Université Pierre et Marie Curie (Paris 6) - Faculté d'Ingénierie (UFR 919)
École Doctorale Informatique, Télécommunications et Électronique de Paris (EDITE)

Habilitation Thesis

in Computer Science

presented by Frédéric GARDI

Toward a mathematical programming solver based on local search

publicly defended on November 25th 2013 before the jury consisting of

Philippe BAPTISTE	Strategy & Innovation Director, MESR/DGRI (Examiner)
Yves CASEAU	Executive Vice President, Bouygues Telecom (Examiner)
Philippe CHRÉTIENNE	Professor, Université Pierre et Marie Curie (Examiner)
Gérard CORNUÉJOLS	Professor, Carnegie Mellon University (Examiner)
Michel GENDREAU	Professor, École Polytechnique de Montréal (Referee)
Jin-Kao HAO	Professor, Université d'Angers (Referee)
Geir HASLE	Chief Research Scientist, SINTEF ICT (Referee)
Marc SEVAUX	Professor, Université de Bretagne-Sud (Examiner)

UPMC
SORBONNE UNIVERSITÉS

LABOR OMNIA VINCIT IMPROBUS[1]

In memory of Max Gardi, my father.

If you want to go fast, go alone.
If you want to go far, go together.

African proverb.

[1] *Hard work conquers all.* Virgil (*Georgics*, Book I, lines 145–146).

LABOR OMNIA VINCIT IMPROBUS [2]

À la mémoire de Max Gardi, mon père.

Si tu veux aller vite, pars seul.
Si tu veux aller loin, pars avec d'autres.

Proverbe africain.

[2] *Un travail opiniâtre vient à bout de tout.* Virgile (*Géorgiques*, Livre I, vers 145–146).

Remerciements

Afin de faciliter sa diffusion, j'ai souhaité écrire ce mémoire en anglais, ou disons plutôt ce que l'on appelle "anglais international" lorsque l'on ne le maîtrise pas, comme moi. Que les amoureux de la langue française m'en excusent. Aussi, que les lecteurs m'autorisent ces quelques mots de remerciements dans ma langue, le français.[3]

En premier lieu, je tiens à remercier Monsieur Michel Gendreau, professeur au sein du département de mathématiques et de génie industriel de l'École Polytechnique de Montréal et chercheur au Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport (CIRRELT), Monsieur Jin-Kao Hao, professeur au sein du département d'informatique de l'Université d'Angers et directeur du Laboratoire d'étude et de recherche en informatique d'Angers (LERIA), et Monsieur Geir Hasle, scientifique en chef au sein du département de mathématiques appliquées du SINTEF ICT (Oslo, Norvège) et professeur associé en optimisation et recherche opérationnelle à l'Université de Jyväskylä (Finlande), d'avoir accepté d'être les rapporteurs de ce mémoire. J'ai conscience de la tâche importante que cela représente. Leur planning, certainement déjà saturé en cette fin d'année, a visiblement été robuste face à cet aléa ; pour cela, je les en remercie encore une fois. De même, je remercie chaleureusement l'ensemble des membres du jury pour m'avoir fait l'honneur d'assister à la soutenance de cette habilitation à diriger des recherches. Je citerai ici les examinateurs : Monsieur Philippe Baptiste, chef du service de la stratégie et de l'innovation de la Direction générale pour la recherche et l'innovation (DGRI) du ministère de l'Enseignement supérieur et de la Recherche (MESR) ; Monsieur Yves Caseau, directeur général adjoint Technologies, Prospective et Innovation et membre du Comité de direction générale de Bouygues Telecom ; Monsieur Philippe Chrétienne, professeur émérite d'informatique à l'Université Pierre et Marie Curie (UPMC) à Paris ; Monsieur Gérard Cornuéjols, *IBM University Professor of Operations Research* au sein de la *Tepper School of Business* de l'Université Carnegie Mellon (Pittsburgh, États-Unis) et professeur associé au département d'informatique de l'Université Aix-Marseille ; Monsieur Marc Sevaux, professeur à l'université de Bretagne-Sud à Lorient. Je remercie particulièrement Monsieur Philippe Chrétienne pour avoir eu la gentillesse de m'éclairer et m'aiguiller dans mon parcours vers l'habilitation à diriger des recherches.

Je me souviens de mes premiers pas en informatique à la Faculté des Sciences de Luminy de l'Université Aix-Marseille puis comme ingénieur chez Prologia, petite entreprise

[3]To facilitate its dissemination, the dissertation was written in English, or rather say in "international English" as called by people who do not master it, like me. Lovers of the French language, please accept my apologies. Then, I ask to the readers to allow me these few words of acknowledgements in my language, the French.

marseillaise connue pour avoir commercialisé le premier langage Prolog au début des années 80. Je ne citerai pas ici toutes les personnes que j'ai rencontrées là-bas, qui m'ont appris beaucoup et influencé mon parcours. Il y a ceux qui m'ont enseigné la discipline, ceux qui m'ont donné le goût de la recherche, ceux qui m'ont donné le goût de la pratique. Je les remercie tous. Ils se reconnaîtront à la lecture de ces lignes. Certains ont contribué aux trois : Michel Van Caneghem, qui supervisa mes travaux de la maîtrise au doctorat et continua à m'encourager au-delà, et Alain Colmerauer, ancien directeur du Laboratoire d'informatique de Marseille dont le parcours scientifique m'inspira grandement. Cela est suffisamment rare et remarquable pour que je les en remercie doublement. Je pense également à Victor Chepoi et Yann Vaxès, qui ont éveillé ma curiosité pour les mathématiques discrètes, notamment la théorie des graphes et leur algorithmique. Enfin, je remercie Alain David, aujourd'hui président de Prologia, pour la confiance qu'il m'a accordée à cette époque où ma fougue de jeune scientifique aurait pu se heurter à la dure réalité économique des projets qu'il conduisait.

Je me permettrai seulement une anecdote à propos de cette période d'apprentissage. Alors que bien classé à la sortie du DEA d'informatique je pensais bénéficier d'une bourse ministérielle, un fâcheux concours de circonstance me poussa à financer ma thèse de doctorat autrement, en travaillant à temps partiel comme ingénieur en optimisation (et plus largement en informatique) pour la société Prologia. Jamais injustice ne m'aura été autant bénéfique ! Bien que très éprouvante, cette expérience – un pied dans le monde de l'entreprise et l'autre dans le milieu de la recherche publique – a été extrêmement enrichissante et m'a profondément marqué. Sans cela, je n'aurais probablement pas suivi la voie que j'ai suivie, et le projet LocalSolver que j'évoque dans ce mémoire, aventure formidable tant sur le plan scientifique, technologique, qu'humain, n'aurait probablement jamais vu le jour. J'encourage donc les plus jeunes à "diversifier leur recherche", comme on dit en optimisation combinatoire : rester cantonner au sein d'une région de l'espace des solutions – l'espace des possibles – est fortement sous-optimal pour la recherche académique, pour le tissu socio-économique, et surtout pour vous ! C'est aussi une des raisons pour laquelle je me suis engagé au sein de diverses structures d'animation scientifique : œuvrer modestement, à mon niveau, avec mes moyens, au décloisonnement de la recherche (entre ingénieurs et chercheurs, entre public et privé, entre écoles et universités, entre disciplines). J'en profite donc pour saluer le bureau de la Société française de recherche opérationnelle et d'aide à la décision (association ROADEF), la section 6 du Comité national du CNRS, ainsi que les comités scientifique et exécutif du GDR RO, avec qui j'ai grand plaisir à travailler et échanger sur ces sujets.

En 2007, après dix années passées à Marseille, je me décidais enfin à "monter à la capitale" : j'étais recruté au Bouygues e-lab, département de recherche et développement de Bouygues SA, maison mère du Groupe Bouygues. Je ne remercierai jamais assez Thierry Benoist, chef du service Optimisation, Etienne Gaudin, directeur du e-lab, ainsi que Monsieur Gilles Zancanaro, directeur central Comptabilité et Système d'Information-Finance, et Monsieur Alain Pouyat, directeur général Informatique et Technologies Nouvelles, pour m'avoir fait confiance. Car je n'ai pas été déçu du voyage : j'ai trouvé là un mélange unique d'ingénieurs, de chercheurs, d'entrepreneurs. Ces années passées au Bouygues e-lab ont naturellement continué à me transformer. Je salue tous mes collègues du e-lab, actuels et anciens. J'ai une pensée particulière pour quelques uns avec qui j'ai eu l'occasion de collaborer de très près : Nicolas Braud (l'entrepreneur self-made-man incarné, le "Lapin Duracell" du e-lab, dont je ne cesse de m'inspirer en matière de relation commerciale), Étienne Gaudin (dont j'ai beaucoup appris de la gestion calme et posée de certaines situations électriques), Antoine Jeanjean (qui m'a transmis sa passion pour l'innovation, son esprit d'entreprise, et qui après maintes péripéties dont une thèse de doctorat est resté un ami), Bruno Martin (qui m'a fait comprendre que pragmatisme ne s'opposait pas à rigueur scientifique), Guillaume Rochart (au contact de qui j'ai réalisé que je m'y connaissais peu en informatique).

Aujourd'hui, une nouvelle page se tourne avec le lancement commercial de notre logiciel LocalSolver à l'international et la création de la filiale Innovation 24 qui portera nos activités de services en recherche opérationnelle et aide à la décision au-delà du Groupe Bouygues. Une nouvelle aventure rendue possible par l'engagement farouche de quelques uns : Thierry Benoist, que je pourrais maintenant qualifier de vieux compagnon de route, auprès de qui j'ai beaucoup appris, bien au-delà des aspects techniques, et qui m'a fait prendre conscience de ce qui fait la force du Polytechnicien (confirmant que non, je n'aurais jamais pu l'être !) ; Julien Darlay, athlète complet de l'informatique et des mathématiques, recruté avant même la fin de sa thèse de doctorat, dont l'algorithme du simplexe et les "relaxations" en tous genres feront parler de lui dans quelques années ; Romain Megel, à la culture générale digne d'un énarque (une chance qu'il se soit contenté de programmer des ordinateurs), une des rares personnes capables de m'épuiser dans un débat politique (je blague), mais surtout l'un des meilleurs informaticiens français de sa génération (je ne blague pas). Je remercie chaleureusement notre direction ainsi que la direction générale de Bouygues SA de nous avoir écoutés, de nous avoir compris, et de soutenir ce projet d'entreprise. La culture entrepreneuriale chez Bouygues n'est pas une légende.

Bien entendu, je n'oublie pas Bertrand Estellon qui développe LocalSolver avec nous, ainsi que Karim Nouioua qui a quitté l'équipe en 2012 pour de malheureuses raisons que je n'évoquerai pas ici. J'ai connu Bertrand et Karim sur les bancs de la faculté de Luminy il y a une dizaine d'années. En dépit des parcours disparates de chacun, nous n'avions cessé de collaborer. Que dire de plus. Je citerai simplement le paragraphe que j'avais écris à leur propos dans les remerciements de ma thèse de doctorat, en mai 2005 :

« [...] Bertrand Estellon (dernier arrivant, mais non des moindres; avec qui les débats sur le métier sont toujours animés) [...] Karim Nouioua (sans doute "le meilleur d'entre nous" et je ne plaisante pas) [...] Je me dois d'ailleurs de m'attarder quelque peu sur ce dernier: Karim Nouioua. Je l'ai rencontré alors que je terminais ma première année de thèse. Plus qu'un collègue de travail, c'est devenu un ami. Un ami avec qui j'ai passé tant de soirées à discuter "optimisation" dans notre bureau du 6ème (« $\mathcal{P} = \mathcal{NP}$? »). Un ami qui m'a poussé il y a maintenant plus d'un an à participer avec lui au Challenge ROADEF'2005, en sus de nos recherches respectives. « Pour voir si nous sommes des "chercheurs opérationnels" ! », me disait-il. Ce Challenge, nous l'avons gagné avec Karim, mais aussi Bertrand, dont l'arrivée en cours de route fut décisive. Un article est maintenant en cours d'écriture. Cela restera un de mes plus beaux souvenirs de ces trois années de Doctorat. De grosses rigolades. De rudes soirées (voire nuits) passées à programmer. En définitive, de sacrés bons moments. Merci Bertrand, merci Karim. »

Même si j'apprécie discuter avec de nombreux collègues, et au-delà de mes mentors, je

n'ai collaboré de façon étroite sur le plan scientifique qu'avec un petit nombre de personnes. Tous ont été cités ci-dessus : Thierry Benoist, Julien Darlay, Bertrand Estellon, Antoine Jeanjean, Romain Megel, Karim Nouioua. Tous les travaux que je présente dans ce mémoire ont été réalisés en collaboration avec eux. Je les remercie encore une fois pour ces bons et fructueux moments passés ensemble. J'espère que cela va continuer longtemps, nous avons encore tant de choses à faire ! Je n'oublie pas les jeunes ingénieurs et chercheurs qui, alors stagiaires au Bouygues e-lab, ont œuvré à nos côtés sur le projet LocalSolver et tant d'autres: Thibaud Cavin, Lucile Robin, Saul Pedraza Morales, Sofia Zaourar, Boris Prodhomme, Clément Pajean.

