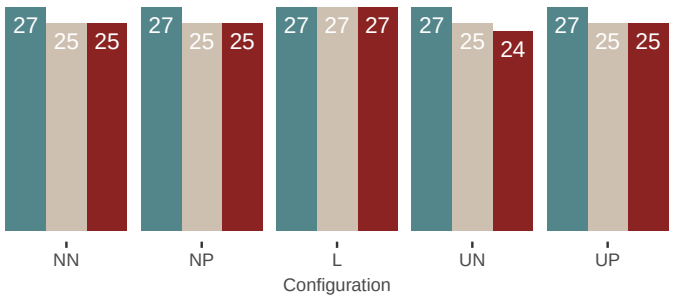
	NG-set creation	Descent
UN	**U**niform	**N**one
NN	**N**eighborhood	**N**one
L	-	**L**imited
UP	**U**niform	**P**redefined
NP	**N**eighborhood	**P**redefined

Table 5.1 – The five versions of the NG-route relaxation

Figures 5.7 illustrate the quality of the NG-route solution for the E-KBR benchmark. The 81 instances are grouped into three sets of 27 instances, according to the parameter L. Figure 5.7a shows the number of instances optimally solved by the NG-route relaxation, *i.e.*, the number of instances for which the NG-route returns an elementary NG-path, so the optimal



(a) Number of instances, out of 27, for which NG = OPT



(b) Number of instances, out of 27, for which NG and OPT have the same sign

Figure 5.7 – Quality of the NG-route solution depending on the configuration on E-KBR instances

solution. Figure 5.7b shows the number of instances for which the lower bound returned by the relaxation has the same sign than the optimal solution, a success criterion for the column generation algorithm. Both figures demonstrate the good performance of the limited DSSR compared to other configurations, even though all of them are quite efficient. Still, the limited DSSR is the only configuration that always returns a solution of the same sign than the optimal one and finds this optimal solution almost every time. The fact that the solution of the NG-route is often the optimal solution explains the fact that increasing the NG-set size is not interesting, as even when they are small the solution is elementary.

5.6.3 Comparisons of EMPPLC algorithms

Four algorithms were implemented and tested on the two benchmarks E-KBR$^-$ and E-KBR^0. Recall that for E-KBR^0, the gap cannot be computed and that CF and LF solutions do not make sense as no feasible solutions of negative weight can be found for these instances. Thus, for this benchmark, Table 5.2 shows only the number of instances for which:

- the solution has the same sign than the optimal one
- the solution is the optimal one

for solutions CM, LM, TSF-ub, NG and TSF-lb.

	Upper bounds			Lower bounds	
	CM	LM	TSF-ub	NG	TSF-lb
# instances with good sign	**27**	**27**	24	**27**	19
# instances with optimal solution	**27**	26	24	**27**	19

Table 5.2 – Quality of solutions for the 27 E-KBR^0 instances

Table 5.3 for E-KBL$^-$ is more complete as it also displays the gap with the optimal solution for instances having a solution of the same sign than the optimal one. This gap is therefore not computed on the same number of instances for all the solutions.

These results illustrate the dominance of the dynamic programs to compute both upper and lower bounds. The NG-route relaxation always finds the optimal solution, except once. No time limit was given to this algorithm, but we observe in Figure 5.4 that it actually runs very quickly. TSF provides poorer results with the same average running time, which, besides, is bounded by the time limit. Similarly, the color coding is very powerful as it

	Upper bounds					Lower bounds	
	CF	CM	LF	LM	TSF-ub	NG	TSF-lb
# instances with good sign	**54**	**54**	41	41	51	**54**	40
Average gap on such instances (%)	17.4	**7.4**	75.8	25.2	15.2	**0.01**	**0**
# instances with optimal solution	18	**40**	1	17	**40**	**53**	40

Table 5.3 – Quality of solutions for the 54 E-KBR$^-$ instances

finds the optimal solution for 67 instances out of 81. Even when it does not succeed to find the optimum, the gap is the smallest among every feasible solutions. On the contrary, the local search sometimes (for 13 instances) fails to find a negative solution when there is one.

Most importantly, the color coding and the NG-route always return a solution which has the same sign than the optimal solution in a small amount of time. Thus, these algorithms are really suitable for the pricing step of a column generation scheme.

	CM**	LM**	TSF*	NG
Average	1.17	1.05	0.44	0.43
Maximum	2.39	3.05	1.51	2.76

Table 5.4 – Average and maximum running times to get solutions

* Time limit of 1 second

** Soft time limit of 1 second (checked only between iterations)

5.7 Conclusion: solving the EMPPLC

In this thesis, the elementary minimum path problem with length constraint has to be solved many times in the pricing step of our column generation framework. We adapted algorithms from the literature of shortest paths problem to better fit the specificity of our problem and designed an experi-

mental protocol to assess their quality. The analysis of these experiments[3] led our choices for the complete column generation algorithm presented in Section 4.5. In particular, the color coding, whose performance is more evaluated in Chapter 6, and the NG-route relaxation turn out to be the most promising algorithms in this context. We believe even better results can be obtained by investigating more deeply these algorithms. In particular, an adaptation of the different techniques proposed by Pecin *et al.* [99], including memory cuts, would probably strengthen our implementation of the NG-route relaxation.

[3]Complete results available on my webpage: https://pagesperso.g-scop.grenoble-inp.fr/~pansartl/.

Chapter 6

New randomized strategies for the color coding algorithm

The elementary minimum path problem with length constraint (EMPPLC) studied in Chapter 5 can be solved with color coding as briefly explained in Section 5.4. The key idea of this method is to randomly color the vertices of the graph with $C \geq L$ colors, then to seek a colorful path, *i.e.*, a path where all vertices have distinct colors. On the one hand, the search for a colorful path is significantly more efficient than the search for a simple path as there are C distinct vertex identifiers instead of $|V|$. On the other hand, the random coloring might be unlucky and give the same color to at least two vertices of any optimal path. The procedure is therefore repeated to ensure that the path is found with a high enough probability.

This technique is actually more general as it can be applied to find any subgraph. The first section presents the algorithm and our motivations for the contributions proposed in the next sections.

6.1 The color coding algorithm

The color coding technique is a two-phase heuristic for optimization problems: the first phase colors the graph, while the second phase uses an algorithm, usually a dynamic program, to find an optimal solution with distinct colors. The two phases are repeated and each trial can return a feasible solution, but many trials might be necessary to find an optimal one[1]. We will not study the second phase in this chapter as it is different for every

[1] When considering a decision problem, the second phase only looks for a feasible solution and each trial either returns such a solution (and the algorithm stops) or no solution at all.

problem, and we refer the reader to Section 5.4 for details on this phase for the EMPPLC.

6.1.1 Motivations

The second phase of the color coding algorithm misses optimal solutions if each of them contains two vertices of the same color. Both from theoretical and practical perspectives, in all prior works related to color coding, the graph is colored according to a discrete uniform distribution: all the colors are equally likely to be selected for each vertex. To the best of our knowledge, only one paper addresses the choice of the coloring, and for a derandomization purpose [69]. Yet the probability distribution used to color the graph is an efficiency factor. Although it does not influence the runtime complexity of the dynamic program, it can highly reduce the number of its executions by increasing the chance for an optimal solution to be colorful.

Our key idea is to bias the coloring in the first phase to increase the probability that an optimal solution has distinct colors and thus reduce the number of calls to the dynamic program. To do so, we propose in Section 6.2 new random but non uniform coloring strategies, which color vertices in a given order. This order is created by a preprocessing taking advantage of the graph structure and presented in Section 6.3. One of the coloring strategy provides a guaranteed improvement over the uniform coloring. When it is associated with a new color coding framework described in Section 6.4, it even guarantees to find an optimal solution with only C calls to the dynamic program for orderings with a particular structural property related to the bandwidth.

6.1.2 Literature review

The color coding approach was proposed in 1995 by Alon *et al.* [4]. It has been studied in several papers in which the graph is colored either with a discrete uniform distribution or with a derandomized procedure and no other coloring strategy has been proposed, except by Kneis *et al.* [69]. This is surprising as this algorithm is studied in various domains.

When combined with derandomization [92], color coding can be used to design deterministic, fixed-parameter tractable (FPT) algorithms. A number of state-of-the-art FPT algorithms rely on color coding for several fundamental problems related to packing [70], matching [45, 70], vertex cover [69] or L-path [4, 122]. Several randomized algorithmic frameworks [130] such as randomized divide-and-conquer [26], the random separation technique [23] or parallel fixed-parameter algorithms [14] are also based on color coding.

This technique has also been successfully used in practice, and in particular in the bioinformatics field (see *e.g.* [3, 41, 84, 117, 119]). A typical example is the detection of signaling pathways in protein interaction

networks [61]. The problem is also cast as an elementary minimum path problem with length constraint and practical experiments regarding the implementation of color coding is reported by Hüffner *et al.* [61]. In particular, they analyze the best trade-off between the number of colors (which controls the number of trials required) and the complexity of the dynamic programming step (the runtime of one trial).

Another relevant study for our application is the work of Borndörfer *et al.* [22] which uses the color coding to solve a path problem, encountered as a pricing subproblem in the context of line planning in public transport. Actually, exponential integer programming formulations based on paths are common in the field of transport and planning. For these formulations, the color coding turns out to be of major interest to solve the pricing problems. Surprisingly though, this algorithm has received little attention in this context and, to the best of our knowledge, only [22] uses it in such a situation.

6.1.3 Description of the algorithm

In the following, let $G = (V, E)$ be a simple graph, directed or not, and $n = |V|$. The color coding method is meant to find any subgraph $H = (V_H, E_H)$ with $L = |V_H|$ in G. The set V is colored with C colors and if by chance the vertices of H all have distinct colors then the problem is solved.

A **coloring** of a subset S of vertices is a tuple $c \in \{1, ..., C\}^{|S|}$. There are $C^{|S|}$ colorings of S. Let c_i bet the color of vertex $i \in S$. A coloring c is said to be **colorful** for the subset $S \subseteq V$, if $c_i \neq c_j \; \forall i \neq j \in S$. A graph is colored when a color is assigned to each vertex of V. It is a randomized procedure and the color assigned to a vertex follows a probability distribution. Let $C_i \in \{1, ..., C\}$ be the random variable giving the color of vertex i. A **coloring strategy** defines $\mathbb{P}(C_i = c)$, $\forall i \in V$ and $\forall c \in \{1, ..., C\}$.

The standard version of the color coding algorithm draws each color according to a discrete uniform distribution. In this case, denoted by `unif`, the probability for a vertex to get a color is the same for each color and each vertex: $\forall i \in V, \; \forall c \in \{1, ..., C\}, \mathbb{P}(C_i = c) = \frac{1}{C}$. The probability that H is colorful can be easily computed and was reported by Alon *et al.* [4] in the case $L = C$.

Property 6.1

If the graph is colored following the `unif` *strategy, then* $\mathbb{P}(H \text{ is colorful}) = \frac{C!}{(C-L)!C^L}$

Proof. The probability that H is colorful is the probability that a colorful colorings was drawn among the C^L possible colorings of H. As a colorful coloring is a L-permutation of C colors, there are $\frac{C!}{(C-L)!}$ many of them. Every coloring being equally likely to happen, we have: $\mathbb{P}(H \text{ is colorful}) = \frac{\frac{C!}{(C-L)!}}{C^L} = \frac{C!}{(C-L)!C^L}$ □

This probability does not depend on H, nor on the structure of the graph or the problem and the standard color coding algorithm assigns a color to each vertex without regard to the other vertices's colors. But if we could identify vertices that are likely to belong together to a solution, extended neighbors for instance, it should be more efficient to assign distinct colors to these vertices. That is the reason why we generalize the definition of extended neighborhood, use them to order vertices and color the graph in this given order.