Pour conclure, je remercie bien sûr ceux sans qui rien de tout cela n'aurait été possible, ceux envers qui j'ai une dette imprescriptible : mes parents Max et Michèle. J'ai notamment une pensée émue pour mon père qui nous a quitté il y a peu. Il nous manque terriblement. Récemment, il m'avait demandé pourquoi je ne créais pas mon entreprise à présent. Même si j'avais répondu de façon évasive, j'étais content qu'il me pose cette question (pour ne pas dire un peu fier). Même si Innovation 24 n'est pas *mon* entreprise, il se serait certainement passionné pour cette rude mais belle aventure. Je salue mon jeune frère Romain, que je suis heureux de voir s'épanouir dans un tout autre domaine. Je remercie également ma tante Christiane ainsi que “Pépé et Mémé” Bonnet, qui m'ont toujours encouragé. Je n'oublie pas ma grand-mère Odette, qui me demandait à qui et à quoi servaient tous ces petits dessins (des graphes) que je gribouillais sur mes cahiers. Ce sont deux excellentes questions, que tous les chercheurs devraient se poser régulièrement. Elle me manque elle aussi. Je salue toute la famille de ma compagne Sabine, les “De Broche Des Combes”, en particulier ses parents Agnès et Jean-François, chez qui une partie de ce mémoire a été écrit : “Quoi ! Il travaille encore !”. Enfin, que tous mes amis sachent que je pense à eux.

Mes derniers mots vont à celle qui partage ma vie et m'apporte tant de bonheur depuis plus de dix ans : Sabine, “mon Panda”. La dette que j'ai envers elle est encore prescriptible, mais j'ai peur de ne pouvoir l'honorer tant elle s'est alourdie au fil de ces années. Je ne la remercierai jamais assez pour tout ce qu'elle m'a appris et permis de comprendre, pour l'intérêt qu'elle porte à mon métier et mes travaux, pour ses encouragements constants, et surtout pour sa patience. Car depuis toutes ces années, Sabine compose avec cette passion qui m'agite. Travailler, travailler, encore et toujours travailler... De longues journées, qui débordent sur nos soirées, nos week-ends, nos vacances, jusqu'au somnambulisme la nuit ! “Tu n'es pas raisonnable, tu ne sais pas t'arrêter. Tu vas te rendre malade !”. D'ici à quelques mois, nous serons trois. Il est donc temps de retrouver raison et de se calmer. Sabine, cette fois c'est promis, j'arrête mes conneries.

À Paris, le 16 novembre 2013

F. G.

Seuls les bons professeurs forment les bons autodidactes.

Jean-François Revel (1924–2006), philosophe français,
membre de l'Académie française, né à Marseille.

De : Antoine JEANJEAN
Envoyé : lundi 16 juillet 2007 17:04
À : Frédéric GARDI; Guillaume ROCHART; Thierry BENOIST
Objet : LocalSolver : compte rendu journée 1

Voici quelques notes prises suite à notre première journée (n'hésitez pas à corriger ou à ajouter des choses).

Le but du solveur basé sur la recherche locale serait de résoudre des problèmes d'optimisation mathématique posés de manière abstraite, afin d'optimiser un objectif sous contraintes, sans faire une recherche complète de solutions. Les problématiques qui nous intéressent sont des problèmes industriels, avec beaucoup de variables de décisions, binaires dans un premier temps, peut-être des variables avec domaines dans un second temps. Notre volonté est de se comparer à des solveurs existants (CPLEX, XPRESS, COIN, GLPK, PPC, SAT, ...) en se basant sur des benchmarks tels que ceux de la CSPLib, de la OR Library, etc. Pour cela, nous nous concentrerons sur trois aspects : la stratégie de recherche, les transformations locales et l'algorithmique d'évaluation. Ces trois aspects étant très liés, nous tâcherons de veiller à leur bonne adéquation. Le langage choisi semble être le C++.

Les transformations. Il est important de randomiser les transformations. On pourra réaliser de l'apprentissage sur les résultats de chacune de ces transformations. Ainsi, au fur et à mesure du déroulé de la recherche, on utilisera à chaque itération les meilleures transformations pour optimiser et diversifier la recherche. Voici quelques exemples de transformations que nous avons commencé à étudier :

- Le Flip qui consiste à échanger la valeur d'une variable par son complément.

- Le Swap qui consiste à échanger les valeurs de deux variables.

- Les 3-Opt, ..., k-Opt qui consistent à modifier les valeurs de k variables.

Ces transformations se déclineront en trois types : les intra contraintes, les inter contraintes et celles entre variables de la même "famille". Qu'est-ce qu'une famille ? Une famille de variables permet de décrire un ensemble de variables qui ont des propriétés communes. On peut imaginer que ces familles soient définies par l'utilisateur lui-même. La stratégie du moteur pourrait être aussi de segmenter lui-même les familles, puis d'ordonner les variables au sein d'une famille et ensuite d'apprendre les relations entre variables. Ensuite, on travaillera sur les variables dont les contraintes sont les plus dures.

Abstract

This dissertation deals with *local search for combinatorial optimization* and its extension to mixed-variable optimization. Our goal is to present local search in a new light. Although not yet understood from the theoretical point of view, local search is the paradigm of choice to tackle large-scale real-life optimization problems. Today end-users ask for interactivity with decision support systems. For optimization software, it means obtaining good-quality solutions quickly. Fast iterative improvement methods, like local search, are suited to satisfy such needs.

When a solution space is gigantic, a complete search becomes unpractical. Given a (possibly infeasible) solution to the problem, local search consists in modifying some parts of this one – that is, some decision variables – to reach a new, hopefully better solution. Exploring a so-called *neighborhood* of the incumbent has a major advantage: the new solution can be evaluated quickly through *incremental calculation*. Then, local search can be viewed as an incomplete and nondeterministic but efficient way to explore a solution space.

First, an iconoclast *methodology* is presented to design and engineer local search algorithms. We show that the performance of a local search mainly relies on the richness of the neighborhoods explored, as well as on the efficiency of their exploration. Ultimately, implementing high-performance local search algorithms is a matter of expertise in *incremental algorithmics* and of dexterity in computer programming. Our concern to *industrialize* local search approaches shall be of a particular interest for practitioners. As examples, this methodology is applied to solve two industrial problems with high economic stakes.

Nevertheless, software applications based on local search induce extra costs in development and maintenance in comparison with the direct use of mixed-integer linear programming solvers. We present the *LocalSolver project* whose goal is to offer the power of local search through a model-and-run solver for *large-scale 0-1 nonlinear programming*. Having outlined its modeling formalism, the main ideas on which LocalSolver relies are described and some benchmarks are presented to assess its performance. We conclude the dissertation by presenting our ongoing and future works on LocalSolver *toward a full mathematical programming solver based on neighborhood search*.

Title: Toward a mathematical programming solver based on local search.
Keywords: combinatorial optimization, local/neighborhood search, mathematical programming.

Résumé

Ce mémoire traite de la *recherche locale en optimisation combinatoire* et de son extension à l'optimisation en variables mixtes. Notre but est de présenter la recherche locale sous un jour nouveau. Bien qu'encore mal cernée d'un point de vue théorique, la recherche locale est un paradigme de choix pour attaquer les problèmes d'optimisation de grande taille tels que rencontrés en pratique. Les utilisateurs demandent toujours plus d'interactivité avec les systèmes d'aide à la décision. Pour les logiciels d'optimisation, cela signifie obtenir de bonnes solutions vite. Les méthodes d'amélioration itérative véloces, comme la recherche locale, sont adaptées à ces besoins.

Lorsqu'un espace de solutions est gigantesque, une recherche complète devient impraticable. Étant donnée une solution (possiblement infaisable) au problème, une recherche locale consiste à modifier des parties de celle-ci – c'est-à-dire des variables de décisions – pour atteindre une nouvelle solution que l'on espère meilleure. Explorer un *voisinage* de la solution courante a un avantage majeur: la nouvelle solution peut être évaluée rapidement par *calcul incrémental*. Ainsi, la recherche locale peut être vue comme une façon incomplète et non déterministe mais efficace d'explorer un espace de solutions.

Tout d'abord, une *méthodologie* iconoclaste est présentée pour la conception et l'ingénierie d'algorithmes de recherche locale. Nous montrons que la performance d'une recherche locale repose essentiellement sur la richesse des voisinages explorés, ainsi que sur l'efficacité de leur exploration. En définitive, l'implémentation d'algorithmes de recherche locale performants est une question d'expertise en algorithmique incrémentale et de dextérité en programmation informatique. Notre souci d'*industrialiser* la recherche locale intéressera particulièrement les praticiens. À titre d'exemples, cette méthodologie est appliquée à la résolution de deux problèmes industriels aux enjeux économiques importants.

Néanmoins, les logiciels basés sur la recherche locale induisent des coûts additionnels de développement et de maintenance en comparaison avec l'utilisation directe de solveurs de programmation linéaire en nombres entiers. Nous présentons le *projet LocalSolver* dont le but est d'offrir la puissance de la recherche locale à travers un solveur *model-and-run* pour la *programmation non linéaire 0-1 à grande échelle*. Après une brève revue de son formalisme de modélisation, les idées maîtresses sur lesquelles reposent LocalSolver sont décrites et un échantillon de ses résultats est donné. Nous concluons le mémoire par une présentation de nos travaux en cours et futurs sur LocalSolver *vers un solveur de programmation mathématique complet basé sur une recherche par voisinage.*

Titre: Vers un solveur de programmation mathématique basé sur la recherche locale.

Mots-clés: optimisation combinatoire, recherche locale/par voisinage, programmation mathématique.

Contents

Chapter 1

Introduction

In this dissertation, we present a survey of our research on local search in combinatorial optimization, from methodological foundations to industrial applications and software. This survey puts into perspective the evolution of our research from the resolution of specific problems to the design of a general-purpose mathematical programming solver based on local search, namely LocalSolver. After a brief introduction about combinatorial optimization and local search, we outline the plan of the dissertation.

1.1 Local search in combinatorial optimization

A *combinatorial optimization* problem is characterized by the discrete nature of its solution space. In very general mathematical terms, such a problem consists in optimizing (minimizing or maximizing) a function f over a finite set $\mathcal{S}$ of solutions. Many organizational problems arising in business and industry can be modeled in these terms. When the cardinality of $\mathcal{S}$ is gigantic, enumerating all the solutions of the space to find the best one is impossible within reasonable running times. A scientific field, namely combinatorial optimization, has grown in the last fifty years whose aim is to study this kind of problems and to provide practical algorithmic solutions to these ones.

Despite positive results, numerous combinatorial optimization problems remain difficult. First, difficult in the sense of complexity theory: it is unlikely that an algorithm exists solving any problem in worst-case polynomial time. Then, difficult because many algorithms efficient according to the theory are not so efficient in practice: $O(n^3)$-time or even $O(n^2)$-time algorithms become unusable face to the actual needs. That is why a vast literature has grown around approximate algorithms, also called heuristics, which do not guaranty to output optimal solutions. This is in this context that *local search* is generally presented. Before being a heuristic approach, local search is an optimization paradigm, referred as *iterative improvement* in the literature. This paradigm is widely used in combinatorial optimization but also in continuous optimization in different terms. Conceptually, the idea is simple and natural. Given a (possibly infeasible) solution to the problem, local search consists in modifying some parts of this one – that is, some of the decision variables – to reach a new, hopefully better solution. Such a modification is generally called *move* or *transformation*. Exploring a so-called *neighborhood* of the incumbent has a major advantage: the quality of the new solution can be evaluated quickly through *incremental calculation*. Then, local search can be viewed as an incomplete and nondeterministic but efficient way to explore a solution space, in contrast to complete search consisting in enumerating all solutions.

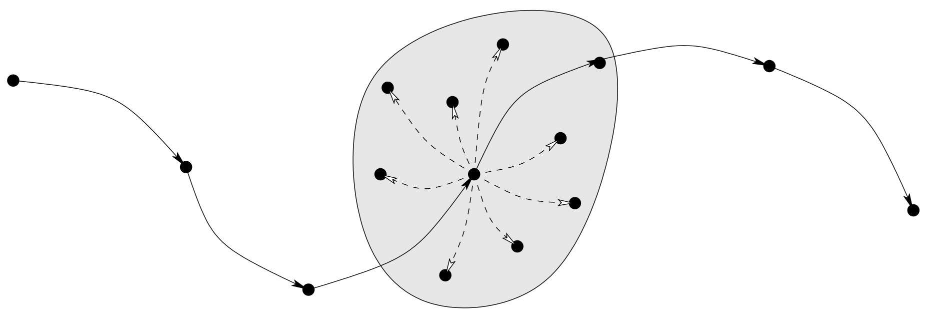

Figure 1.1: Local search = neighborhood search, that is, searching for better solutions in neighborhoods of an incumbent solution.

Local search appears hidden behind many textbook algorithms from the simplest one like bubble sort to more complex ones like maximum flow or maximum matching algorithms (see [1, p. 24] for more details). The simplex algorithm for linear programming is probably the most famous local search algorithm, as a pivoting-based combinatorial approach to a continuous optimization problem. It is also the best example of our little understanding of local search algorithms: widely used by practitioners because empirically efficient, but frustrating for theorists due to invariably pessimistic worst-case results. What a puzzling situation: despite being the star of mathematical programming, the simplex algorithm is not efficient according to complexity theory. Now the situation is changing: dramatic progresses in the understanding of heuristic algorithms have been done during the last decade with the rise of smoothed complexity [112], providing as first striking result an explanation of why the simplex algorithm usually takes polynomial time.

On the other hand, the concept of metaheuristic [50] – simulated annealing, tabu search, ant colony optimization, genetic algorithms, etc. [51] – has progressively supplanted the basic concept of local search. First published at the end of the nineties, the survey by Aarts and Lenstra [1] is the first and last bestseller using the term “local search” in its title. Now thousands of papers are published each year presenting new metaheuristics, new applications of metaheuristics, comparisons of metaheuristics. Too much of these papers have a poor technical content as well as a low practical impact [110]. Unfortunately, this phenomenon has contributed to the idea that local search is not a way to make “serious” optimization, in particular among mathematicians.

1.2 Mathematical programming

There are different views of what is *mathematical programming*. In the broad sense, it can be seen as an old wording for mathematical optimization. On the other hand, mathematical programming also refers to the discipline concerned by the design of algebraic languages to *model* declaratively optimization problems, as well as by the design of general-purpose algorithms to *solve* these mathematical models. This discipline mixes both scientific and technological aspects: mathematics, computer science and software engineering. Nevertheless, our view of mathematical programming goes beyond. Math programming software, usually called *solver*, is essentially built to deserve people who need to solve *practical* optimization problems. This implies to carefully listen and understand the needs of these people, generally referred as *practitioners* or simply *users*. Common to all branches of applied mathematics, this latter aspect is critical to the success of the discipline: the dissemination of its outcomes in business and industry, and more generally in the whole society.

Mathematical programming is divided into two main fields: *continuous optimization* and *discrete optimization*. Historically, these fields do not share so much: different mathematical foundations, different algorithmic techniques, different application areas, different scientific journals and conferences. Even if there is a trend in this direction, rare are the researchers and the practitioners involved in both fields. Linear programming can be viewed as the historical and scientific bridge between the two fields.

Continuous optimization, also called numerical optimization, is traditionally the domain of (applied) mathematicians. Applications of continuous optimization can be found in almost all branches of engineering: mechanics, design, process control, electricity, energy, finance. The algorithmic techniques used in continuous optimization mainly rely on iterative improvement based on derivatives – first or second-order, exact or approximate – to guide the search (see [79, pp. 96–288]). Beyond linear programming, convex optimization has emerged as the largest class of problems for which efficient algorithms can be designed from both theoretical and practical points of view: interior point algorithms [10, 117]. But in practice, approximating problems through accurate convex models is not always possible. Then, the development of non-convex programming solvers remains a hot topic. Several heuristic techniques have been designed to handle real-world nonlinear programs [71]: interior point methods, sequential linear and quadratic programming, augmented lagrangian methods, or direct search also known as derivative free methods. For more details on this topic, the reader is invited to consult [9, 10, 21, 70, 79, 86].

Combinatorial optimization is historically the field of operations researchers and computer scientists. Most of the applications can be found in management science: transportation and logistics, manufacturing, workforce scheduling and timetabling, network planning, etc. Tree search techniques – consisting in enumerating and pruning partial solutions, in particular by exploiting linear relaxations [33] – are massively used by practitioners through mixed-integer linear programming solvers. Since many real-world problems are very large and highly combinatorial, heuristic techniques have been developed to provide good-quality solutions in reasonable running times. Iterative improvement techniques like local search are particularly used in practice. Some attempts have been done to exploit direct local search approaches inside general-purpose solvers [2, 32, 87, 97]. But to the best of our knowledge, no effective and widely-used solver has emerged so far based on the power of local search. Nevertheless, tree search solvers integrate more and more heuristic ingredients to speed up the search for good primal feasible solutions [74].

Until recently, mixed-variable optimization – involving both discrete and continuous decisions – was more studied by people working in combinatorial optimization. Mixed-variable nonlinear programming now becomes the focal point of the two communities. General-purpose solvers are now available to tackle this very general class of optimization problems, or some interesting subclasses like mixed-integer convex quadratic problems. For recent surveys on techniques and software in this area, we refer the reader to [12, 23, 24, 35]. However, these solvers are not widely used by practitioners since only small or specific problems can be efficiently addressed. Therefore, mixed-variable non-convex optimization appears as the next challenge in mathematical programming.

1.3 Outline of the dissertation

This dissertation deals with *local search for combinatorial optimization* and its extension to mixed-variable optimization. Our goal is to present local search in a new light. Although not yet understood from the theoretical point of view, local search is the paradigm of choice to tackle large-scale real-life optimization problems. Today end-users ask for interactivity with decision support systems. For optimization software, it means obtaining good-quality solutions quickly. Fast iterative improvement methods, like local search, are suited to satisfy such needs.

First, an iconoclast *methodology* is presented to design and engineer local search algorithms. We show that the performance of a local search mainly relies on the richness of the neighborhoods explored, as well as on the efficiency of their exploration. Ultimately, implementing high-performance local search algorithms is a matter of expertise in *incremental algorithmics* and of dexterity in computer programming. Our concern to *industrialize* local search approaches shall be of a particular interest for practitioners. As examples, this methodology is applied to solve two industrial problems with high economic stakes.

Nevertheless, software based on local search induces extra costs in development and maintenance in comparison with the direct use of mixed-integer linear programming solvers. We present the *LocalSolver project* whose goal is to offer the power of local search through a model-and-run solver for *large-scale 0-1 nonlinear programming*. Having outlined its modeling formalism, the main ideas on which LocalSolver relies are described and some benchmarks are presented to assess its performance. We conclude the dissertation by presenting our ongoing and future works on LocalSolver *toward a full mathematical programming solver based on local search.*

Chapter 2

Local search: methodology and industrial applications

In this chapter, we present our methodology for tackling combinatorial or even mixed-variable optimization problems by local search. This methodology is designed to *industrialize* the engineering of local search heuristics, especially to solve large-scale combinatorial problems encountered in real-world situations. It may be viewed as iconoclast by some readers since it is not focused on “meta” aspects. In a sense, we advocate for a back to basics. Having exposed this methodology, we illustrate it through two challenging industrial applications: car sequencing for painting and assembly lines (combinatorial optimization), vehicle and inventory routing (mixed-variable optimization).

2.1 Our methodology: back to basics

As presented in introduction, local search is an iterative improvement technique used by practitioners since the premises of combinatorial optimization. This is an approach of choice to tackle large-scale combinatorial problems, as encountered in the practice of operations research (OR). Unfortunately, we have no theoretical foundations yet which explain the good empirical results obtained by local search, contrasting with the bad behaviors predicted by worst-case studies. Then, local search is usually viewed as “cooking” by many practitioners and researchers. This view was considerably amplified with the rise of metaheuristics these last 20 years. Mainly built on bio-inspired metaphors, the zoo of metaheuristics obscures local search today. This tendency can be observed in the literature related to combinatorial optimization, but also in its teaching and its practice.

Many papers have been published describing methodologies or good practices to engineer local search heuristics. These methodological works are essentially concentrating on search strategies and more particularly on metaheuristics (see for example [58, 75, 90]): how to choose the good ones? how to tune them? We claim that this trend, namely “local search = metaheuristics”, has resulted in bad engineering practices. Much time is spent in tuning parameters of metaheuristics. Still worse, much time is spent in trying different metaheuristics for a same problem. By increasing the cost of optimisation software instead of lowering it, such practices go against the trend in information technology – better, faster, cheaper – and then against the spreading of OR in business and industry. If we get back to the seminal papers on local search, we can notice that the emphasis was not primarily put on the search strategy (now called metaheuristic), but more on the moves and their evaluation. For instance, a great part of the works done about the Lin-Kernighan heuristic for the traveling salesman problem concerns the speedup of the evaluation of the celebrated k-opt moves through incremental algorithmics.

Here we propose a methodology to design and engineer local search heuristics for combinatorial or mixed-variable optimization. Our goal is to provide a simple recipe to help practitioners delivering quality, fast, reliable and robust solutions to their clients, while lowering their development and maintenance costs as well as the risks on their optimization projects. This methodology is based on about fifteen years of works in enterprise spent in solving real-life problems for operational users [14, 40, 41, 42, 48, 49, 67]. The concrete nature of our experiments coupled with the business context was crucial in the design of this methodology: we insist on the fact that its purpose is to *industrialize the development of local search solutions for practical combinatorial optimization*.

What are the needs in business and industry?

After many years of practice of operations research, we learnt important lessons about the needs of users in business and industry. First, clients have optimization problems, and rarely satisfaction problems. The "no solution found" answer is not an acceptable answer for end-users. Indeed, they already have solutions in operations, even if these ones may be bad (objectives poorly optimized, important constraints violated). Thus, once the optimization model is stated, finding a feasible solution should be easy. A good way to proceed is to adopt a *goal programming* approach [68]: relaxing some constraints by introducing slack variables, then minimizing the violation of these constraints as primary objective. Even when a solution violating such "soft" constraints is unacceptable, seeing it is very useful for the user to detect the cause of the problem, which is almost always an inconsistency in input data.