Extended neighborhoods. We define the **extended neighborhood** $\gamma(i)$ of vertex i as the set of vertices that may appear in a subgraph H including i. It is a generalization of the definition given in Section 5.1.2 for the EMPPLC. In this case, $\gamma(i)$ is the set of vertices that may appear in any path starting at the source of length at most L including i: $\gamma(i) := \{j \in V : d(s,i) + d(i,j) \leq L \text{ or } d(s,j) + d(j,i) \leq L\}$. Recall that $d(i,j)$ is the distance between vertices i and j.

Ordering preprocessing. An **ordering**, or a **coloring sequence**, x is a permutation of the set $\{0, ..., n-1\}$. $\forall i \in V$, let x_i be the position of vertex i in the ordering and $X_i \in \{0, ..., n-1\}$ the associated random variable. An **ordering strategy** defines a probability distribution for the variable $X = (X_1, ..., X_n)$. When the random variable X follows a discrete uniform distribution each coloring sequence has the same probability to occur than the others.

Coloring step. A coloring sequence x has been already drawn in the preprocessing, *i.e.*, $X_i = x_i \ \forall i \in V$. We propose to make the choice of a color for each vertex i dependent on the previously colored vertices, $\eta(i) = \{j \in V : X_j < X_i\}$. Note that the ordering starts at position 0, so $|\eta(i)| = X_i$, meaning that when i has to be colored, X_i vertices are already colored. We now define a coloring strategy by $\mathbb{P}(C_i = c | C_j \ \forall j \in \eta(i))$.

6.2 Coloring ordered vertices

Our objective is to make an optimal solution colorful, but of course, we cannot guarantee in general to make such a coloring in one trial. Several trials will still be necessary to find an optimal solution, so each trial must be different from the others and the coloring procedure randomized. Many probability distributions can be chosen to color the graph, and our idea is to force the colors to be equally spread in the graph.

6.2.1 Spreading colors

To spread the colors equally, the probability that a vertex takes a color must depend on the frequency of each color taken by previously colored vertices. The more a color is used, the less likely it will be drawn. Let F_c^i denote the random variable counting the number of vertices colored with c before i: $F_c^i \in \{0, ..., X_i\}$ is such that $F_c^i = \big|\{j \in \eta(i) | C_j = c\}\big|$. Note that $\sum_{c=1}^{C} F_c^i = X_i =$ the number of vertices already colored.

A first idea is to draw a color c with a probability inversely proportional to the gap between its frequency and the number of previously colored vertices. In this case, the probability does not depend on the other colors.

Definition 6.1

The following probability distribution defines a coloring strategy:

$$\mathbb{P}\left(C_i = c | X_i = x_i, F_c^i = f_c^i\right) = \begin{cases} \dfrac{x_i - f_c^i}{x_i(C-1)} & \text{if } x_i \neq 0 \\ \dfrac{1}{C} & \text{otherwise} \end{cases} \tag{6.1}$$

Note that $\sum_{c=1}^{C} \frac{x_i - f_c^i}{x_i(C-1)} = 1$ so (6.1) is a well-defined probability distribution. When the number of colored vertices is high, all the probabilities are quite the same. So rather than comparing the frequency of c with the number of colored vertices, another idea is to use the highest frequency $f_{max}^i = \max_{c=1,...,C} f_c^i$.

Definition 6.2 spread

The coloring strategy **spread** *is defined by the probability distribution:*

$$\mathbb{P}\left(C_i = c | X_i = x_i, F_1^i = f_1^i, ..., F_C^i = f_C^i\right) = \begin{cases} \dfrac{f_{max}^i - f_c^i}{C \times f_{max}^i - x_i} & \text{if } C \times f_{max}^i \neq x_i \\ \dfrac{1}{C} & \text{otherwise} \end{cases} \tag{6.2}$$

Note that $\sum_{c=1}^{C} \frac{f_{max}^i - f_c^i}{C \times f_{max}^i - x_i} = 1$ so (6.2) is a well-defined probability distribution. Note that if c is a most used color (there may be several most used colors), it cannot be selected for vertex i. The second case occurs only when all the colors are equally used: $C \times f_{max}^i = x_i \Leftrightarrow f_c^i = \frac{x_i}{C}\ \forall c \in [\![1; C]\!]$. In this case, it is relevant to choose the color using a discrete uniform distribution as no color is less used than another.

Using extended neighborhoods. When choosing a color for a vertex, the probability distribution depends on the colors taken by all the previously colored vertices. However it does not matter that vertex i takes the same color than a vertex which cannot belong to a solution with i. Thus, it should be advantageous to compute color frequencies only among the previously colored extended neighbors of i. It seems yet even more advantageous to keep the `spread` strategy as it is and to rather base the ordering preprocessing on these extended neighborhoods. First, because the extended neighborhoods do not change from a color coding trial to another, so it is preferable to use them in the preprocessing as it is also common to every trials. Such an ordering is presented in Section 6.3. Secondly, the `spread` strategy turns out to have strong properties which prove that it is not more costly than a uniform coloring, as it does not require to indeed compute the color frequencies. The next section is dedicated to the study of this strategy.

6.2.2 Strategy `spread`: coloring by intervals

For sake of simplicity assume in this section that $n = qC$ with $q \in \mathbb{N}$ (the following results directly adapt when $n = qC + r$). The `spread` strategy actually colors the ordered vertices by intervals of size C so that each interval is colorful. We divide the coloring sequence $[\![0; n-1]\!]$ into q intervals $I_1, I_2, ..., I_q$ of size C: $I_i = [\![(i-1)C; iC - 1]\!]$ and $I(X_i)$ denote the interval containing the position of i, *i.e.*, $X_i \in I_k \Leftrightarrow I(X_i) = I_k$. By slightly abusing the notation we also write $i \in I_k$. We then observe that two vertices in

the same interval cannot have the same color and that the coloring of two intervals is independent.

Property 6.2

If the coloring step follows the `spread` *strategy, then* $\forall i, j \in V$ $i \neq j$*:*

$$\mathbb{P}\left(C_i \neq C_j | X_i, X_j\right) = \begin{cases} 1 & \text{if } I(X_i) = I(X_j) \\ \dfrac{C-1}{C} & \text{otherwise} \end{cases}$$

This means that the `spread` strategy simply draws for each interval I a permutation of $\{1, ..., C\}$ chosen with a uniform probability over all the permutations.

Probability that H is colorful. Recall that we want to color G such that V_H is colorful. The probability that this event occurs is an indicator of performance for a coloring strategy. In the `unif` case, $\mathbb{P}\left(H \text{ is colorful}\right)$ is easily computed and does not depend on an ordering. The analysis of the `spread` strategy is detailed below.

From the property 6.2, we know that vertices in the same interval have distinct colors, but in general the vertices of H are likely to be spread in several intervals. Let $V_H = \{v_1, ..., v_L\}$ denote the L vertices of H and thus $\mathbb{P}\left(H \text{ is colorful}\right) = \mathbb{P}\left(C_{v_1} \neq ... \neq C_{v_L}\right)$.

Let Y_k be the random variable counting the number of vertices of H in the k^{th} interval I_k. Y_k is calculated from the random variables X_i as follows: $Y_k = \left|\{i \in V_H : X_i \in I_k\}\right|$. Recall that q is the number of intervals ($n = qC$). A realization x of the random variable X can be used to compute $y \in \mathcal{Y} = \left\{(y_1, ..., y_q) \in [\![0, L]\!]^q : \sum_{k=1}^{q} y_k = L\right\}$. By the law of total probability, we have:

$$\mathbb{P}\left(C_{v_1} \neq ... \neq C_{v_L}\right) = \sum_{y \in \mathcal{Y}} \mathbb{P}\left(C_{v_1} \neq ... \neq C_{v_L}, Y = y\right) \tag{6.3}$$

and

$$\begin{aligned} \mathbb{P}\left(C_{v_1} \neq ... \neq C_{v_L}, Y = y\right) = \mathbb{P}\left(Y_1 = y_1, ..., Y_q = y_q\right) \\ \times \mathbb{P}\left(C_{v_1} \neq ... \neq C_{v_L} | Y_1 = y_1, ..., Y_q = y_q\right) \end{aligned} \tag{6.4}$$

Let $y \in \mathcal{Y}$ be a repartition of the vertices of H in the q intervals. The first term of the product (6.4) is the probability that $Y = y$. With a uniform ordering strategy, it is:

$$\mathbb{P}\left(Y_1 = y_1, ..., Y_q = y_q\right) = \frac{\prod_{k=1}^{q} \binom{C}{y_k}}{\binom{n}{L}}$$

The second term of the product (6.4) is the probability that H is colorful knowing the repartition is fixed to y.

$$\mathbb{P}\left(C_{v_1} \neq ... \neq C_{v_L} | Y_1 = y_1, ..., Y_q = y_q\right) = \frac{\binom{C}{y_1} \times \binom{C-y_1}{y_2} \times \binom{C-(y_1+y_2)}{y_3} \times ... \times \binom{C-(y_1+...+y_{q-1})}{y_q}}{\prod_{k=1}^{q} \binom{C}{y_k}} \tag{6.5}$$

Best and worst cases. The best case for this coloring strategy is when all the vertices of H are in the same interval, as H is then colorful with a probability of 1. On the contrary, the worst case for this strategy is when all the vertices are ordered in different intervals. We denote this event S. Namely, $S := \{I(X_{v_i}) \neq I(X_{v_j}) \ \forall v_i, v_j \in V_H\}$. We show that in this case the `spread` strategy is equivalent to the `unif` strategy.

Property 6.3

$\mathbb{P}\left(C_{v_1} \neq ... \neq C_{v_L} | S\right) = \frac{C!}{(C-L)!C^L}$

Proof. The event S can be defined as "exactly L intervals I_k have $y_k = 1$". Using (6.3), we have:

$$\mathbb{P}\left(C_{v_1} \neq ... \neq C_{v_L} | S\right) = \frac{\binom{C}{1} \times \binom{C-1}{1} \times \binom{C-2}{1} \times ... \times \binom{C-(L-1)}{1}}{\prod_{k=1}^{L} \binom{C}{1}} = \frac{C!}{(C-L)!C^L}$$

□

Thus, the new strategy always surpasses the original one. Indeed, the chance for H to be colorful is always greater or equal in the `spread` case than in the `unif` case. However, as n increases, S is more likely and `spread` is more and more equivalent to `unif`. This convergence is yet slower as n grows. Figure 6.1 shows $\mathbb{P}(S)$ depending on n for various L.