Then, optimal solutions is not what clients really want first. Proof of optimality is much less what they want. They want first a nice and easy-to-use software providing good solutions quickly. Visualizing and modifying solutions, as well as interacting with the optimizer, is often more critical than the quality of results. Furthermore, clients are more sensible to what we call local optimality. If the solution output by the optimizer can be improved by hand (generally by mimicking local search), then the client surely consider this one as bad. Surprisingly, it will be the case even if the original solution is close to a global optimum. On the other hand, optimal solutions are sometimes rejected because their structure is too far from the ones commonly known by the user, or just too complicated to understand for him. Ultimately, the trust in solutions is critical for the end-users and so for the good use of an optimization software. Having completed dozens of optimization projects (more or less successful), we can conclude that proving optimality or gaining the last percents in solution quality are insignificant in comparison with the relevance of the mathematical model, the accuracy of input data, the ergonomics of the whole software and its integration in the information system, as well as the project and change management.

The interested reader is invited to consult our paper [16] for more details on these non scientific but crucial aspects in the practice of OR.

The main ingredients of the recipe

Our approach to local search is primarily *pure* and *direct*. No decomposition is done: the problem is tackled frontally. The search space explored by our algorithm is close to (or even extends) the original solution space. In case of mixed-variable optimization, the combinatorial and continuous parts of the problem are treated together: combinatorial and continuous decisions can be simultaneously modified by a move during the search. By avoiding decompositions or reductions, no solution is lost and the probability to find high-quality solutions is increased. To simplify the software design and then to facilitate maintenance, we avoid hybridization: no particular metaheuristic is used, no tree search technique is used. Then, our methodology is mainly focused on *designing rich neighborhoods ensuring an effective randomized exploration of the search space* and *speeding up the evaluation of these neighborhoods through incremental computation.*

Like any resolution approach, local search is sensible to the way the solution space is designed. Since the main idea of local search is to move from one feasible solution to another by preferably modifying a few decisions only, the search space should be modeled to ensure that such moves can succeed with a high probability. The main cause of failure of a move is the presence of severe constraints in the optimization model: the neighbor reached through the move is infeasible with respect to these constraints. Therefore, local search is not suited to tackle tightly constrained problems. More the search space is constrained, larger must be the neighborhoods explored to hope finding better feasible solutions. However, as explained previously, tightly constrained problems generally result from bad modeling practices. The other cause of failure is related to the combinatorial landscape induced by the objective optimized over the solution space [99]. When the landscape is very rugged, many local optima are encountered during the search, requiring possibly uphill moves to escape them. On the other hand, when the landscape is very flat, the search may be lost over large plateaus (solutions with equal cost), slowing the convergence toward high-quality solutions. Rugged landscapes may result from numerical objectives involving many insignificant digits (which can be viewed as a kind of noise), while flat landscapes generally result from objectives inducing high combinatorial steps. Refining the optimization model to avoid such situations is generally fruitful for both solving and business purposes. Note that neutral walks [8] – moves along solutions with equal cost – also known as plateau search [105] are a good way to diversify the search without uphill moves (see for example our work on the car sequencing problem [40, 41] described in the next section).

Nevertheless, exploring large neighborhoods is the best way to diversify the search and to ensure the convergence toward high-quality local optima. Since exploring such neighborhoods requires more running time, a trade-off between quality and efficiency has to be found. We propose to first explore *small but rich neighborhoods* in order to exploit at best invariants and then benefit of *fast incremental evaluations* (generally running in constant time for each explored neighbor). In this way, a huge number of solutions can be visited within the time limit, which becomes crucial when partially exploring a gigantic solution space: more solutions you explore, more chance you have to find good ones.

Enriching and enlarging neighborhoods

A pragmatical way to proceed in practice is to design neighborhoods step by step – from the cheapest to the largest ones – as the need for quality solutions grow. It allows to reduce the time to deliver the first solutions to the clients, which is critical to the success of real-world optimization projects [16]. First, we advocate the use of a *large variety of small randomized moves*. The neighborhood of the incumbent corresponds to the set of solutions reachable through all possible moves. Denoting by n the size of the solution, the idea is to explore an $O(n^k)$-size neighborhood with small values for k (typically $k = 2$) but with a large constant hidden by the big O notation (see Figure 2.1). To speed up the search while avoiding bias or cycling phenomena, the neighborhood is explored following a *first-improvement fashion* (as soon as a better or equal solution is found, this one becomes the new incumbent) *coupled with randomization*. For example, for a machine scheduling problem, the following standard moves are commonly used: insert a task in the schedule, remove a task from the schedule, move a task in the schedule, swap two tasks of the schedule. These moves can be naturally generalized by considering a block of consecutive tasks and not only one task. If needed, we can go further. By designing larger and richer moves: move sequentially k blocks or exchange cyclically k blocks in the schedule. By designing more specific moves: select tasks randomly, select tasks on the same machine, select tasks on different machines, select tasks along the critical path, select tasks saturating some constraints. Adding more complex moves allows to enlarge the explored neighborhood. Adding more specific moves allows to concentrate the search on more promising parts of the whole neighborhood. In practice, these kinds of rich neighborhoods generally suffice to obtain high-quality solutions quickly: we present strong empirical evidences of this claim in the next sections, derived from our papers [14, 41].

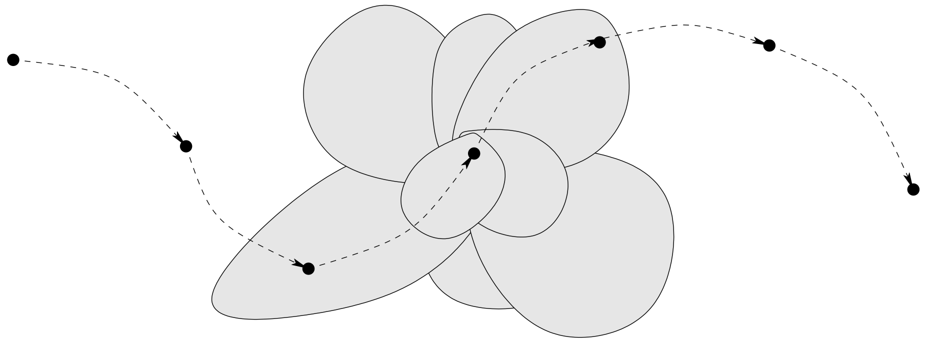
Figure 2.1: Large neighborhood induced by the union of multiple small neighborhoods.

A first way to still enlarge the neighborhood explored is to use *compound moves*. The idea is simple: applying a number of small moves consecutively (or possibly following a backtracking scheme) to find a better feasible solution. The decisions modified at each step of the process are frozen to avoid cycling. This kind of neighborhood exploration allows to jump along infeasible or uphill solutions during the search. This technique is related to the Lin-Kernighan heuristic [61, 62] or to the Ruin & Recreate heuristic [104] when the first move destroys the feasibility of the incumbent. Some applications of this kind of neighborhoods are described in our papers [42, 49]. To still go further, one can explore *exponential-size neighborhoods*. The exploration can be performed using tree search techniques or efficient algorithms in some specific cases (through reduction to polynomial or pseudo-polynomial problems). Such neighborhoods are very time-consuming to explore in comparison to small ones, which make them rarely useful in practice. An application of this kind of neighborhoods as well as a discussion about their practical relevance can be found in our papers [40, 41].

The idea of mixing neighborhoods having different structures and sizes to improve the quality of local optima is not new. This is the main idea behind the variable neighborhood search metaheuristic [58]. During the writing of this dissertation, we discovered that Sörensen et al. [111] recently introduced the term "multiple neighborhood search" to describe this approach, commonly encountered in commercial vehicle routing software. Here we detail some engineering ingredients which make the difference in practice. In particular, the systematic use of randomization during the search is critical for its diversification. Randomization limits the risk of missing out some promising regions of the search space. More generally, randomization helps in avoiding pathological worst-case behaviors.

High-performance software engineering

Incrementality is a core concept in local search. Let S be the current solution and S' the solution to evaluate in the neighborhood of S. Denote by $|S|$ (resp. $|S'|$) the length of S (resp. $|S'|$). Then, denote by Δ the "amount of change" between S and S'. For example, when optimizing a boolean function, Δ represents the number of variables whose value is changing between S and S'. The part of S which remains unchanged in S' is called *invariant*. Evaluating S' incrementally consists in evaluating the cost of S' in a time smaller than $O(|S'|)$, hopefully in $O(\Delta)$ time. For more formal details about incremental computation, the interested reader is referred to [96]. In this way, smaller is the change from S to S' faster can be the evaluation. In particular, if a move is such that Δ remains constant with respect to $|S|$ (that is, $\Delta \ll |S|$), then this move may be evaluated in constant time. Consequently, exploiting invariants helps to speed up the convergence of local search by several orders of magnitude. Besides, careful implementations – aware of the locality principle ruling the cache memory allocation and optimized by code profiling [80, 81, 82] – still helps to accelerate the search (see [120] for some experiments on SAT solvers). Then, it is not surprising to observe an order of 10 or even 100 between the times of convergence of two local search heuristics, apparently based on the same principles. Thus, refining search strategy or tuning metaheuristic parameters is irrelevant while the evaluation machinery is not well optimized. Our experience show that the improvements in terms of solution quality brought by working on search strategy are negligible compared to improvements obtained by enriching the moves and speeding up the evaluation machinery. Ultimately, our local search heuristics are composed of three layers: the search strategy, the pool of moves inducing neighborhoods, the incremental evaluation machinery. The working time spent to engineer each layer during a project follows approximately this distribution: 10 % for the search strategy, 30 % for the moves, 60 % for the evaluation machinery.

Linked to algorithmics, software engineering aspects like reliability are no less crucial than efficiency. Local search is an iterative improvement approach relying on complex algorithms and data structures. Then, engineering local search requires larger efforts in verification and testing than for common software. Consequently, the verification process of local search software has to be industrialized too. Here are summarized our best practices. The first one is to program with assertions [100]. It consists in verifying preconditions, postconditions and invariants all along the program in order to validate each of its steps. Since formal verification remains expensive for basic software projects, the general idea behind programming with assertions is to check the results of a piece of code by a different piece of code. Such a practice reduces drastically the number of coding errors and then the probability of software failures. Concretely, the consistency of all dynamic data structures – in particular the ones used for incremental evaluation of the moves – are checked after each iteration of the local search in debugging mode, by recomputing them from scratch using brute algorithms independent from the local search code. Consequently, a large part of the source code of our local search heuristics is dedicated to verification and testing: from experience, code checkers represent more than 10 % of the whole source code. Hence, reliability aspects, as well as maintainability and portability issues, must be taken into account to cost tightly optimization projects relying on local search.

2.2 Car sequencing for painting and assembly lines

The subject of the ROADEF 2005 Challenge[1] addressed a real-life car sequencing problem proposed by the French automotive company Renault. This problem consists in determining the order in which a set of vehicles should go inside the factory so as to make easier the whole process of fabrication: first through the paint workshop where the car is painted, then along the assembly line where the equipments and options of each vehicle are set. For the paint workshop, the objective is to minimize the number of purges (or color changes), which amounts to group vehicles which have the same color. Nevertheless, the number of consecutive vehicles having the same color must not exceed a certain value, namely the paint limit. Then, in order to smooth the workload on the different stations composing the assembly line, it is necessary to space out the vehicles for which setting options needs some heavy operations. This need of spacing out vehicles is formalized by defining a ratio for each option. For example, for an option to which is associated the ratio 3/7, one hopes to find no more than 3 vehicles requiring the option in any contiguous subsequence consisting of 7 vehicles (such subsequences are called windows). Then, the objective is to minimize the number of violations for all the ratios. In the previous example, if 5 vehicles have the option in a window of 7, then 2 violations are counted. Here the options are classified into two kinds: priority or non-priority. The objective function is composed of three terms which are minimized in lexicographic order: the number of color changes (RAF), the number of violations on priority options (EP), the number of violations on non-priority options (ENP). Three kinds of objectives are possible: EP/ENP/RAF, EP/RAF/ENP and RAF/EP/ENP.

The car sequencing problem is strongly NP-hard, even if only options with ratio 1/2 are considered [39]. The previous works on the subject [54, 55, 69, 95, 108] dealt only with the constraints and objectives related to the assembly line. The brute force use of integer or constraint programming software reaches its limit when one hundred cars with few options are considered, while some instances with one thousand cars have to be tackled. Then,

[1]`http://challenge.roadef.org/2005/en`

several heuristic approaches have been proposed to solve practically the problem: greedy algorithms [54], ant colony optimization [54, 55, 108], local search [54, 95]. All these approaches have been intensively studied and experimented in the context of ROADEF 2005 Challenge [109]. In effect, the heuristics designed by the different competitors to the Challenge are essentially based on local search, integrated into different metaheuristic schemes. Besides, several works have been done to hybridize local search and exact approaches like constraint programming [91, 92] or integer programming [40, 41, 94]. Our extensive works on this subject [40, 41] show that exploring large neighborhoods through matching algorithms in addition to small moves leads to some improvements, but negligible in comparison to the engineering efforts necessary to obtain them.

We outline a local search heuristic which enabled us to win the Challenge in both Junior and Senior categories. The results obtained on the classical or Renault's car sequencing problems remain the state of the art. Our approach follows the methodology described above: a fast exploration of several small but rich neighborhoods. Our search strategy is simply a standard first-improvement descent. The main conclusion drawn of our victory is that sophisticated metaheuristics are useless to solve car sequencing problems. More generally, it demonstrates that algorithmic aspects, often neglected in favor of trendy "meta" aspects, remain the key ingredients for designing and engineering high-performance local search heuristics. For more details on this topic, the reader is referred to our papers [40, 41].

Search strategy and moves

The initial solution is built using a greedy algorithm [41, 54]. We use a first-improvement standard descent as general heuristic. The algorithm applies at each iteration a move to the current sequence. The move is picked randomly in a pool of moves following a nonuniform distribution. The reader is referred to [41] for more details about this distribution, which is not crucial outside a context of competition. If the move induces no violation on paint limit constraints and does not deteriorate the cost of the current solution, then this move is committed. Otherwise, it is rejected.

Four classical moves [54, 95] are used: swap, forward insertion, backward insertion, reflection. A swap consists in exchanging the positions of two vehicles in the sequence. A forward insertion localized on a portion v_i, x, y, z, v_j of vehicles consists in extracting v_j, shifting the vehicles v_i, x, y, z to the right, and reinserting v_j at the position which remains unfilled (the former position of v_i); after the move, the initial portion contains in order the vehicles v_j, v_i, x, y, z. A backward insertion is defined in a symmetric way, by extracting v_i instead of v_j. A reflection between two vehicles v_i and v_j consists in reversing the portion of vehicles between v_i and v_j.

There are $n(n-1)/2$ combinations for choosing the positions where to apply a move. Then, the neighborhood explored through these moves is of size $O(n^2)$, with n the number of cars to sequence. The selection of the neighbor is guided by no sophisticated rule: the first neighbor lowering or even equaling the cost of the current solution is retained for a new neighborhood exploration. Note that accepting the moves which do not strictly improve the cost (that is, neutral moves) is crucial. Coupled with a fast evaluation, this is a way to widely diversify the search [8, 105].

Enriching the moves and boosting their evaluation

The simplest way to choose the positions to apply swaps, insertions or reflections consists in picking randomly the positions i and j. When numerous violations appear, such random moves are effective to minimize the ratio objectives quickly. However, smarter strategies for choosing positions are useful when finding better solutions becomes difficult. More formally, this issue arises when finding an improving neighbor randomly in an $O(n^2)$-size neighborhood asks for $O(n^2)$ time. The idea is to reduce this running time by visiting neighbors (that is, by attempting moves) having a higher probability of success. Such a selection can be made without significant time overheads. These strategies depend on what is the objective to optimize and on which kind of move is applied. For swaps, choosing vehicles sharing some options or having the same color augments naturally the chance of success of the move. In the same way, choosing adjacent positions in the sequence limits the risk of deterioration while making the evaluation faster. For insertions and reflections, a good strategy consists in choosing i and j such that the distance $|j - i|$ is equal to the denominator of one of the ratios. When RAF is the prime objective, choosing the positions i and j as starting or ending point of a sequence of vehicles having the same color limits the chances to break such sequences. All the strategies are detailed in [41].

The bottleneck of each iteration of the local search in terms of time complexity is clearly the evaluation of the move which is attempted. Fortunately, almost all moves reveals some invariants which can be exploited using special data structures to evaluate quickly the impact of a move on the cost of the current solution. For ratios, the crucial remark is that the number of windows which are impacted by swaps, insertions or reflections depends only on the denominator of each ratio, generally small in practice. For swaps, only windows containing the two exchanged vehicles v_i and v_j are perturbed. In the case of insertions, the windows which are entirely contained between the vehicles v_i and v_j are just shifted of one position. Thus, only windows containing v_i and v_j must be considered for the evaluation of insertion moves. The same idea holds for reflections since windows entirely contained between extremal positions i and j are reversed, which lets the number of violations into these ones unchanged. These remarks form the basis of the proofs of the following propositions [41]. For any ratio P_k/Q_k, the number of windows impacted by a swap, an insertion or a reflection is at most $2\,Q_k$. Then, evaluating the new number of violations for this ratio following one of these moves can be done in $O(Q_k)$ time. Detecting some violations of paint limit constraints and evaluating the new number of purges following a swap, an insertion or a reflection can be done in $O(1)$ time.

Since there is no compensation between the objectives EP, ENP, RAF, the order of their evaluation is significant. In many cases, the evaluation process can be stopped before evaluating all the objectives. In the same way, starting the evaluation by checking if the move respects the paint limit constraints is judicious, since the verification takes only $O(1)$ time. The evaluation can be further improved heuristically for ratios. Indeed, the ordering in which the ratios are evaluated is significant. Suppose that having evaluating a move, the number of violations newly created for a subset of ratios is greater than the total number of violations for the remaining ratios (which are not evaluated yet). Even if the number of violations for the remaining ratios falls to zero, the move still deteriorates the sequence. Then, the evaluation can be stopped immediately and the move rejected.

Accordingly, evaluating ratios following the decreasing order of the number of violations is a good heuristic to decide earlier of the rejection of the move. Since the evaluation of ratios is the most time-consuming routine, it still helps to reduce the practical running time of the local search.

Experimental results and discussion

Our local search heuristic was implemented in C programming language (4000 lines of code). The running time was limited to 10 minutes on a standard computer by Renault. Our algorithm, ranked first, obtains the best result for almost all instances of the Challenge. All the data, results and rankings can be found at `http://challenge.roadef.org/2005/en` or in papers [41, 109]. The algorithm greatly improves the results obtained by Renault thanks to an apparently equivalent local search heuristic (simulated annealing coupled with swap moves) but without fast incremental computation. As a result of the competition, Renault deployed our algorithm in his 17 plants worldwide, generating important annual savings.

The number of attempted moves exceeds 150,000 per second, that is 100 millions over the 10 minutes of running time. Almost all these attempted moves are feasible. Here a feasible move means that its application results in a feasible solution (that is, a feasible sequence of cars). This is due to the fact that constraints of the problem are not so hard to satisfy here, inducing a huge solution space. Then, the diversification rate, namely the percentage of accepted moves (that is, which are not rejected), varies between 1 and 10 % according to the instances. For all instances, the percentage of accepted moves remains constant while the objective function is lowered. Such a diversification rate, constant all along the search thanks to neutral moves, allows to diversify widely the search and thus to explore many different regions of the solution space. This point is crucial for the effectiveness of any incomplete search algorithms, especially local search approaches.

Another interesting observation is that exploring larger neighborhoods is useless on this problem. Indeed, it is possible to explore exponential-size neighborhoods efficiently using (polynomial) flow or even assignment algorithms [40, 41]. Even customized and highly optimized [40], such explorations bring no substantial improvement in solution quality as well as in convergence time, even for longer running times. Despite being polynomial, each exploration remains very time-consuming in comparison to the evaluation of small moves like swaps: 100 large moves versus 10 million small moves, per minute. Consequently, the expected running time to improve (the time spent to explore the neighborhood multiplied by the probability to find an improving solution) for these larger moves is of several orders of magnitude higher that the one for small moves. We can conclude that for the car sequencing problem, a basic search strategy coupled with small but enriched moves boosted by fast incremental evaluations is better than sophisticated search strategies, even if based on nice mathematical properties (see Propositions 3.1, 3.3, 3.4 in [40]).

2.3 Vehicle routing with inventory management

Vehicle routing with inventory management refers to the optimization of transportation costs for the replenishment of customers' inventories: based on consumption forecasts, the vendor organizes delivery routes. For the sake of concision, only the main features of the problem are outlined. A product is produced by the vendor's plants and consumed at customers' sites. Both plants and customers store the product in tanks. Reliable forecasts of consumption (resp. production) at customers (resp. plants) are available over a discretized short-term horizon. The inventory of each customer must be replenished by tank trucks so as to never fall under its safety level. The transportation is performed by vehicles formed by coupling three kinds of heterogenous resources: drivers, tractors, trailers. Each resource is assigned to a depot. Now, a solution to the problem is given as a set of shifts. A shift is defined by: the depot (from which it starts and ends), the vehicle (driver, tractor, trailer), its starting date, its ending date, and the chronological-ordered list of performed operations. Then, an operation is defined by the site where the operation takes place, the quantity delivered or loaded, its starting and ending dates. Thus, the inventory levels for customers, plants and trailers can be computed from the quantities delivered or loaded during the shifts.

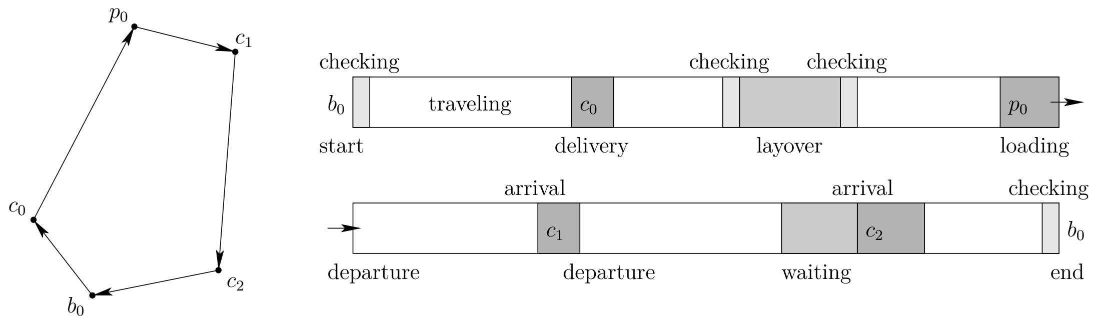

Figure 2.2: Two views of the shift $s = (b_0, c_0, p_0, c_1, c_2, b_0)$: the route and the schedule.

The constraints on shifts are called *routing constraints*. A shift must start from the depot which are located the resources composing the vehicle and must end by returning to this one. Some triplets of resources are not admissible. Each resource can be used only during one of its availability time windows. The working and driving times of drivers are limited; as soon as a maximum duration is reached, the driver must take a layover with a minimum duration. In addition, the duration of a shift cannot exceed a maximal value depending on the driver. The sites visited along the tour must be accessible to the resources composing the vehicle. The date of pickup/delivery must be contained in one of the opening time windows of the visited site. Two graphical views of a shift are illustrated on Figure 2.2.

Inventory constraints can be modeled as a flow network. Two kinds of inventories have to be managed: tanks of sites (customers and plants) and trailers. In any case, the quantity in a storage must remain between zero and its capacity. For a customer c, the tank level $l(c, h)$ at each time step h is equal to the tank level at the previous time step $h - 1$, minus the forecasted consumption $F(c, h)$ over h, plus all the deliveries performed over h. The quantities delivered to customers must be positive (loading is forbidden at customers). Hence, the inventory dynamics at any time step h for customer c can be formally written as

$$\begin{cases} l(c,h) = l(c,h-1) - F(c,h) + \sum\limits_{o \in O(c,h)} q(o) \\ \text{if } l(c,h) < 0, \text{ then } l(c,h) = 0 \end{cases}$$

with $q(o)$ the quantity delivered during an operation o and $O(c,h)$ the set of operations performed at site c whose starting date belongs to time step h. Forbidding stockouts corresponds to force $l(c,h)$ to be greater than the safety level for each customer c and each time step h. Similar equations hold for plants whereas the inventory dynamics for trailers is much simpler since operations performed by a trailer cannot overlap. Hence the quantity in a trailer is not defined for each time step but after each of its operations. Starting at an initial level, this quantity is merely increased by loadings and decreased by deliveries.