Property 6.4

With a uniform ordering, $\mathbb{P}(S) \underset{n \to +\infty}{\longrightarrow} 1$

Proof. Let $\mathcal{U} = \left\{(y_1, ..., y_q) \in \{0,1\}^q : \sum_{k=1}^{q} y_k = L\right\}$ be the set of realizations y such that S happens. $|\mathcal{U}| = \binom{q}{L} = \binom{\frac{n}{C}}{L}$. Every tuple (y) of $\mathcal{U}$ is equally

likely to happen with probability $\frac{\prod_{k=1}^{L}\binom{C}{1}}{\binom{n}{L}} = \frac{C^L}{\binom{n}{L}}$.

$$\begin{aligned}
\mathbb{P}(S) &= \sum_{y\in\mathcal{U}} \mathbb{P}(Y_1 = y_1, ..., Y_q = y_q) = \sum_{y\in\mathcal{U}} \frac{C^L}{\binom{n}{L}} = \binom{\frac{n}{C}}{L}\frac{C^L}{\binom{n}{L}} \\
&= \frac{\frac{n}{C}\times(\frac{n}{C}-1)\times ...\times(\frac{n}{C}-L+1)}{L!} \times \frac{C^L\times L!}{n\times(n-1)\times ...\times(n-L+1)} \\
&= \frac{n\times(n-C)\times ...\times(n-C(L-1))}{L!\times C^L} \times \frac{C^L\times L!}{n\times(n-1)\times ...\times(n-L+1)} \\
&= \left(\frac{n-C}{n-1}\right)\times\left(\frac{n-2C}{n-2}\right)\times ...\times\left(\frac{n-C(L-1)}{n-(L-1)}\right)
\end{aligned}$$

Each term of this product converges to 1 as n tends to $+\infty$, so the product converges to 1. □

Property 6.5

The derivative $\frac{\partial\mathbb{P}(S)}{\partial n}$ is in $O\left(\frac{1}{n^2}\right)$.

Proof. $\mathbb{P}(S) = \prod_{i=1}^{L-1}\frac{(n-iC)}{(n-i)} = \frac{\prod_{i=1}^{L-1}(n-iC)}{\prod_{i=1}^{L-1}(n-i)}$

$\mathbb{P}(S)$ is the quotient of two polynomials of the same degree $L-1$, so the result is a quotient with one degree less in the numerator. Thus $\mathbb{P}(S) = \frac{u}{v}$ with u a polynomial of degree at most $L-2$ and v is still a polynomial of degree $L-1$. The derivative $\frac{\partial\mathbb{P}(S)}{\partial n}$ is a polynomial quotient with a polynomial of degree at most $2L-4$ as numerator and a polynomial of degree $2L-2$ as denominator. Therefore $\frac{\partial\mathbb{P}(S)}{\partial n}$ is in $O\left(\frac{1}{n^2}\right)$. □

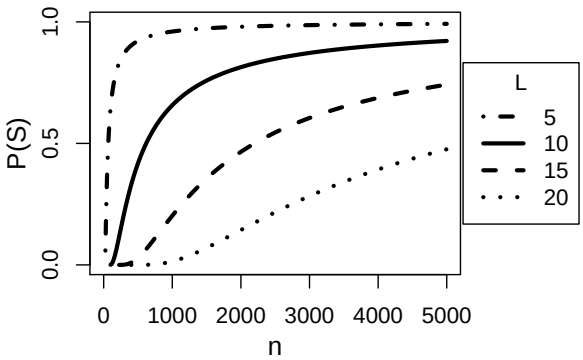


Figure 6.1 – $\mathbb{P}(S)$ with $C = L$

The `spread` strategy is at least as good as the `unif` strategy. For small n, the improvement is significant but the gap between the two strategies follows a decreasing function in $O(\frac{1}{n^2})$. As an example, let $L = C = 10$: when $n = 30$, the probability that H is colorful increases more than fivefold, and when $n = 100$ the factor is about 1.6. A major property of our coloring strategy is that some subsets of vertices (the intervals I_k) are colorful. Thus, if the vertices of H could be gathered in a single interval, then the `spread` strategy would always make H colorful. In the next section, we propose to order vertices by taking advantage of the graph structure to address this goal.

6.3 Ordering vertices to benefit from coloring phase

The `spread` coloring strategy is very efficient when the coloring sequence drawn in the ordering preprocessing puts the vertices of H in the smallest number of intervals. Even more, an ordering where the vertices of H belong to a single interval ensures a probability of 1 that H is colorful. This ideal ordering cannot be easily found but we can define a strategy that orders close to each other two vertices that may belong to H. We propose a new strategy dedicated to this purpose when H is connected.

Our first idea is to control the maximum difference between the position of two extended neighbors, defined as $\Delta = \max_{\substack{i \in V \\ j \in \gamma(i)}} |x_i - x_j|$. V_H is contained in a coloring subsequence of size at most Δ. To create an ordering that minimizes Δ, we solve the following optimization problem:

$$\begin{aligned} \Delta^* = \min\ & \Delta = \max_{\substack{i \in V \\ j \in \gamma(i)}} |x_i - x_j| \\ & x_i \neq x_j && \forall i \neq j \in V \\ & x_i \in \{0, ..., n-1\} && \forall i \in V \end{aligned} \tag{6.6}$$

This problem is equivalent to the graph bandwidth problem in an auxiliary graph $G' = (V, E')$ with the same set of vertices V but which contains an edge (ij) if i and j are extended neighbors. The **graph bandwidth problem** on G' aims at labeling the vertices of G' with distinct integers such that the maximum difference between the label of two adjacent vertices is minimized. In our case the label of a vertex i is its position X_i and solving the bandwidth problem provides a coloring sequence x minimizing $\Delta = \max_{(ij) \in E'} |x_i - x_j| = \max_{\substack{i \in V \\ j \in \gamma(i)}} |x_i - x_j|$.

The optimal solution Δ^* of (6.6) is called the **bandwidth** of G', denoted by $\varphi(G')$. Finding the bandwidth of a graph is an NP-hard problem [97] for

which various algorithms exist, as the widely used Cuthill-McKee heuristic and its reversed version [33, 54]. Many of them are heuristics but some exact approaches are able to solve instances of reasonable size (250 nodes). We refer the reader to [126] for a comprehensive review.

The graph bandwidth problem focuses only on the maximum difference between the positions of two extended neighbors and does not control the average difference between them. However, it could be efficient to move the positions of two extended neighbors away from each other, at the risk of increasing Δ, if that means many other extended neighbors are contained in a smaller coloring subsequence. We thus propose to solve a slightly different problem that minimizes the sum of all differences, that is $\delta = \sum_{(ij)\in E'} |x_i - x_j| = \sum_{\substack{i\in V \\ j\in\gamma(i)}} |x_i - x_j|$. This problem is known as the (minimum) linear arrangement problem [2] that was proven to be NP-complete [49].

$$\delta^* = \min\ \delta = \sum_{(ij)\in E'} |x_i - x_j|$$

$$x_i \neq x_j \qquad \forall i \neq j \in V$$

$$x_i \in \{0, ..., n-1\} \qquad \forall i \in V$$

In this case, Δ can be big, but in average the difference between the positions of two extended neighbors might be small. We therefore define an ordering strategy based on this linear assignment problem, called **`la ordering`**, which constructs an ordering aimed at minimizing δ. We use a simple local search approach swapping the vertices in the sequence. This strategy gathers extended neighbors in the coloring sequence and its quality depends on its running time as the local search is an anytime algorithm that provides an ordering whenever it is asked. The longer it runs, the better the ordering, *i.e.*, the smaller δ. Note that the solution to the linear arrangement problem does not have to be optimal, even if its quality has an impact on the efficiency of the coloring. The running time is determined by the user depending on the algorithm application and the time allocated to the dynamic program of the second phase.

Even when the positions of V_H are close enough, `la` ordering cannot guarantee that they belong to the same interval I_k. Indeed, the size of the position sequence of V_H may be smaller than C while being split into two intervals. In this case and with a standard color coding, the `spread` coloring strategy would not find H with a probability of 1. This motivates the new color coding technique introduced in the next section.

6.4 Shifted color coding

Given a coloring sequence x and a subgraph H, we can compute the size Δ_H of the coloring subsequence containing all the vertices of H:

$\Delta_H := \max_{\substack{i \in V_H \\ j \in \gamma(i)}} |x_i - x_j|$. Recall that the `spread` coloring strategy colors the vertices of G by intervals of size C. However, even if Δ_H is smaller than C, the coloring subsequence of H can straddle two coloring intervals and H can be non colorful.

We introduce a new coloring strategy, called `shifted-spread` (Algorithm 5), making the color coding more expensive (C dynamic program calls by trial) but stronger. It applies C times the `spread` coloring while shifting the coloring sequence between each iteration. Thus, if $\Delta_H \leq C$, at least one coloring sequence will put the vertices of H in a single coloring interval I_k and H will be colorful. As we do not know H, we cannot compute Δ_H, but the algorithm guarantees that when $\Delta \leq C$ only one trial of the color coding algorithm, with C calls to the dynamic program, is required to find H.

Algorithm 5 `shifted-spread`

Input: #`trials`, `tmax`
  if $\Delta \leq C$ **then** #`trials` $\leftarrow 1$
  for 1...#`trials` **do**
    apply a `shifted-spread` coloring:
    for $k : 1...C$ **do**
      - color: draw C_i $\forall i \in V$ using a `spread` coloring
      - solve: apply an algorithm that finds H if colorful
      - shift the ordering: $x = (x_{1+k}, ..., x_n, x_1, ..., x_k)$

Property 6.6

With a `shifted-spread` coloring strategy, if $\Delta \leq C$ then $\mathbb{P}(H \text{ is colorful}) = 1$

We saw in Section 6.2 that the `spread` strategy gives a significant improvement for small n which does not scale with n. When combined with a `shifted-spread` ordering strategy, this improvement now depends on Δ rather than n. Indeed, the probability that H is colorful if G is colored with a `shifted-spread` strategy is at least as good as the probability that H is colorful in a graph with Δ vertices colored with a `spread` strategy.

6.5 Experimental results

We present experimental results[2] for the color coding algorithm applied to a variant of EMPPLC: the minimum-weighted L-path problem, *i.e.*, the

[2] Complete results available on my webpage: https://pagesperso.g-scop.grenoble-inp.fr/~pansartl/.

problem of finding a simple path of **exactly** L vertices of minimum weight in a weighted directed graph.

6.5.1 Protocol

We want to measure how the coloring of the first phase impacts the probability to find any optimal path in the second phase. This happens each time an optimal path is colorful at the end of the first phase and does not depend on the second phase. As the performances of the dynamic program are not affected by our strategies, our experiments focus only on the first phase. Thus, we do not run the dynamic program and assume that the optimal paths are known[3], constituting the set of paths $\mathcal{P}$. To assert the effectiveness of the various coloring strategies, we only check if one path among $\mathcal{P}$ is colorful. We execute 150 000 trials of each strategy and compute the number of times one of the optimal paths is colorful out of the 150 000 trials. This frequency estimates the probability of the first phase to be successful.

Choice of the instances. We test our algorithms on two benchmarks of graphs and with L in $\{10, 15, 20\}$.

The first benchmark is E-KB, which contains graphs coming from the pricing step of our column generation framework solving the KEP (see Section 5.6.1). Initial KEP instances are created using a Saidman-based generator [114] available at `http://www.dcs.gla.ac.uk/~jamest/kidney-webapp/#/generator` using realistic parameters, leading to sparse graphs (see Section 3.3.1). The size of the graphs varies between 68 and 334 vertices. Depending on the two parameters $|\mathcal{P}|$ and L, 18 or 21 instances are solved.