The objective of the vendor over the long term is to minimize the total cost of shifts. The cost of a shift depends on the working duration and on the traveled distance, but also on the number of deliveries, loadings, and layovers appearing during the shift. This cost is usually divided by the total delivered quantity in order to increase the readability of this economic indicator. The cost per ton ratio (or miles per ton when costs are approximated by distances), namely the logistic ratio, is widely used in the industry and in the academic literature. Since reliable forecasts (for both plants and customers) are only available over a 15-day horizon, shifts are operationally planned day after day with a short-term rolling horizon of 15 days. It means that each day, a distribution plan is deterministically built for the next 15 days, but only shifts starting at the current day are fixed. Large-scale instances have to be tackled within short computing times. A geographic area can contain up to several hundred customers. All dates and durations are expressed in minutes and the inventory dynamics for plants and customers are computed with a time step of one hour. The execution time for computing a short-term planning is limited to 5 minutes on standard computers.

Following the methodology previously described, a new practical solution approach is presented for tackling this real-life inventory routing problem (IRP). This generalization of the vehicle routing problem was often handled in two stages in the past: inventory first, routing second. On the contrary, a characteristic of our local search approach is the absence of decomposition, made possible by a fast volume assignment algorithm. Moreover, thanks to a large variety of randomized neighborhoods, a standard first-improvement descent is used instead of tuned, complex metaheuristics. An extensive computational study shows that our solution provides long-term savings exceeding 20 % on average compared to solutions built by a classical urgency-based constructive algorithm or even by expert planners. Confirming the promised gains in operations, the resulting decision support system was deployed worldwide. For more details on this topic, the reader is referred to our paper [14].

State of the art

Since the seminal work of [11] on a real-life inventory routing problem, a vast literature has emerged on the subject. In particular, a long series of papers was published by [25, 26, 28, 102, 103], motivated by a real-life problematic encountered in the industry. However, in many companies, inventory routing is still done by hand or supported by basic software, with rules like: serve emergency customers (that is, customers whose inventory is near to run out) using as many maximal deliveries as possible (that is, deliveries with quantity equal to the trailer capacity or, if not possible, to the customer tank capacity). For more references, the interested reader is referred to the recent papers by [102, 103], which give a comprehensive survey of the research done on the IRP over the past 25 years.

To our knowledge, the sole papers describing practical solutions for such a broad inventory routing problem are the ones by Savelsbergh et al. [26, 30, 102, 103]. The approaches described in these papers are the same in essence: the short-term planning problem is decomposed to be solved in two phases. In the first phase, it is decided which customers are visited in the next few days, and a target amount of product to be delivered to these customers is set. In the second phase, vehicle routes are determined taking into account vehicle capacities, customer delivery windows, drivers restrictions, etc. The first phase is solved heuristically by integer programming techniques, whereas the second phase is solved with specific insertion heuristics [29].

Search strategy and moves

Following the methodology described above, when feasible solutions are not trivial to find or even are unlikely to exist, a good modeling practice is to relax some constraints by introducing appropriate penalties. In the present case, the no-stockout constraints are removed and a penalty is introduced, counting for each customer the number of time steps for which the quantity in tank is smaller than the safety level. This stage ensures that the software always returns a (possibly infeasible) solution. The initial solution is built using a classical urgency-based constructive algorithm. Then, the heuristic is divided into two optimization phases: the first one consists in minimizing the number of stockout time steps regardless of costs, and the second one consists in optimizing the logistic ratio while preserving the number of stockout to zero. For each phase, a simple first-improvement randomized descent is used. Accepting neutral moves helps to diversify the search and then to converge toward high-quality solutions. The move to apply is chosen randomly with equal probability over all moves in the pool (improvements being marginal, further tunings with non-uniform distribution have been abandoned to facilitate maintenance and evolutions).

Two kinds of moves are used: the ones work on operations, the others work on shifts. The moves on operations are described on Figure 2.3). The moves on shifts can be outlined in a few words: insertion, deletion, rolling, move, swap, fusion, separation. These moves, and more generally our multiple neighborhood approach, is fully in line with the trends observed in [111] (discovered after this work). As done for the car sequencing problem, these core moves are derived from the very generic (move an operation randomly picked from all) to the very specific (move an operation from a customer to a closest one). The latter ones allow to speed up the convergence, while the former ones ensure to diversify the search. One can observe that no very large-scale neighborhood is employed. Roughly speaking, the neighborhood explored has a size $O(n^2)$ with n the number of operations and shifts in the current solution, but the constant hidden by the O notation is large. Indeed, the number of moves (counting all the derivations) exceeds fifty.

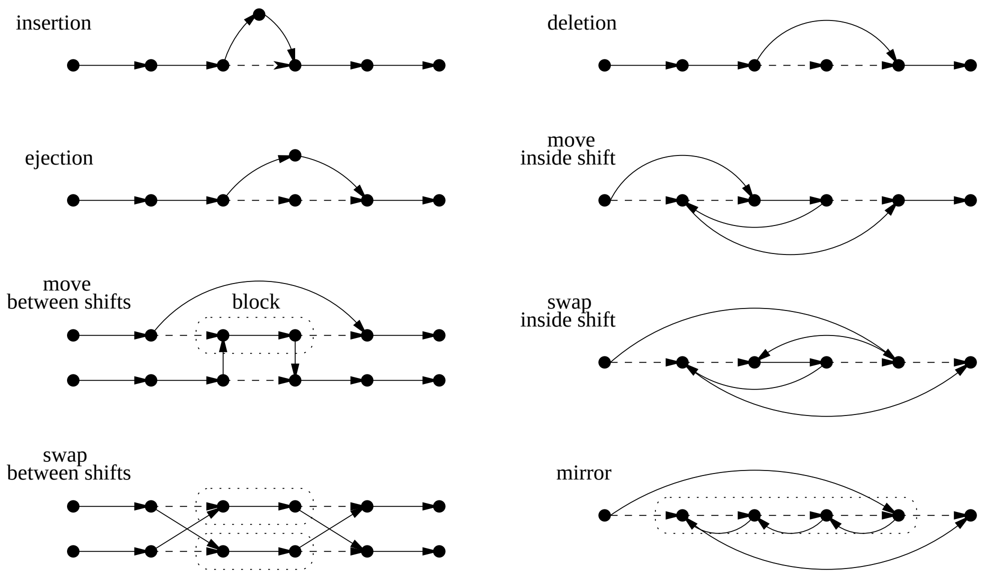

Figure 2.3: The moves on operations. Original tours are given by straight arcs, dashed arcs are removed by the move, curved and vertical arcs are added by the move.

Incremental evaluation machinery

Playing a central role in the efficiency of the local search heuristic, only the evaluation procedure is outlined here. For each move, this procedure follows the same process. First, the modified shifts are rescheduled in order to compute new dates for its operations, checking all routing constraints. Then, the delivered or loaded quantities are recomputed for all operations impacted by the move and their inventories are updated. Finally, the new number of stockouts and new logistic ratio are computed. Roughly speaking, the objective of the scheduling routine is to build shifts with smallest costs, whereas the volume assignment tends to maximize the quantity delivered to customers. Even approximately, it leads to minimize the logistic ratio.

When a shift is impacted by a move (for example, an operation is inserted into the shift), the starting and ending dates of its operations must be computed anew. We can reschedule dates forward or backward. Here computing dates can be done without assigning volumes to operations, because their durations do not depend on delivered/loaded quantities. Then, the problem is: having fixed its starting date and the order of its stops, schedule the shift with the earliest ending date. Similar problems have been recently studied in [5, 52, 53] but the algorithms proposed (exact or approximate) runs in $O(n^2)$ or even in $O(n^3)$ time, with n the number of operations to schedule (no more than ten per shift). For the sake of efficiency, a $O(n)$-time algorithm has been designed to solve heuristically this shift scheduling problem. This algorithm is greedy in the sense that operations are chronologically set without backtracking. We try to minimize the unproductive time over the shift. Thereby, the main idea behind the algorithm is to take rests as late as possible during the trip and to avoid waiting time due to opening time windows of locations as much as possible. Here we try to remove waiting time by converting it into rest time, but only on the current arc, which is suboptimal. On the other hand, we have observed that waiting time is rarely generated in practice since many trips are completed in a day or even half a day, ensuring the optimality of the algorithm in most cases.

Having rescheduled modified shifts, we have to reassign quantities to impacted operations. Having fixed the dates of all operations, the problem consists in assigning volumes such that inventory constraints are respected, while maximizing the total delivered quantity over all shifts. From the theoretical point of view, the present problem is not so hard once observed that it can be formulated as a maximum flow problem in a directed acyclic network. This one can be solved in $O(n^3)$ time by using a classical maximum flow algorithm [34, pp. 625–675], with n the number of operations. Once again, such a time complexity is not desirable here, even if guaranteeing an optimal volume assignment. Then, an $O(n \log n)$-time greedy algorithm has been designed to solve approximately the problem. The main idea behind the algorithm is simple: having ordered operations chronologically (that is, according to increasing starting dates), quantities are assigned to operations in this order following a greedy rule. Here we use the basic rule consisting in maximizing the quantity delivered/loaded at each operation. In graph-theoretical terms, the algorithm consists in pushing flow in the induced directed acyclic network following a topological order of the nodes, ensuring that no node is visited twice. Because the number n of operations is large (several hundreds), this algorithm remains too time-consuming, even if running in nearly linear time.

Hence, the greedy algorithm has been revisited to compute minimal reassignments. It consists in changing only the volumes on the impacted operations: the operations whose dates are modified by the move and the ones whose inventory has to be updated. This complicates notably its practical implementation. Indeed, changing the quantity delivered at an operation is delicate since increasing (resp. decreasing) the quantity may imply overflows (resp. stockouts) at future operations. Then, determining the quantity to deliver/load at each operation is not straightforward. Hence, an $O(\bar{n} \log \bar{n})$-time algorithm was designed for assigning volumes, with $\bar{n}$ the number of impacted operations. In theory, this greedy algorithm is far from being optimal. One can construct simple networks for which the greedy algorithm fails to find an optimal assignment. On the other hand, two sufficient conditions hold for which the greedy assignment is optimal: each customer is served at most once over the planning horizon; each shift visits only one customer. These conditions are interesting because likely to be met in practice. The running time of this critical routine is shown to be *100 times faster than the full application of the greedy algorithm* ($\bar{n} = n$) and *2000 times faster than exact algorithms*. On the other hand, the total volume delivered by the routine is close to the optimal assignment, in particular when no stockout appears: the average gap between the greedy assignment and an optimal one is lower than 2 %.

The whole local search heuristic was implemented in C# programming language. The resulting program includes nearly 30,000 lines of code, whose 20 % are dedicated to check the validity of all incremental data structures at each iteration (only active in debug mode). On average, our algorithm explores nearly 1 million feasible solutions within the 5 minutes of running time, even for large-scale instances. The average gain exceeds 20 % over urgency-based greedy solutions or even by expert planners. An extensive computational study can be found in our paper [14].

Chapter 3

Local search for 0-1 nonlinear programming

Mixed-integer linear programming (MILP or simply MIP) is undoubtedly the most powerful general-purpose approach to combinatorial optimization. Based on a simple and rather generic mathematical formalism, MIP offers powerful software tools to OR practitioners to solve real-world combinatorial problems. The success of MIP [20] is so big that MIP solvers are now an indispensable tool for OR engineers and researchers. We think that this success is due to two factors: first the simplicity of the MIP formalism (the language serving as interface with the user), then the ease of use of MIP "model-and-run" solvers (the user models his problem, the solver resolves it). One can observe that constraint programming (CP) now follows the road toward pure model-and-run solvers [89].