We use another benchmark to analyze our results on different and structured instances that can be found online for reproducibility purposes. It is composed by graphs designed for a tree-width problem of the 2016 PACE challenge [35], available at `http://bit.ly/pace16-tw-instances-20160307`. We select graphs having a size between 30 and 3300 vertices and containing enough paths of the given length L. We decompose this benchmark into two parts: the PACE-exact (up to 600 vertices) and the PACE-heuristic instances which contains larger instances. Depending on the two parameters $|\mathcal{P}|$ and L, 93 to 97 instances are solved.

Choice of the path set $\mathcal{P}$. As none of the used strategies depends on the weight function, any path can belong to the set $\mathcal{P}$, so we draw them randomly in the graph. Their number $|\mathcal{P}|$ varies in $\{3, 10, 50\}$.

[3] actually $\mathcal{P}$ is composed by random paths playing the role of optimal paths, the optimality is not important here.

Choice of the algorithms. We compare four strategies:

1. color coding with `unif` coloring (standard version)
2. color coding with `spread` coloring
3. color coding with `la` ordering and `spread` coloring
4. color coding with `la` ordering and `shifted-spread` coloring

The point of these experiments is mainly to establish the impact of the `spread` coloring strategy and how a *good* ordering can affect it. Thus, even if it is known that using more colors than L leads to a higher chance that H is colorful, we stand in a simpler case where $C = L$. For the same reason, the time limit given to the local search of the `la` ordering is sufficiently long (5 minutes) to hope for a good ordering. This time limit can be adapted to fit the need of the various applications of the color coding. Note that both strategies 3 and 4 use the same `la` ordering.

6.5.2 Analysis

Coloring `spread` versus `unif`. The `spread` coloring (strategy 2) reveals moderately better results than the `unif` coloring. The average frequency with which a path of $\mathcal{P}$ is found increases from 2.26% to 3.54%. As neither the coloring nor an ordering take profit from any structure and because of the size of our instances, the benefit of this coloring strategy alone is quite small. This is consistent with the analysis made in Section 6.2 for large values of n.

`la` ordering versus uniform ordering. The frequencies with which at least one path of $\mathcal{P}$ is colorful are denoted by f_u, f_3 and f_4 for strategies 1, 3 and 4 respectively. They estimate the probability to find an optimal path within one trial of the color coding. The average frequencies on a set of instances are denoted by $\overline{f_u}$, $\overline{f_3}$ and $\overline{f_4}$. We also computed the minimum (resp. average) number of ordering intervals I needed to cover a path of $\mathcal{P}$, denoted by $\#I_u^m$ (resp. $\overline{\#I_u}$) for the uniform ordering and $\#I_{la}^m$ (resp. $\overline{\#I_{la}}$) for `la` ordering.

The gain on the frequencies estimates the gain on the probability that an optimal path is colorful, which is, equivalently, a gain on the number of color coding trials needed to find an optimal path. We are however interested in the gain on the number of calls to the costly dynamic program. For the non shifted strategy, each trial calls only one dynamic program, so $\overline{g_3} = \frac{\overline{f_3}}{\overline{f_u}}$. On the contrary, one trial of `shifted-spread` calls C dynamic programs, thus, we compute the gain of the normalized frequency $normf_4 = \frac{f_4}{C}$ and $\overline{g_4} = \frac{\overline{normf_4}}{\overline{f_u}}$.

Tables 6.1, 6.2 and 6.3 show the aggregated results for the benchmarks E-KB, PACE-exact and PACE-heuristic respectively: the average

frequency of `unif` coloring ($\overline{f_u}$), the average frequency and gain for `la` ordering with `spread` ($\overline{f_3}$ and $\overline{g_3}$) and the average frequency with its net gain for `shifted-spread` ($\overline{f_4}$ and $\overline{g_4}$). They also display the average (over instances) minimum number of intervals covering a path of $\mathcal{P}$ for each ordering ($\overline{\#I_u^m}$ and $\overline{\#I_{la}^m}$). In these tables, # denotes the number of instances of the benchmark that were tested. We also detail individual results for PACE-exact benchmark, $L = C = 15$ and $|\mathcal{P}| = 3$ (chosen as examples), in Appendix B.

The bold values in Tables 6.1, 6.2 and 6.3, enlighten the significant gain of our methods in almost every cases, so we can expect a substantial reduction of the dynamic program executions to find an optimal path. This gain sometimes reaches several orders of magnitude, in particular for the PACE-exact benchmark (Table 6.2). Note that strategy 4 gives very high frequencies, but the third strategy has a slightly better net gain and the shift seems unnecessary. In one configuration (out of 9), even these high frequencies do not compensate this shifting technique and $\overline{g_4}$ is smaller than 1. However, only `shifted-spread` provides the guarantee that a path is found in one trial if $\Delta \leq C$. Actually, as discussed in Section 6.3, Δ might big (see Figure 6.2), `shifted-spread` still outperforms `unif`, and we can observe in Appendix B a number of instances with $f_4 = 1$ but $\Delta > C$ (in Table B.2). This probability of one occurs when $\#I_{la}^m = 1$, *i.e.*, when our ordering sufficiently gathered the vertices of one path of $\mathcal{P}$ so that one of the shifts puts them in a single coloring interval. This happens many times for the PACE-exact benchmark, which explains the good performance of `shifted-spread`. The side effect of the normalization on values bounded by 1 explains the important gap between $\overline{g_3}$ and $\overline{g_4}$ for this benchmark. More generally, when $\overline{\#I_{la}^m}$ is much smaller than $\overline{\#I_u^m}$, we observe a significant improvement of the colorful frequency, for both strategies 3 and 4, since our ordering clusters the vertices of the paths of $\mathcal{P}$ in a small number of intervals. On the contrary, when $\overline{\#I_{la}^m}$ and $\overline{\#I_u^m}$ are close, our methods behave similarly to `spread` alone since the ordering strategy gathers the vertices of the paths in almost as many intervals as a uniform ordering strategy. When this happens, the gain is attributable only to `spread` and remains, therefore, rather small.

6.6 Conclusion

This chapter introduced a new framework for the color coding approach including a preprocessing phase ordering the vertices and a new procedure to color the graph. The coloring strategy significantly improves the probability to find a subgraph for small graphs by spreading colors uniformly. When combined with an ordering strategy based on the graph structure, the proposed algorithm dominates the original color coding, as it preserves the

$\|\mathcal{P}\|$	L	#	uniform ordering		`la` ordering				
			`unif`	$\overline{\#I_u^m}$	`spread`		`shifted-spread`		$\overline{\#I_{la}^m}$
			$\overline{f_u}$		$\overline{f_3}$	$\overline{g_3}$	$\overline{f_4}$	$\overline{g_4}$	
3	10	21	0.0108	6.1	0.0278	**2.6**	0.1945	**1.8**	5.2
	15	18	0.0001	7.2	0.0003	**2.7**	0.0051	**2.6**	6.3
	20	18	3.7E^{-7}	7.2	1E^{-5}	**27**	0.0002	**26.1**	6.1
10	10	21	0.034	5.7	0.1088	**3.2**	0.4537	**1.3**	4.6
	15	18	0.0004	6.5	0.0011	**2.4**	0.0159	**2.4**	5.8
	20	18	6.7E^{-6}	6.7	2.9E^{-5}	**4.3**	0.0006	**4.7**	5.9
50	10	18	0.1571	4.8	0.2015	**1.3**	0.8918	0.57	4.6
	15	18	0.0022	5.8	0.0053	**2.4**	0.0749	**2.2**	5.3
	20	18	2.6E^{-5}	6.1	0.0002	**5.9**	0.003	**5.7**	5.4

Table 6.1 – Results for E-KB

$\|\mathcal{P}\|$	L	#	uniform ordering		`la` ordering				
			`unif`	$\overline{\#I_u^m}$	`spread`		`shifted-spread`		$\overline{\#I_{la}^m}$
			$\overline{f_u}$		$\overline{f_3}$	$\overline{g_3}$	$\overline{f_4}$	$\overline{g_4}$	
3	10	40	0.0106	4.8	0.2419	**22.8**	0.6109	**5.8**	2
	15	39	0.0001	4.8	0.0978	**765.2**	0.3803	**198.3**	2
	20	39	1E^{-6}	4.4	0.0859	**83734**	0.2261	**11021**	2.6
10	10	40	0.0336	4.4	0.4449	**13.2**	0.8904	**2.6**	1.7
	15	39	0.0004	4.4	0.2435	**568.7**	0.5133	**79.9**	1.8
	20	39	4.4E^{-6}	4.2	0.1653	**37184**	0.3021	**3399**	2.3
50	10	39	0.1308	3.9	0.7907	**6**	0.9953	0.76	1.4
	15	38	0.0019	4.2	0.4667	**248.9**	0.7251	**25.8**	1.7
	20	36	1.9E^{-5}	4.2	0.1954	**10243**	0.425	**1114**	2.1

Table 6.2 – Results for PACE-exact

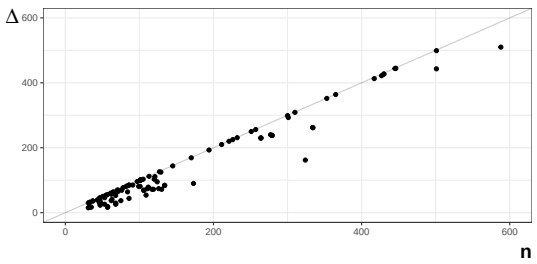


Figure 6.2 – Δ for each instance of size < 600 with $L = 10$

$\lvert\mathcal{P}\rvert$	L	#	uniform ordering		`la` ordering				
			`unif`	$\overline{\#I_u^m}$	`spread`		`shifted-spread`		$\overline{\#I_{la}^m}$
			$\overline{f_u}$		$\overline{f_3}$	$\overline{g_3}$	$\overline{f_4}$	$\overline{g_4}$	
3	10	55	0.0109	5.8	0.0187	**1.7**	0.1651	**1.5**	5
	15	52	0.0001	6.3	0.0006	**4.5**	0.0077	**4.1**	5.6
	20	52	1.2E^{-6}	6.3	3.8E^{-5}	**33**	0.0007	**29**	5.7
10	10	55	0.0355	5.3	0.0597	**1.7**	0.4327	**1.7**	4.5
	15	52	0.0004	5.9	0.0019	**4.3**	0.0262	**3.9**	5.1
	20	52	4.7E^{-6}	6.1	0.0001	**22.8**	0.0021	**22.3**	5.3
50	10	55	0.1635	4.7	0.2568	**1.6**	0.921	0.66	4.1
	15	52	0.0022	5.4	0.0089	**4**	0.1169	**3.5**	4.6
	20	52	2.8E^{-5}	5.5	0.0006	**20.1**	0.0111	**20**	4.9

Table 6.3 – Results for PACE-heuristic

improvement of the coloring strategy in graph with small Δ, a parameter related to the bandwidth.

The complete framework also includes a shifting technique that guarantees to find an optimal solution with only C calls to the dynamic program when the ordering step ensures that Δ is smaller than C. With or without the shifting, a randomized color coding approach using our algorithm needs far fewer calls to the dynamic programming step to expect the same chance to obtain an optimal solution than in the standard version using a uniform coloring. These algorithms were tested on realistic instances of the kidney exchange problem, but also on graphs coming from structural problems [35]. Similar results can be expected for graphs from other domains, particularly if the graphs are sparse. We now intend to investigate the consequences of our strategies for derandomization purposes.

Conclusion: contributions and future work

Different types of problems and algorithms are addressed in this thesis, all emerging from our study of the kidney exchange problem (KEP). Our contributions on these topics are summed up below, together with directions for future work.

We first surveyed the different variants of the KEP and its closely related problems, both from an applicative and mathematical point of view. From this literature review, we observed that kidney exchange programs are growing and that altruistic donors are more and more allowed, but difficult to take into account. In this case, the problem is to find cycles and paths of maximum weight, but limited length, in a graph modeling the compatibilities between donors and patients. We motivated our choice to consider the problem as static and deterministic by the difficulty to accommodate several new parameters at the same time. We thus focused on dealing with altruistic donors, but it would certainly be interesting to investigate how the stochasticity and dynamism of kidney exchange programs could be integrated in our models and algorithms.

We also explained how the kidney exchange problem is actually a stable set problem, for which we introduced a new extended formulation. We proved interesting properties for this formulation: it is ideal for perfect graphs, compact for claw-free graphs, and we know which standard inequalities are contained in its polytope. Despite these results, we did not implement the formulation as it would require a complex column-and-row generation to solve the KEP. This is however the next step we aim at taking.

We detailed other integer programming formulations for the KEP in an extensive literature review. Both from a theoretical and practical analysis, the exchange formulation (EF) seems to be the best candidate to solve large instances. This is confimed by the experiments we conducted, in particular the comparison between a compact formulation and the exponential formulation EF. Some polyhedral comparisons of these formulations are however missing and we plan to fill this gap in further work.

The exchange formulation is a large-scale model as it contains a variable for each feasible path and cycle. To solve its linear relaxation, we thereby developed a complete column generation framework integrating altruistic donors. We proposed filtering techniques but also to avoid the implementation of a whole branch-and-price by computing feasible solutions of good quality with other methods. In particular, we built an approximation based on iterative rounding providing an integer solution with a guaranteed quality over the optimal one. Experimental results show that our column generation indeed finds excellent solutions for realistic (up to 300 vertices) but also larger (up to 800 vertices) instances. In average, we were able to assess that the obtained solution weighs at least 99.88% of the optimal value. Our column generation has however some practical limits when it comes to large instances: memory issues and an increased running time to solve the final integer program. We are confident in the fact that an efficient implementation, taking profit from the sparsity of graphs, would overcome some of these limits and reach a new scale in the KEP instances that can be solved. Moreover, our work mainly focused on the pricing problem, so we intend to develop new ways to get feasible solutions, during and after the column generation.

Solving this linear relaxation with column generation requires to solve the elementary minimum path problem with length constraint, which is NP-hard. This problem is actually the core issue of our algorithm and is the subject of two chapters. We proposed a local search approach as well as an integer program to solve it. More importantly, we adapted, from the literature of elementary path problem with resource constraints, two dynamic programs: the color coding and the NG-route relaxation. This adaptation involves to take profit from the graph structure but also from the problem itself, using the length constraint to reduce the search space. We presented a computational evaluation of these algorithms and their different configurations, showing that the dynamic programming approach is very efficient to solve the considered problem. Indeed, the two dynamic programs always returned a solution with the same sign than the optimal one and this behavior is more important than finding an optimal solution in a column generation framework. Further work can be conducted on these algorithms, in particular to improve the filtering efficiency. We plan for example to apply the memory cuts proposed by Pecin *et al.* [99].

In addition, we carried out a theoretical analysis of the different techniques that we proposed for the color coding. In the original heuristic, vertices are randomly and simultaneously colored, then a dynamic program searches for a best solution with different colors (a colorful path), and the procedure is repeated multiple times. The color coding can be used to solve several problems and in particular our path problem. Its iterative aspect makes it particularly interesting in our column generation, as feasible solutions can be returned anytime. We proposed a new strategy that sequentially

colors the vertices and aims at giving optimal paths a higher chance to be colorful. We proved that our color coding technique indeed finds an optimal solution with a higher probability than the original one. To do so, a preprocessing step orders the vertices by reasoning on the graph and problem structures to gather feasible solutions. When the preprocessed graph satisfies a structural property related to the bandwidth, it guarantees that a bounded number of different colorings makes every feasible solution colorful, so the probability to find an optimal solution is equals to 1. In practice, the frequency with which our algorithm finds an optimal path is very high, and can reach 1 on instances with more than 300 vertices, even when this particular property is not satisfied. As the color coding has been studied in several papers in a derandomized version, we intend to pursue our research on this topic by combining our improvement with this derandomization procedure.

The results presented in this thesis led to the following research articles:

[I] **Column Generation for the Kidney Exchange Problem**, with Hadrien Cambazard, Nicolas Catusse and Gautier Stauffer
Proceedings of the 12th International Conference on MOdeling, Optimization and SIMlation, 2018, 149–156
https://hal.archives-ouvertes.fr/hal-01989427

[II] **New randomized strategies for the color coding algorithm**, with Hadrien Cambazard and Nicolas Catusse
accepted in ECAI 2020

Appendix A

Comparisons of formulations

Lemma A.1

EF is tighter than HPIEF
EF is tighter than PICEF

Proof. See Dickerson *et al.* [37] □

Lemma A.2

EF is tighter than MTZ-AF

Proof. Let x be the optimal solution EFL, the linear relaxation of EF. We show how to construct an equivalent optimal solution to the linear relaxation of MTZ-AF.

$$y_{ij} = \sum_{\substack{e\in\mathcal{E}\\(ij)\in A(e)\\ e \text{ is a path}}} x_e \text{ and } u_{ij} = \sum_{\substack{e\in\mathcal{E}\\(ij)\in A(e)\\ e \text{ is a cycle}}} x_e.$$

The physiological constraint (3.7) and (3.8) of MTZ-AF are verified due to the equivalent constraints (3.2) in EF. By the very definition of exchanges, flow constraints (3.5) and (3.6), as well as the length constraints (3.10) and (3.9), are satisfied. By construction, paths and cycles of $\mathcal{E}$ do not induce subtours and from constraints (3.2) we have: $\forall S \subset V, \sum_{(ij)\in A(S)} y_{ij} \leq |S| - 1$. Thus, we can construct time stamps t_i which are well-defined for MTZ-AF as proposed by Velednitsky [125]:

Given the graph with new arc weights defined as follows: $\forall (ij) \in A$, $w'_{ij} = (|P| + 1) - (|P| + 2)y_{ij} - |P|y_{ji}$. We set $-t_i$ as the cost of the shortest path from any vertex of N (altruistic donor) to i in D. MTZ constraints (3.11) are satisfied:

$$\begin{aligned} -t_j &\leq -t_i + w'_{ij} = -t_i + (|P| + 1) - (|P| + 2)y_{ij} - |P|y_{ji} \\ \Rightarrow t_i - t_j &+ |P|y_{ji} + (|P| + 2)y_{ij} \leq |P| + 1 \end{aligned}$$

Moreover, the shortest paths are well-defined as the graph contains no cycle of negative weight. Let C be a cycle in D and $c = |V(C)|$.

$$
\begin{aligned}
w'(C) = \sum_{(ij)\in A(C)} w'_{ij} &= \sum_{(ij)\in A(C)} ((|P|+1) - (|P|+2)y_{ij} - |P|y_{ji}) \\
&= c(|P|+1) - (|P|+2) \sum_{(ij)\in A(C)} y_{ij} - |P| \sum_{(ij)\in A(C)} y_{ji} \\
&\geq c(|P|+1) - (|P|+2) \sum_{(ij)\in A(C)} (y_{ij} + y_{ji})
\end{aligned}
$$

As discussed above, $\sum_{(ij)\in A(C)} (y_{ij} + y_{ji}) \leq c - 1$, so:

$$
\begin{aligned}
w'(C) &\geq c(|P|+1) - (|P|+2)(c-1) \\
&\geq |P| + 2 - c
\end{aligned}
$$

As a cycle can include only vertices of P (pairs), we have $c \leq |P|$ and $w'(C) \geq 0$.

□

Lemma A.3

EF is tighter than MTZ-EAF

Proof. The proof is similar. □

Lemma A.4

EF is tighter than PC-TSP.

Proof. Let x be the optimal solution EFL, the linear relaxation of EF. We show how to construct an equivalent optimal solution to the linear relaxation of MTZ-AF.

$$
y^l_{ij} = \sum_{\substack{e\in\mathcal{E} \\ (ij)\in A(e) \\ e \text{ is a path starting from vertex } l}} x_e.
$$

By definition the physiological constraint constraints (3.40) and (3.41) are respected, as well as the flow constraints (3.42) and length constraints (3.43). Moreover, by construction, paths and cycles of $\mathcal{E}$ do not induce subtours so constraints (3.44) are also satisfied.

□

Appendix B

Detailed results for color coding

The following tables show, for E-KB and PACE-exact benchmarks, $L = C = 15$ and $|\mathcal{P}| = 3$, the following results:

- The frequencies with which at least one path of $\mathcal{P}$ is colorful for strategies 1, 3 and 4: f_u, f_3 and f_4.
- The minimum and average number of ordering intervals I needed to cover a path of $\mathcal{P}$: $\#I_u^m$ and $\overline{\#I_u}$ for the uniform ordering, $\#I_{la}^m$ and $\overline{\#I_{la}}$ for `la` ordering.
- Δ
- $n = |V|$, $m = |E|$ and $d = \frac{m}{n(n-1)}$ the density of the graph $G = (V, E)$

					uniform ordering			`la` ordering				
					`unif`	$\overline{\#I_u}$	$\#I_u^m$	`spread`	`shift`	Δ	$\overline{\#I_{la}}$	$\#I_{la}^m$
instance		n	m	d	f_u			f_3	f_4			
final	100-10-4-l12	112	731	0.059	0.0001	5.7	5	0.0004	0.0074	78	5	5
	100-25-4-l12	134	834	0.047	0.0001	6	5	0.0004	0.0063	84	6	5
	100-5-4-l12	106	679	0.061	0.0001	5	5	0.0007	0.0112	69	4.7	3
	250-10-4-l12	279	4652	0.06	0.0001	9.7	9	0.0002	0.0029	238	10.7	8
	250-25-4-l12	334	4670	0.042	0.0002	11	9	0.0001	0.0028	262	10.3	8
first	100-10-4-l12	112	731	0.059	0.0002	5.7	5	0.0005	0.0068	78	5	4
	100-25-4-l12	134	834	0.047	0.0001	7	7	0.0005	0.0073	84	5.3	4
	100-5-4-l12	106	679	0.061	0.0001	5	5	0.0007	0.0078	69	4.3	4
	250-10-4-l12	279	4652	0.06	0.0001	9.7	9	0.0002	0.0026	238	10.7	9
	250-25-4-l12	334	4670	0.042	0.0002	9.7	9	0.0002	0.0029	262	9.7	8
	250-5-4-l12	264	4524	0.065	0.0001	10.3	9	0.0002	0.003	230	9.3	9
middle	100-10-4-l12	112	731	0.059	0.0002	5.7	5	0.0004	0.0059	78	5.3	5
	100-25-4-l12	134	834	0.047	0.0002	6.3	6	0.0003	0.0068	84	5.7	5
	100-5-4-l12	106	679	0.061	0.0001	5	5	0.0005	0.0063	69	5	4
	250-10-4-l12	279	4652	0.06	0.0001	8.7	8	0.0002	0.0026	238	11	8
	250-25-4-l12	334	4670	0.042	0.0001	11.3	10	0.0002	0.0025	265	11	8
	250-5-4-l12	264	4524	0.065	0.0001	9.3	9	0.0002	0.0031	230	8.7	8