Despite the remarkable progresses made over the past 15 years [20], MIP solvers always fail to solve many of the highly combinatorial problems encountered in the practice of OR. Indeed, MIP solvers still fail to find even feasible solutions to problems inducing only thousands of binary decisions. The Renault's car sequencing problem [41] studied in the previous chapter is a good example. The linear relaxation is not good enough to effectively prune the exponential branch-and-bound tree. Then, the search provides no feasible solution after hours of computation. When MIP solvers are ineffective, practitioners generally implement hand-made local search algorithms to obtain good-quality feasible solutions quickly. However, dedicated local search approaches, even if industrialized as explained in the previous chapter, induce extra costs in development and maintenance.

3.1 The LocalSolver project

Tree search approaches like branch-and-bound are in essence designed to *prove optimality*, which is different from what users first expect: *good feasible solutions quickly.* Moreover, tree search has an exponential behavior which makes it *not scalable* faced with real-world combinatorial problems inducing millions of binary decisions. The two problems addressed in the previous chapter are some good examples of combinatorial or mixed-variable optimization problems intractable with current model-and-run solvers, in particular MIP solvers. The most striking fact about this reality can be observed on the famous and intensively studied Traveling Saleman Problem (TSP). The largest TSP instances for which the optimal solution is known counts 85,900 cities [4]. The solution is first computed by local search and then proved to be optimal by branch-and-cut. While obtaining the solution takes hours using LKH – the high-end implementation of Lin-Kernighan local search by Helsgaun [61, 62], it takes 136 CPU years to prove its optimality using Concorde – the branch-and-cut code by Applegate et al. [4]. Moreover, LKH is able to provide near-optimal solutions in minutes to TSP instances with 10 millions of cities; it holds the record for the World TSP with 1,904,711 cities.

Indeed, tree search suffers from several drawbacks in practice. When tackling highly-combinatorial problems, linear relaxation in MIP or filtering in CP are often useless but costs a lot in computational efficiency. Then, why losing time to enumerate partial solutions? Another issue concerns the erraticism of tree search [47]: tree search is not really suited to explore randomly (without bias) a solution space. So why an incomplete tree search (which is mostly the case in practice) would be better than local search? An interesting fact is that MIP and CP solvers integrate more and more local search ingredients in their branching heuristics [74]: large neighborhood search [107, 46], local branching [44, 45], or relaxed induced neighborhood search [36]. Consequently, we are convinced that local search is the technique of choice to scale up, since each step (that is, each move) can generally be performed in sublinear or even constant time. If carefully designed and implemented, the resulting algorithms converge *empirically* toward high-quality solutions in weakly polynomial time. For example, the convergence of LKH [62] toward near-optimal TSP solutions is shown to grow in linear time. Similar behaviors can be exhibited for the numerous local search heuristics implemented by our team [14, 16, 40, 41, 42, 49, 67] to solve large-scale combinatorial optimization problems arising in business and industry.

Started in 2007, the goal of the *LocalSolver* project [15] was to offer the power of pure and direct local search to practitioners through a model-and-run solver. The objectives of this applied research project were to design a simple and general mathematical formalism suited for local search (model) and to develop an effective solver based on pure and direct local search (run). The design of the local search solver was guided by a fundamental principle according to us: the solver must at least do what an expert would do facing the problem (as exposed in Section 2.1). Since 2012, LocalSolver[1] is commercialized by Innovation 24, subsidiary of Bouygues Group, in partnership with Aix-Marseille Université and the French National Center for Scientific Research (CNRS–LIF Marseille), but its use remains free for the academic community. Having reviewed the related works in the literature, the modeling formalism associated with the current version of LocalSolver (3.1) is presented. Then, the main ideas on which LocalSolver relies are outlined. Finally, some computational results are provided demonstrating the effectiveness of LocalSolver compared to state-of-the-art model-and-run solvers.

[1] `http://www.localsolver.com`

3.2 State of the art

Leveraging the power of local search into software tools easy to use by OR practitioners is a quest which has started in the beginning of the nineties. Most proposals made to offer tools or reusable components for local search programmers take the form of a framework handling the top layer of the algorithm, namely metaheuristics (see for example [27, 37]). In this case, moves and associated incremental algorithms are implemented by the user, while the framework is responsible for applying the selected parameterized metaheuristic. However, designing moves and implementing incremental evaluation algorithms represent the largest part of the work (and of the resulting source code). As mentioned in the previous chapter, these two layers consume about 30 % and 60 % of the development times respectively. Moreover, the algorithmic layer dedicated to the evaluation of moves is particularly difficult to engineer, because it requires both an expertise in algorithms and a dexterity in computer programming. Hence, these frameworks do not address the hardest issues of the engineering of local search algorithms. Two frameworks aim at answering to these needs: Comet Constraint-Based Local Search (CBLS) [114] (and its ancestor Localizer [78]) and iOpt [115]. These tools allow an automatic evaluation of moves, but the implementation of these moves remains the responsibility of the user. Note that a generic swap-based tabu search procedure [38, pp. 330–331] is available in Comet CBLS 2.1, which can be used for tackling directly integer models.

State-of-the-art MIP or CP solvers include more and more heuristic features, some of which related to local search. Berthold [17] provides an extensive survey of primal heuristics for MIP. Among them, we can cite large neighborhood search [46, 107] used in constraint programming, as well as local branching [44, 45] or relaxed induced neighborhood search [36] in mixed-integer programming. Nevertheless, all these works consist in integrating high-level local search ingredients in the tree search paradigm, whereas the actual power of local search relies on fast incremental computations made possible by small modifications of the incumbent solution.

Some attempts have been done to use pure and direct local search approaches in discrete mathematical programming. The most famous ones have been introduced by researchers from the artificial intelligence community. Indeed, some of the best provers for Satisfiability Testing (SAT) or Pseudo-Boolean Programming rely on stochastic local search (see for example Walksat [106] and WSAT(OIP) [116]). Some attempts have been done in binary programming [32, 87] or in integer programming [2, 97]. In all cases, the main difficulty encountered by the authors is that modifying randomly some binary variables lead frequently to infeasible solutions and that recovering feasibility is difficult and long, causing a very slow convergence toward good-quality solutions or even feasible solutions. This issue is highlighted by the fact that local search works very well in solving unconstrained binary programming, in particular unconstrained binary quadratic programming [76].

Ultimately, to the best of our knowledge, no effective and widely used model-and-run solver exploiting the power of pure and direct local search is currently available for tackling large-scale real-life combinatorial optimization problems. Developing such a solver was initially the goal of the LocalSolver project [15].

3.3 Enriching modeling standards

Modeling is a crucial part of the job in mathematical optimization. The user of a math programming software has to model his problem in a suitable way to hope an effective automatic resolution by the software. In this way, the mathematical formalism which is offered to the user is decisive. From the beginning of the use of mathematical optimization in business and industry, there is a need of simple and clear algebraic modeling languages to ease the communication with optimization solvers. In a recent historical paper, Fourer [43] describes the evolution from matrix generators to modeling languages. Nevertheless, a bottleneck remains. Modeling languages allow a high-level description of mathematical optimization problems, but this one is partially lost when passed to the solver. Indeed, math programming solvers still only accept matrix representations as standard input. One can observe that such a low-level representation has several drawbacks for the resolution process itself.

For the sake of clarity, practitioners profusely use intermediate variables in their models. Now in matrix representations, decision variables are not distinguished from intermediate variables, the ones whose value can be deduced from the values of decision variables. Then, all the assignments induced by the declaration of these intermediate variables are set as constraints when passed to the solver. Many primal heuristics in mixed-integer programming suffer from this loss of information [17]. The prior attempts to tackle pseudo-boolean or integer programs [2, 32, 87, 97] by local search were facing the same problem. Note that modern MIP solvers extensively use heuristic preprocessing procedures to recover a part of this lost information [74]. Another drawback of matrix representation arises in nonlinear optimization. Specifying a nonlinear model as a matrix is not convenient for users. Expressions built with classical mathematical functions like the division or the power are difficult to set in such a format. Making the interface with nonlinear solvers complicated, this issue appears as an obstacle to their use and thus to their development. According to us, more expressiveness is better for users, and better for solvers. Inspired by the state-of-the-art modeling systems [43], we propose a new data representation for mathematical optimization solvers, as well as a new mathematical modeling/scripting language to support this one. This data representation, called LocalSolver Model (LSM), and the associated scripting language, called LocalSolver Programming language (LSP), are implemented in LocalSolver [15].

LocalSolver modeling formalism

The modeling formalism supported by LocalSolver is close to the classical 0-1 integer programming formalism. However, it is enriched with common mathematical operators, which makes it easy to use by OR practitioners. The LSM representation of a mathematical optimization instance corresponds to its algebraic expression tree. More precisely, the subjacent data structure is a directed acyclic graph (DAG). Its root nodes are the decision variables of the model. Each inner node correspond to an intermediate variable, whose value is computed by applying a predefined mathematical operator on its parent variables in the DAG. These inner nodes are related to so-called one-way constraints or invariants in CP-oriented frameworks like iOpt [115] or Comet [114] (or its ancestor Localizer [78]).

Then, each variable (also called expression) can be tagged as constrained, minimized, or maximized. Only boolean expressions can actually be constrained to be true. In this way, a solution is a complete instantiation of the decisions such that all constraints in the model are satisfied.

```
x1 <- bool();
x2 <- bool();
x3 <- bool();
y1 <- bool();
y2 <- bool();
y3 <- bool();
sx <- sum(x1, x2, x3);
sy <- sum(y1, y2, y3);
constraint leq(sx, 2);
constraint geq(sy, 2);
obj <- max(sx, sy);
minimize obj;
```

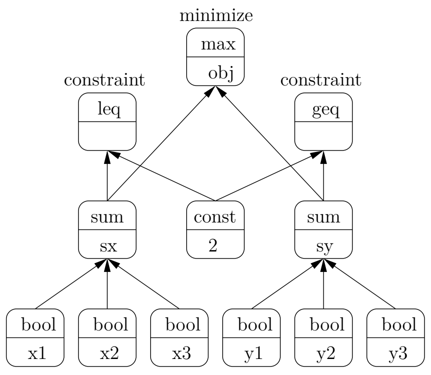

Figure 3.1: The directed acyclic graph (on the right) induced by a toy LSM model (on the left). For each node, the type (resp. name) of the node is given above (resp. below).

Following the LSM format, a mathematical optimization model simply consists in declaring the decision variables, in building some expressions based on these decisions, and then in tagging some expressions as constraints or objectives. Note that the user shares no information about the way which the model is stored or even solved internally. For example, a toy problem is described on Figure 3.1. The statement `bool()` creates a boolean decision variable, that is, a decision variable whose value is false or true. Boolean variables are treated as integers, with the convention false=0 and true=1. Then, the keyword `<-` is used to define intermediate variables `sx` and `sy`, which can be boolean or integer. The keyword `constraint` prefixes each constraint definition. In the same way, the keyword `minimize` prefixes the objective of the program. In the current version of LocalSolver (3.1), only boolean decisions are managed but extending the grammar to integer or continuous decision variables is straightforward. Expressions can be built upon decisions or other expressions (without recursion) by using given arithmetic, logical, relational or conditional operators like `sum`, `prod`, `min`, `max`, `div`, `mod`, `sub`, `abs`, `log`, `exp`, `pow`, `cos`, `sin`, `tan`, `and`, `or`, `xor`, `not`, `eq`, `neq`, `geq`, `leq`, `gt`, `lt`, `iif`, and others. In summary, the LSM format is based on a functional syntax, with no limitation on the nesting of expressions. Some mathematical operators only apply to a certain number of arguments or to a certain type of expressions. For instance, the `not` operator takes only one argument whose type must be boolean, whereas `sum` or `and` can take an arbitrary number of arguments. The conditional operator `iif` takes exactly three arguments, the first one being necessarily boolean.