Table B.1 – Results for E-KBR, $L = C = 15$ and $|\mathcal{P}| = 3$

	instance	n	m	d	uniform ordering			la ordering				
					unif	$\overline{\#I_u}$	$\#I_u^m$	spread	shift	Δ	$\overline{\#I_{la}}$	$\#I_{la}^m$
					f_u			f_3	f_4			
easy-contiki	cxmac-input-packet	91	284	0.035	0.0002	6	6	0.0022	0.0183	90	4	3
	dhcpc-dhcpc-init	35	102	0.086	0.0001	3	3	0.1364	**1**	**20**	**2**	**1**
	httpd-cfs-send-file	45	140	0.071	0.0001	3	3	0.0016	0.0219	44	3	3
	ifft-ifft	173	532	0.018	0.0002	9	7	0.0009	0.0138	143	4	3
	ircc-list-channel	71	222	0.045	0.0001	4.7	4	0.0265	0.0776	70	3.7	3
	lpp-send-packet	117	356	0.026	0.0001	6.3	6	0.14	**1**	**116**	**2.7**	**1**
	polite-announcement-send-timer	32	93	0.094	0.0001	2.3	2	0.0578	**1**	**23**	**2**	**1**
	powertrace-add-stats	47	140	0.065	0.0001	3.3	3	0.0097	0.0627	46	2.7	2
	powertrace-powertrace-print	324	969	0.009	0.0001	10.3	10	0.0064	**1**	**162**	**2.3**	**1**
	profile-profile-episode-start	32	95	0.096	0.0001	2.7	2	**1**	**1**	**29**	**2**	**1**
	psock-psock-generator-send	62	197	0.052	0.0001	2	3	0.0012	0.0155	61	3.7	3
	rudolph1-send	31	88	0.095	0.0002	3	3	**1**	**1**	**30**	**4**	**1**
	shell-shell-register-command	43	132	0.073	0.0001	3	3	0.0043	0.0338	42	2.7	2
	shell-collect-view-process-thread-data	62	185	0.049	0.0001	4.7	4	0.0046	**1**	**54**	**2.7**	**1**
	shell-rime-recv-collect	63	190	0.049	0.0001	4.3	4	0.0034	0.357	50	2	2
	shell-rime-ping-recv-mesh	48	141	0.062	0.0001	3.7	3	0.1345	0.3154	47	3	2
	tcpip-eventhandler	99	322	0.033	0.0001	6	5	0.0018	**1**	**95**	**3.3**	**1**
	uip-neighboradd	68	209	0.046	0.0001	4.7	4	0.002	0.0247	57	3	2
	uip-over-mesh-recv-data	86	261	0.036	0.0002	5.7	5	0.0195	**1**	**43**	**2.3**	**1**
	webclient-senddata	109	326	0.028	0.0001	7.3	6	0.0117	0.0521	72	2.3	2

					uniform ordering			la ordering				
					unif	$\overline{\#I_u}$	$\#I_u^m$	spread	shift	Δ	$\overline{\#I_{la}}$	$\#I_{la}^m$
instance		n	m	d	f_u			f_3	f_4			
easy-fuzix	devf-fd-transfer	120	377	0.026	0.0001	6.7	6	0.0038	0.0697	119	2.3	2
	devio-kprintf	70	225	0.047	0.0002	5	5	0.0005	0.009	69	4.7	4
	difftime-difftime	75	220	0.04	0.0001	4.7	4	0.0647	**1**	**37**	**2**	**1**
	fgets-fgets	54	169	0.059	0.0002	4	4	0.0007	0.01	53	4	3
	filesys-filename	46	141	0.068	0.0001	3	3	0.1343	0.3535	45	3	2
	filesys-getinode	53	166	0.06	0.0001	4	4	0.0029	0.0345	52	3.3	2
	filesys-i-open	130	415	0.025	0.0001	8.7	8	0.004	0.0539	129	3	2
	getpass–gets	32	101	0.102	0.0001	3	3	0.0041	0.0701	31	2.7	2
	malloc—insert-chunk	105	336	0.031	0.0001	6	6	0.0008	0.0182	104	4.7	4
	process-getproc	33	102	0.097	0.0002	2.3	2	0.0022	0.0602	32	3	2
	ran-rand	47	142	0.066	0.0001	4	4	0.0214	**1**	**23**	**2**	**1**
	regexp-regcomp	119	376	0.027	0.0001	7.3	7	0.0007	0.0126	116	4	3
	se-ycomp	84	275	0.039	0.0001	5.3	5	0.0009	0.0112	83	3.7	3
	stat-statfix	53	154	0.056	0.0001	4	4	**1**	**1**	**26**	**1.7**	**1**
	tty-tty-read	124	397	0.026	0.0001	6.7	6	0.0031	**1**	**123**	**3**	**1**
easy-stdlib-sincoshf		111	344	0.028	0.0002	7	6	0.0016	0.0342	98	3.3	2
hard	DoubleStarSnark	31	120	0.129	0.0001	12	11	0.0004	0.0051	30	6.3	4
	contiki-dhcpc-handle-dhcp	277	902	0.012	0.0001	2	2	0.0046	0.0701	270	2	2
	fuzix-vfscanf-vfscanf	588	1923	0.006	0.0001	12	12	0.0007	0.0268	584	4.7	2

Table B.2 – Results for the PACE-exact benchmark, $L = C = 15$ and $|\mathcal{P}| = 3$

Index of definitions

Index of symbols and abbreviations

Bibliography

[1] David J Abraham, Avrim Blum, and Tuomas Sandholm. Clearing algorithms for barter exchange markets: Enabling nationwide kidney exchanges. In *Proceedings of the 8th ACM conference on Electronic commerce*, pages 295–304. ACM, 2007.

[2] D Adolphson and T Ch Hu. Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25(3):403–423, 1973.

[3] Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and S. Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24:i241–i249, 2008.

[4] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

[5] Filipe Alvelos, Xenia Klimentova, and Ana Viana. Maximizing the expected number of transplants in kidney exchange programs with branch-and-price. *Annals of Operations Research*, 272(1-2):429–444, 2019.

[6] Ross Anderson, Itai Ashlagi, David Gamarnik, and Yash Kanoria. A dynamic model of barter exchange. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 1925–1933. Society for Industrial and Applied Mathematics, 2015.

[7] Ross Anderson, Itai Ashlagi, David Gamarnik, and Alvin E Roth. Finding long chains in kidney exchange using the traveling salesman problem. *Proceedings of the National Academy of Sciences*, 112(3):663–668, 2015.

[8] Itai Ashlagi, David Gamarnik, Michael A Rees, and Alvin E Roth. The need for (long) chains in kidney exchange. Technical report, National Bureau of Economic Research, 2012.

[9] Itai Ashlagi, Duncan S Gilchrist, Alvin E Roth, and Michael A Rees. Nonsimultaneous chains and dominos in kidney-paired donation—revisited. *American Journal of transplantation*, 11(5):984–994, 2011.

[10] Itai Ashlagi and Alvin E Roth. Individual rationality and participation in large scale, multi-hospital kidney exchange. Technical report, National Bureau of Economic Research, 2011.

[11] Itai Ashlagi and Alvin E Roth. New challenges in multi-hospital kidney exchange. In *The American Economic Review*. American Economic Association, 2012.

[12] Pranjal Awasthi and Tuomas Sandholm. Online stochastic optimization in the large: Application to kidney exchange. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[13] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283, oct 2011.

[14] Max Bannach, Christoph Stockhusen, and Till Tantau. Fast parallel fixed-parameter algorithms via color coding. arXiv preprint arXiv:1509.06984, 2015.

[15] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

[16] Dimitris Bertsimas, Vivek F Farias, and Nikolaos Trichakis. Fairness, efficiency, and flexibility in organ allocation for kidney transplantation. *Operations Research*, 61(1):73–87, 2013.

[17] Péter Biró, Lisa Burnapp, Bernadette Haase, Aline Hemke, Rachel Johnson, Joris van de Klundert, and David Manlove. First handbook: Kidney exchange practices in europe. Technical report, European Network for Collaboration on Kidney Exchange Programmes, 2017.

[18] Péter Biró, Bernadette Haase-Kromwijk, Tommy Andersson, Eyjólfur Ingi Ásgeirsson, Tatiana Baltesová, Ioannis Boletis, Catarina Bolotinha, Gregor Bond, Georg Böhmig, Lisa Burnapp, et al. Building kidney exchange programmes in europe—an overview of exchange practice and activities. *Transplantation*, 103(7):1514, 2019.

[19] Péter Biro, David F Manlove, and Romeo Rizzi. Maximum weight cycle packing in directed graphs, with application to kidney exchange programs. *Discrete Mathematics, Algorithms and Applications*, 1(04):499–517, 2009.

[20] Péter Biró, Joris van de Klundert, David Manlove, William Pettersson, Tommy Andersson, Lisa Burnapp, Pavel Chromy, Pablo Delgado, Piotr Dworczak, and Bernadette Haase. Modelling and optimisation in european kidney exchange programmes. Corrected proof, 2019.

[21] Georg A Böhmig, Samantha Fidler, Frank T Christiansen, Gottfried Fischer, and Paolo Ferrari. Transnational validation of the australian algorithm for virtual crossmatch allocation in kidney paired donation. *Human immunology*, 74(5):500–505, 2013.

[22] Ralf Borndörfer, Martin Grötschel, and Marc E Pfetsch. A column-generation approach to line planning in public transport. *Transportation Science*, 41(1):123–132, 2007.

[23] Leizhen Cai, Siu Man Chan, and Siu On Chan. Random separation: A new method for solving fixed-cardinality optimization problems. In *International Workshop on Parameterized and Exact Computation*, pages 239–250. Springer, 2006.

[24] LáZaro CáNovas, Mercedes Landete, and Alfredo MarıN. New facets for the set packing polytope. *Operations Research Letters*, 27(4):153–161, 2000.

[25] Ioannis Caragiannis, Christos Kaklamanis, Panagiotis Kanellopoulos, and Maria Kyropoulou. The efficiency of fair division. *Theory of Computing Systems*, 50(4):589–610, 2012.

[26] Jianer Chen, Joachim Kneis, Songjian Lu, Daniel Mölle, Stefan Richter, Peter Rossmanith, Sing-Hoi Sze, and Fenghui Zhang. Randomized divide-and-conquer: Improved path, matching, and packing algorithms. *SIAM Journal on Computing*, 38(6):2526–2547, 2009.

[27] Yanhua Chen, Jack D Kalbfleisch, Yijiang Li, Peter XK Song, and Yan Zhou. Computerized platform for optimal organ allocations in kidney exchanges. In *Proceedings of the International Conference on Bioinformatics & Computational Biology (BIOCOMP)*, page 1. The Steering Committee of The World Congress in Computer Science, 2011.

[28] Eddie Cheng and William H Cunningham. Wheel inequalities for stable set polytopes. *Mathematical programming*, 77(2):389–421, 1997.

[29] Eddie Cheng and Sven de Vries. Antiweb-wheel inequalities and their separation problems over the stable set polytopes. *Mathematical Programming*, 92(1):153–175, 2002.

[30] Vašek Chvátal. On certain polytopes associated with graphs. *Journal of Combinatorial Theory, Series B*, 18(2):138–154, 1975.

[31] Miguel Constantino, Xenia Klimentova, Ana Viana, and Abdur Rais. New insights on integer-programming models for the kidney exchange problem. *European Journal of Operational Research*, 231(1):57–68, 2013.

[32] COST. European network for collaboration on kidney exchange programmes. http://www.cost.eu/COST_Actions/ca/CA15210, 2016.

[33] Elizabeth Cuthill and James McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, pages 157–172. ACM, 1969.

[34] Marry De Klerk, Karin M Keizer, Frans HJ Claas, Marian Witvliet, Bernadette JJM Haase-Kromwijk, and Willem Weimar. The dutch national living donor kidney exchange program. *American Journal of Transplantation*, 5(9):2302–2305, 2005.

[35] Holger Dell, Thore Husfeldt, Bart MP Jansen, Petteri Kaski, Christian Komusiewicz, and Frances A Rosamond. The first parameterized algorithms and computational experiments challenge. In *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2017.

[36] M. Desrochers. An algorithm for the shortest path problem with resource constraints. Technical Report G-88-27, GERAD, Ecole des HEC, Canada, 1988.

[37] John P Dickerson, David F Manlove, Benjamin Plaut, Tuomas Sandholm, and James Trimble. Position-indexed formulations for kidney exchange. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, pages 25–42. ACM, 2016.

[38] John P Dickerson, Ariel D Procaccia, and Tuomas Sandholm. Optimizing kidney exchange with transplant chains: Theory and reality. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 711–718. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[39] John P Dickerson, Ariel D Procaccia, and Tuomas Sandholm. Failure-aware kidney exchange. In *Proceedings of the fourteenth ACM conference on Electronic commerce*, pages 323–340. ACM, 2013.

[40] John P. Dickerson, Ariel D. Procaccia, Tuomas Sandholm, Computer Science Department, and Carnegie Mellon University. Price of fairness in kidney exchange. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, 2014.

[41] Banu Dost, Tomer Shlomi, Nitin Gupta, Eytan Ruppin, Vineet Bafna, and Roded Sharan. Qnet: a tool for querying protein interaction networks. *Journal of Computational Biology*, 15(7):913–925, 2008.

[42] Blake Ellison. A systematic review of kidney paired donation: Applying lessons from historic and contemporary case studies to improve the us model. Technical report, The Wharton School of the University of Pennsylvania, 2014.

[43] Hossein Esfandiari and Guy Kortsarz. A bounded-risk mechanism for the kidney exchange game. *Discrete Applied Mathematics*, 243:46–53, 2018.

[44] Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle and routing problems. *Networks: An International Journal*, 44(3):216–229, 2004.

[45] Michael R Fellows, Christian Knauer, Naomi Nishimura, Prabhakar Ragde, F Rosamond, Ulrike Stege, Dimitrios M Thilikos, and Sue Whitesides. Faster fixed-parameter tractable algorithms for matching and packing problems. In *European Symposium on Algorithms*, pages 311–322. Springer, 2004.

[46] P Ferrari, S Fidler, J Wright, C Woodroffe, P Slater, Van Althuis-Jones, R Holdsworth, FT Christiansen, et al. Virtual crossmatch approach to maximize matching in paired kidney donation. *American Journal of Transplantation*, 11(2):272–278, 2011.

[47] Lester R Ford Jr. Network flow theory. Technical report, Rand Corp Santa Monica Ca, 1956.

[48] Kenneth R Fox, Bezalel Gavish, and Stephen C Graves. An n-constraint formulation of the (time-dependent) traveling salesman problem. *Operations Research*, 28(4):1018–1021, 1980.

[49] Michael R Garey, David S Johnson, and Larry Stockmeyer. Some simplified np-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63. ACM, 1974.

[50] Sommer Gentry, Michal Mankowski, TS Michael, and Dorry Segev. Maximum matchings in graphs for allocating kidney paired donation. arXiv preprint arXiv:1710.00953, 2017.

[51] Sommer E Gentry, Robert A Montgomery, and Dorry L Segev. Kidney paired donation: fundamentals, limitations, and expansions. *American Journal of Kidney Diseases*, 57(1):144–151, 2011.

[52] Sommer E Gentry, Robert A Montgomery, Bruce J Swihart, and Dorry L Segev. The roles of dominos and nonsimultaneous chains in kidney paired donation. *American Journal of Transplantation*, 9(6):1330–1336, 2009.

[53] Sommer E Gentry, Dorry L Segev, and Robert A Montgomery. A comparison of populations served by kidney paired donation and list paired donation. *American Journal of Transplantation*, 5(8):1914–1921, 2005.

[54] J Alan George. Computer implementation of the finite element method. Technical report, Stanford University, Computer Science Department, 1971.

[55] K Glorie, A Wagelmans, and J van de Klundert. Iterative branch-and-price for large multi-criteria kidney exchange. Technical report, Econometric institute, Erasmus University Rotterdam, 2012.

[56] Kristiaan M Glorie, J Joris van de Klundert, and Albert PM Wagelmans. Kidney exchange with long chains: An efficient pricing algorithm for clearing barter exchanges with branch-and-price. *Manufacturing & Service Operations Management*, 16(4):498–512, 2014.

[57] Ralph E Gomory et al. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical society*, 64(5):275–278, 1958.

[58] Ruthanne L Hanto, William Reitsma, and Francis L Delmonico. The development of a successful multiregional kidney paired donation program. *Transplantation*, 86(12):1744–1748, 2008.

[59] A Hart, JM Smith, MA Skeans, SK Gustafson, AR Wilk, S Castro, A Robinson, JL Wainright, JJ Snyder, BL Kasiske, et al. Optn/srtr 2017 annual data report: kidney. *American Journal of Transplantation*, 19:19–123, 2019.

[60] Michael Held and Richard M Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[61] Falk Hüffner, Sebastian Wernicke, and Thomas Zichner. Algorithm engineering for color-coding with applications to signaling pathway detection. *Algorithmica*, 52(2):114–132, 2008.

[62] Institute for Health Metrics and Evaluation. Gbd compare: Global burden of disease study, 2018.

[63] Zhipeng Jia, Pingzhong Tang, Ruosong Wang, and Hanrui Zhang. Efficient near-optimal algorithms for barter exchange. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 362–370. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[64] Inessa Kaplan, Julie A Houp, Mary S Leffell, John M Hart, and Andrea A Zachary. A computer match program for paired and unconventional kidney exchanges. *American Journal of Transplantation*, 5(9):2306–2308, 2005.

[65] KM Keizer, Md de Klerk, BJJM Haase-Kromwijk, and W Weimar. The dutch algorithm for allocation in living donor kidney exchange. *Transplantation proceedings*, 37(2):589–591, 2005.

[66] Beom Seok Kim, Yu Seun Kim, Soon Il Kim, Myoung Soo Kim, Ho Yung Lee, Yong-Lim Kim, Chan Duck Kim, Chul Woo Yang, Bum Soon Choi, Duck Jong Han, et al. Outcome of multipair donor kidney exchange by a web-based algorithm. *Journal of the American Society of Nephrology*, 18(3):1000–1006, 2007.

[67] Xenia Klimentova, Filipe Alvelos, and Ana Viana. A new branch-and-price approach for the kidney exchange problem. In *International Conference on Computational Science and Its Applications*, pages 237–252. Springer, 2014.

[68] Xenia Klimentova, João Pedro Pedroso, and Ana Viana. Maximising expectation of the number of transplants in kidney exchange programmes. *Computers & Operations Research*, 73:1–11, 2016.

[69] Joachim Kneis, Alexander Langer, and Peter Rossmanith. Derandomizing non-uniform color-coding I. Technical report, RWTH Aachen - Department of Computer Science, 08 2011.

[70] Ioannis Koutis. A faster parameterized algorithm for set packing. *Information processing letters*, 94(1):7–9, 2005.

[71] Anneke Kramer, Maria Pippias, Marlies Noordzij, Vianda S. Stel, Anton M. Andrusev, Manuel I. Aparicio-Madre, Federico E. Arribas Monzón, Anders Åsberg, Myftar Barbullushi, Palma Beltrán, et al. The european renal association–european dialysis and transplant association (era-edta) registry annual report 2016: a summary. *Clinical kidney journal*, 12(5):702–720, 2019.

[72] Leonieke W. Kranenburg, Tatjana Visak, Willem Weimar, Willij Zuidema, Marry de Klerk, Medard Hilhorst, Jan Passchier, Jan N.M. IJzermans, and Jan J.V. Busschbach. Starting a crossover kidney transplantation program in the netherlands: ethical and psychological considerations. *Transplantation*, 78(2):194–197, 2004.

[73] Vivek B Kute, Narayan Prasad, Pankaj R Shah, and Pranjal R Modi. Kidney exchange transplantation current status, an update and future perspectives. *World journal of transplantation*, 8(3):52, 2018.

[74] JY Kwak, OJ Kwon, Kwang Soo Lee, Chong Myung Kang, Hae Young Park, and JH Kim. Exchange-donor program in renal transplantation: a single-center experience. *Transplantation proceedings*, 31(1):344–345, 1999.

[75] Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*, volume 46. Cambridge University Press, 2011.

[76] Adam N Letchford, Fabrizio Rossi, and Stefano Smriglio. The stable set problem: Clique and nodal inequalities revisited. eprint on Optimization Online, 2018.

[77] Yijiang Li, Peter X-K Song, Yan Zhou, Alan B Leichtman, Michael A Rees, and John D Kalbfleisch. Optimal decisions for organ exchanges in a kidney paired donation program. *Statistics in biosciences*, 6(1):85–104, 2014.

[78] Yicheng Liu, Pingzhong Tang, and Wenyi Fang. Internally stable matchings and exchanges. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[79] Live Organ Donor Consensus Group. Consensus statement on the live organ donor. *Jama*, 284:2919–2926, 2000.

[80] AO Mahendran and PS Veitch. Paired exchange programmes can expand the live kidney donor pool. *British journal of surgery*, 94(6):657–664, 2007.

[81] Stephen J Maher. Solving the integrated airline recovery problem using column-and-row generation. *Transportation Science*, 50(1):216–239, 2015.

[82] Vicky Mak-Hau. On the kidney exchange problem: cardinality constrained cycle and chain problems on directed graphs: a survey of integer programming approaches. *Journal of Combinatorial Optimization*, 33(1):35–59, 2017.

[83] David F Manlove and Gregg O'Malley. Paired and altruistic kidney donation in the uk: Algorithms and experimentation. In *International Symposium on Experimental Algorithms*, pages 271–282. Springer, 2012.

[84] Itay Mayrose, Tomer Shlomi, Nimrod D Rubinstein, Jonathan M Gershoni, Eytan Ruppin, Roded Sharan, and Tal Pupko. Epitope mapping using combinatorial phage-display libraries: a graph-based algorithm. *Nucleic acids research*, 35(1):69–78, 2006.

[85] Jerry Menikoff. Organ swapping. *Hastings Center Report*, 29(6):28–34, 1999.

[86] Jerry Menikoff. An organ sale by any other name. *The American Journal of Bioethics*, 4(4):42–44, 2004.

[87] Beata Mierzejewska, Magdalena Durlik, Wojciech Lisik, Caitlin Baum, Paul Schroder, Jonathan Kopke, Michael Rees, and Stanislaw Stepkowski. Current approaches in national kidney paired donation programs. *Ann Transplant*, 18:112–124, 2013.

[88] Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[89] Edward F Moore. The shortest path through a maze. In *Proc. Int. Symp. Switching Theory, 1959*, pages 285–292, 1959.

[90] Christian Morath, Martin Zeier, Bernd Döhler, Gerhard Opelz, and Caner Süsal. Abo-incompatible kidney transplantation. *Frontiers in immunology*, 8:234, 2017.

[91] İbrahim Muter. *Simultaneous column-and-row generation for solving large-scale linear programs with column-dependent-rows*. PhD thesis, Sabancı University, 2011.

[92] Moni Naor, Leonard J Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 182–191. IEEE, 1995.

[93] E. Nemati, B. Einollahi, M. Lesan Pezeshki, V. Porfarziani, and M. Reza Fattahi. Does kidney transplantation with deceased or living donor affect graft survival? *Nephro-Urology Monthly*, 6(4):1–5, 2014.

[94] Brendon Lange Neuen, Steven James Chadban, Alessandro Rhyl Demaio, David Wayne Johnson, and Vlado Perkovic. Chronic kidney disease and the global ncds agenda. *BMJ Global Health*, 2(2):1–4, 2017.

[95] Manfred W Padberg. On the facial structure of set packing polyhedra. *Mathematical programming*, 5(1):199–215, 1973.

[96] Manfred W Padberg. Perfect zero–one matrices. *Mathematical Programming*, 6(1):180–196, 1974.

[97] Ch H Papadimitriou. The np-completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270, 1976.

[98] Jong-Hyun Park, Joong-Won Park, Young-Mo Koo, and Jang Han Kim. Relay kidney transplantation in korea—legal, ethical and medical aspects. *Legal Medicine*, 6(3):178–181, 2004.

[99] Diego Pecin, Artur Pessoa, Marcus Poggi, Eduardo Uchoa, and Haroldo Santos. Limited memory rank-1 cuts for vehicle routing problems. *Operations Research Letters*, 45(3):206–209, 2017.

[100] Diego Pecin, Marcus Poggi, and Rafael Martinelli. Efficient elementary and restricted non-elementary route pricing. Technical report, Pontifical Catholic University of Rio de Janeiro, 2013.

[101] Joao Pedro Pedroso. Maximizing expectation on vertex-disjoint cycle packing. In *International Conference on Computational Science and Its Applications*, pages 32–46. Springer, 2014.

[102] Benjamin Plaut. Algorithms for social good: Kidney exchange. Master's thesis, Carnegie Mellon University, 2016.

[103] Benjamin Plaut, John P Dickerson, and Tuomas Sandholm. Fast optimal clearing of capped-chain barter exchanges. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

[104] Benjamin Plaut, John P Dickerson, and Tuomas Sandholm. Hardness of the pricing problem for chains in barter exchanges. arXiv preprint arXiv:1606.00117, 2016.

[105] Felix T Rapaport. The case for a living emotionally related international kidney donor exchange registry. *Transplantation proceedings*, 18(3) Suppl. 2):5–9, 1986.

[106] Giovanni Righini and Matteo Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks: An International Journal*, 51(3):155–170, 2008.

[107] Lainie Friedman Ross, David T Rubin, Mark Siegler, Michelle A Josephson, J Richard Thistlethwaite, and E Steve Woodle. Ethics of a paired-kidney-exchange program. *New England Journal of Medicine*, 336(24):1752–1755, 1997.

[108] Lainie Friedman Ross and E Steve Woodle. Ethical issues in increasing living kidney donations by expanding kidney paired exchange programs. *Transplantation*, 69(8):1539–1543, 2000.

[109] Lainie Friedman Ross and Stefanos Zenios. Restricting living-donor–cadaver-donor exchanges to ensure that standard blood type o wait-list candidates benefit. *Transplantation*, 78(5):641–646, 2004.

[110] Alvin E Roth, Tayfun Sönmez, and M Ünver. Efficient kidney exchange: Coincidence of wants in markets with compatibility-based preferences. *The American economic review*, 97(3):828–851, 2007.

[111] Alvin E Roth, Tayfun Sonmez, and M Utku Unver. Kidney exchange. Technical report, National Bureau of Economic Research, 2003.

[112] Alvin E Roth, Tayfun Sönmez, and M Utku Ünver. Pairwise kidney exchange. *Journal of Economic theory*, 125(2):151–188, 2005.

[113] Ruslan Sadykov and François Vanderbeck. Column generation for extended formulations. *EURO Journal on Computational Optimization*, 1(1-2):81–115, 2013.

[114] Susan L Saidman, Alvin E Roth, Tayfun Sönmez, M Utku Ünver, and Francis L Delmonico. Increasing the opportunity of live kidney donation by matching for two-and three-way exchanges. *Transplantation*, 81(5):773–782, 2006.

[115] Rajiv Saran, Bruce Robinson, Kevin Abbott, Lawrence Agodoa, Patrick Albertus, John Ayanian, Rajesh Balkrishnan, Jennifer Bragg-Gresham, Jie Cao, Joline Chen, et al. US renal data system 2016 annual data report: epidemiology of kidney disease in the United States. *American journal of kidney diseases: the official journal of the National Kidney Foundation*, 69(3):A7–A8, 2017.

[116] Rajiv Saran, Bruce Robinson, Kevin C. Abbott, Lawrence YC. Agodoa, Nicole Bhave, Jennifer Bragg-Gresham, Rajesh Balkrishnan, Xue Dietrich, Ashley Eckard, Paul W. Eggers, et al. US renal data system 2017 annual data report: epidemiology of kidney disease in the United States. *American journal of kidney diseases: the official journal of the National Kidney Foundation*, 71(3 Suppl 1):A7, 2018.

[117] Jacob Scott, Trey Ideker, Richard M Karp, and Roded Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. *Journal of Computational Biology*, 13(2):133–144, 2006.

[118] Alfonso Shimbel. Structure in communication nets. In *Proceedings of the symposium on information networks*, pages 119–203. Polytechnic Institute of Brooklyn, 1954.

[119] Tomer Shlomi, Daniel Segal, Eytan Ruppin, and Roded Sharan. Qpath: a method for querying pathways in a protein-protein interaction network. *BMC bioinformatics*, 7(1):199, 2006.

[120] Transplant Observatory. Global observation on donation and transplantation, 2018.

[121] Leslie E Trotter Jr. A class of facet producing graphs for vertex packing polyhedra. *Discrete Mathematics*, 12(4):373–388, 1975.

[122] Dekel Tsur. Faster deterministic parameterized algorithm for k-path. arXiv preprint arXiv:1808.04185, 2018.

[123] M Utku Ünver. Dynamic kidney exchange. *The Review of Economic Studies*, 77(1):372–414, 2010.

[124] UWHealth. Longest kidney chain ever completed wraps up at uw hospital and clinics.

[125] Mark Velednitsky. Short combinatorial proof that the dfj polytope is contained in the mtz polytope for the asymmetric traveling salesman problem. arXiv preprint arXiv:1805.06997, 2018.

[126] Chen Wang, Chuan Xu, and Abdel Lisser. Bandwidth minimization problem. In *10th International Conference on MOdeling, Optimization and SIMulation - MOSIM14 - November 5-7 2014 Nancy - France "From linear economy to circular economy."*, 2014.

[127] E Steve Woodle and Lainie Friedman Ross. Paired exchanges should be part of the solution to abo incompatibility in living donor kidney transplantation. *Transplantation*, 66(3):406–407, 1998.

[128] Nicholas M Wragg, Liam Burke, and Samantha L Wilson. A critical review of current progress in 3d kidney biomanufacturing: advances, challenges, and recommendations. *Renal Replacement Therapy*, 5(1):18, 2019.

[129] Kyung Don Yoo, Clara Tammy Kim, Myoung-Hee Kim, Junhyug Noh, Gunhee Kim, Ho Kim, Jung Nam An, Jae Yoon Park, Hyunjeong Cho, Kyoung Hoon Kim, et al. Superior outcomes of kidney transplantation compared with dialysis: an optimal matched analysis of a national population-based cohort study between 2005 and 2008 in korea. *Medicine*, 95(33):1–11, 2016.

[130] Meirav Zehavi. Mixing color coding-related techniques. In *Algorithms-ESA 2015*, pages 1037–1049. Springer, 2015.

[131] Stefanos A Zenios, E Steve Woodle, and Lainie Friedman Ross. Primum non nocere: avoiding harm to vulnerable wait list candidates in an indirect kidney exchange. *Transplantation*, 72(4):648–654, 2001.

[132] Qipeng P Zheng, Siqian Shen, and Yuhui Shi. Loss-constrained minimum cost flow under arc failure uncertainty with applications in risk-aware kidney exchange. *IIE Transactions*, 47(9):961–977, 2015.

This thesis deals with elementary path problems and their application to the kidney exchange problem. We focus on kidney exchange programs including altruistic donors, which are crucial for patients with renal disease and challenging for operations research methods. The goal of this work is to develop an efficient algorithm that can be used to solve future instances, which are likely to involve a large number of donors and patients. While we progress on this topic, we encounter closely related problems on packing, vehicle routing and stable set. For this last problem, we introduce a new extended formulation and prove it is ideal and compact for claw-free perfect graphs by characterizing its polytope. We then concentrate on the design of a column generation dedicated to the kidney exchange problem and confront its NP-hard pricing problem. The specific problem that we address is the elementary path problem with length constraint, which models the search for interesting chains of donation to add during the pricing step. We investigate dynamic approaches, in particular the NG-route relaxation and the color coding heuristic, and improve them by exploiting the length constraint and sparsity of graphs. We study the color coding in a more general context, providing a guaranteed improvement by proposing new randomized strategies. They are based on ordering the graph before coloring it and introduce a bias in the probability distribution to increase the probability of finding an optimal solution.
Keywords: linear programming, kidney exchange problem, elementary path problem, column generation, color coding, NG-route, stable set problem

Cette thèse traite de problèmes de chemins élémentaires et leur application au problème d'échange de reins. Nous nous concentrons sur des programmes d'échange de reins qui incluent des donneurs altruistes, qui sont essentiels pour les patients avec une maladie rénale, mais représentent un défi pour les méthodes de recherche opérationnelle. Notre objectif est de développer un algorithme efficace qui pourra être utilisé pour résoudre des instances futures, qui sont susceptibles d'impliquer un grand nombre de participants. Nous rencontrons des problèmes étroitement lié au notre : problèmes de packing, de tournée de véhicules, de stable. Pour ce dernier, nous présentons une nouvelle formulation étendue et prouvons qu'elle est idéale et compacte pour les graphes parfaits sans griffe. Nous nous focalisons ensuite sur la conception d'une génération de colonnes dédiée au problème d'échange de reins et nous attaquons à son problème de pricing, NP-difficile. Nous abordons le problème du chemin élémentaire minimum avec contrainte de taille, qui modélise la recherche de chaînes de dons intéressantes à ajouter dans la phase du pricing. Nous étudions des approches dynamiques, en particulier la relaxation NG-route et l'heuristique de color coding, et les améliorons en exploitant la contrainte de taille et la faible densité des graphes considérés. Nous nous intéressons ensuite au color coding dans un contexte plus général, proposant de nouvelles stratégies randomisées qui apportent une garantie d'amélioration. Ces stratégies s'appuient sur un ordonnancement du graphe et introduisent un biais dans la loi de probabilité pour augmenter les chances de trouver une solution optimale.

Mots-clefs: programmation linéaire, problème d'échange de reins, problème de chemin élémentaire, génération de colonnes, color coding, NG-route, problème du stable