Any boolean expression can be constrained to be true by prefixing the expression by `constraint`. An instantiation of decision variables is valid if and only if all constraints take value 1, coding for satisfied. As explained in the previous chapter, hard constraints have to be carefully considered when modeling real-world problems. Many of the constraints expressed by users are unlikely to be satisfied in real life: these ones have to be considered as primary objectives. The LSM formalism offers a feature making it easy to do. The objectives can be defined using the modifier `minimize` or `maximize`. Any expression can be used as objective. If several objectives are defined, they are interpreted as a lexicographic objective function. The lexicographic ordering is induced by the order in which the objectives are declared. For instance, in car sequencing with paint colors, when the goal is to minimize violations on ratio constraints and then the number of paint color changes, the objective function can be directly specified as: `minimize ratio_violations; minimize color_changes;`. This feature allows avoiding the classical modeling workaround where a big coefficient is used to simulate the lexicographic order: `minimize 1000 ratio_violations + color_changes;`. Note that the number of objectives is not limited and can have different directions (minimization or maximization).

For the sake of simplicity and readability, LSM was presented as a file format (that is, as a grammar). But LSM must be viewed as an interface between the user and the solver. Then, it can be implemented as application programming interface for any computing framework (for example, C or XML). Such interfaces are currently provided as libraries for C++, Java and .NET languages in the LocalSolver package.

LocalSolver programming language

The mathematical expressiveness offered by the LSM formalism is richer than the classical matrix representation required in input of math programming solvers. Even if LSM can be viewed as a language, it is not convenient to be used as a modeling language. Indeed, LSM offers no programming feature. Then, we have embedded LSM into an innovative programming language for mathematical optimization, namely LSP. The current version of the LSP language is dedicated to rapid prototyping: it embeds modeling features into a lightweight scripting language.

In summary, the LSP language is interpreted, strongly typed but dynamic, with an implicit declaration of variables. It offers many built-in variables and functions. The mathematical built-in functions can be used for both modeling or programming. For instance, `c <- a * b` means that an expression `c` is declared corresponding to the product of modeling variables `a` and `b`. On the other hand, `c = a * b` means that the product of programming variables `a` and `b` is assigned to `c`. Usual mathematical symbols like `+` or `<=` can be used as shortcuts in an infix way. An innovative compact looping syntax `[i in 0..n]` can be used to iterate over a set of variables or expressions. A short LSP program is described below to illustrate the main features of the language. More examples can be found in the LocalSolver example tour, directly available from the web[2].

```
/* multiobjective_knapsack.lsp */

function input() {
  nbItems = 8; sackBound = 102;
  weights = {2, 20, 20, 30, 40, 30, 60, 10};
  values = {15, 100, 90, 60, 40, 15, 10, 1};
```

[2]http://www.localsolver.com/exampletour.html

```
function model() {
  // 0-1 decisions
  x[0..nbItems-1] <- bool();

  // weight constraint
  sackWeight <- sum[i in 0..nbItems-1](weights[i] * x[i]);
  constraint sackWeight <= sackBound;

  // maximize value
  sackValue <- sum[i in 0..nbItems-1](values[i] * x[i]);
  maximize sackValue;

  // secondary objective: minimize the product of minimum and maximum values
  sackMinValue <- min[i in 0..nbItems-1](x[i] ? values[i] : 1000);
  sackMaxValue <- max[i in 0..nbItems-1](x[i] ? values[i] : 0);
  sackProduct <- sackMinValue * sackMaxValue;
  minimize sackProduct;
}

function param() {
  lsTimeLimit = 60; lsNbThreads = 4;
}
```

3.4 The core algorithmic ideas

In this section, we outline the two ingredients which are critical to the effectiveness of our approach. These ones follow naturally the methodology exposed previously: multiple randomized small-neighborhood moves as well as a highly-optimized incremental evaluation machinery. Our main innovation lies in the design of local search moves working on abstract combinatorial structures, far beyond the basic "flips" used in SAT solvers.

Effective local search moves

As suggested in introducing the LocalSolver project, we wish that our solver autonomously performs the moves that an experimented practitioner would have designed to solve the problem, and ultimately some moves he would certainly not implement: complex small-neighborhood moves (like 3,4,5-opt moves for the traveling salesman problem) or even compound moves. The main characteristic of such moves is to maintain the feasibility of the incumbent solution, or at least the feasibility of the constraints inducing the essential of its combinatorial structure. Indeed, the main conclusions of the few attempts to tackle 0-1 programs by local search [2, 32, 87, 97] are related to this issue: simple flips or even k-flips are ineffective because recovering feasibility takes too much time. Maintaining feasibility means performing moves ensuring that the resulting solution remains in the solution space. Thus, performing moves allowing to search on hypersurfaces (for instance, on the classical hyperplane induced by the linear equality constraint $\sum_i A_i x_i = B$) is crucial for effectiveness and efficiency. Of course, more constrained is the problem, smaller is the solution space, and more difficult will be the search. As explained in the previous chapter, hardly-constrained problems encountered in the real world mostly result from a bad understanding of the user needs or from bad modeling practices, yielding the risk of "no solution found" answers to the end-users. Our idea to maintain feasibility while performing a move on the 0-1 solution vector is inspired from ejection chain techniques [98] or destroy and repair techniques [93]: *having flipped a boolean decision, repair violated constraints by flipping other boolean decisions.* Finding the latter boolean decisions can be done by exploiting the hypergraph induced by decisions (as vertices) and constraints (as edges). Figure 3.2 illustrates such a move, inducing a cycle along six boolean variables while satisfying the constraints involving these six decisions.

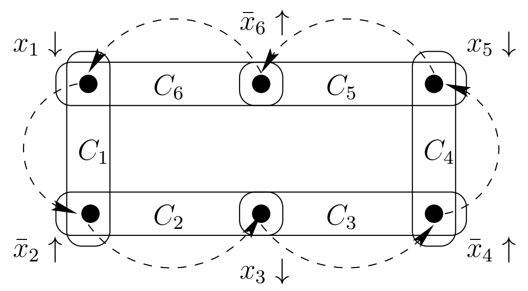

Figure 3.2: A cyclic move involving six boolean variables x_1, x_3, x_5 (whose current value is 1) and $\bar{x}_2$, $\bar{x}_4$, $\bar{x}_6$ (whose current value is 0), and six constrained sums $C_1, \ldots, C_6$. Each variable belongs to two sums (for example, x_1 belongs to C_1 and C_6). Now, x_1, x_3, x_5 are decreased ($\downarrow$) while $\bar{x}_2$, $\bar{x}_4$, $\bar{x}_6$ are increased ($\uparrow$). This move preserves the values of the sums, and thus the feasibility of the constraints.

To understand the effect of such moves, let us consider the car sequencing problem [40, 41] introduced in the previous chapter. This problem can be modeled as an assignment problem by defining for each car i and position p a boolean decision $x_{i,p}$. A basic move for this model consists in exchanging the positions of two vehicles. Exchanging the positions p and q of two cars i and j corresponds to flipping the 4 decisions $x_{i,p}$, $x_{i,q}$, $x_{j,q}$, $x_{j,p}$, which preserves the feasibility of the 4 partition constraints where these variables appear. In fact, paths (or cycles) of alternating flips along the constraints correspond to classical k-move or k-exchange neighborhoods for assignment, partitioning, packing, or covering problems.

Many variants of the chain or cycle described above can be derived. These variants can be specific to some combinatorial structures (that is, to some constraint hypergraphs) – with higher probabilities of success, to speed up the convergence of the search – or more generic – with lower probabilities of success, but favoring the diversification of the search. In the same way, they can induce small neighborhoods or larger ones, leading to more or less efficient incremental evaluations. For example, flipping more than one decision when repairing a constraint induces more complex walks along the constraint network, whose trace is then related to subgraphs in the constraint hypergraph. In packing problems, such moves allow ejecting two objects of size 1 while adding an object of size 2 in the same set. At each iteration of the search, the move to apply is selected randomly following a distribution dynamically adapted along the search. Moves often leading to infeasible solutions tend to be abandoned. On the other hand, moves with larger acceptance or improvement rates are more frequently attempted.

Incremental evaluation machinery

The practical power of local search relies on the efficiency of the moves. When impacting only locally the solution vector (that is, modifying only a few of its coordinates), these moves can be quickly evaluated through incremental computations. Incremental computation belongs to the folklore of computer science and algorithm engineering. To our knowledge, the first mathematical optimization frameworks involving an automatic incremental evaluation are Localizer [78] (the ancestor of Comet [114]) and iOpt [115]. Based on a modeling formalism similar to LocalSolver modeling formalism, the incremental evaluation relies on invariants induced by mathematical operators. Although we claim no novelty for this mechanism, we emphasize the main specificities of our implementation in LocalSolver.

The fast evaluation of moves is obtained by exploiting the invariants induced by each type of nodes (that is, operators) during the propagation [78]. A breadth-first search propagation of the modifications is performed along the DAG, guarantying that each node is evaluated at most once. Following a classical observer pattern, the propagation is reduced to impacted nodes: a node is said to be impacted if some of its parents have been modified. For example, consider the node $z \leftarrow a < b$ with a current value equals to true. This one will not be impacted if a is decreased or b increased. Then, each node of the DAG implements the following methods: `init`, `eval`, `commit`, `rollback`. The method `init` is responsible of the initialization of the value of the node according to (the values of) its parents, before starting the search. The specific data structures attached to the node, used for speeding up its incremental evaluation, are also initialized by this method. Having applied a move on decision variables, the `eval` method is called to reevaluate incrementally the value of a node, when this one is impacted during the DAG propagation. Then, if the move is accepted (according to the search strategy), the `commit` method is called on each modified node for validating the changes implied by the move. Otherwise, the move is rejected, and the `rollback` method is used instead.

The `eval` method takes in input the list of modified parents, that is, the parent nodes whose current value has changed. For a linear operator like `sum`, the evaluation is easy: if k terms of the sum are modified, then its new value is computed in $O(k)$ time. But for nonlinear operators, significant accelerations can be obtained in practice. For example, consider the node $z \leftarrow \text{or}(a_1, \ldots, a_k)$ with M the list of modified a_i's and T the list of a_i's whose current value is true. Thus, one can observe that if $|M| \neq |T|$, then the new value of z is necessarily true, leading to a constant-time evaluation. Indeed, if $|M| < |T|$, then at least one parent remains with value equals to true; otherwise, there exists at least one parent whose value is modified from false to true. Our implementation is focused on the *empirical* time complexity, and not only on the worst-case time complexity. Constant factors do matter: fine algorithmic and code optimizations improve speed of evaluation by several orders of magnitude. At our acquaintance, the property mentioned above to maintain the `or` operator is not employed in Comet or iOpt systems. In the same way, in Localizer [78, p. 67], the `min` operator is classically maintained in $O(\log k)$ time with k the number of operands using a binary heap. In LocalSolver, we distinguish two cases. If the minimum value among the modified operands is lower than or equal to the current value of the `min` operator, or if one support remains unmodified, then the evaluation is optimally done in $O(|M|)$ time with $|M|$ the number of modified values. Otherwise, the evaluation is performed in $O(k)$ time. In practice, the former case is by far the most frequent and the number of modified operands is small (that is, $|M| = O(1)$), ensuring an amortized constant-time evaluation.

3.5 Benchmarks

LocalSolver was tested on a benchmark mixing academic and industrial problems. We insist on the fact that our purpose is not to achieve state-of-the-art results for all the tested problems. The main goal of LocalSolver is to obtain, used as a black box, good-quality solutions quickly, in particular when tree search solvers fail to find any solution. Then, LocalSolver is compared to existing model-and-run solvers, here the state-of-the-art MIP solver Gurobi[3], on a standard computer equipped with the operating system Windows 7 x64 and a chip Intel Core i7-820QM (4 cores, 1.73 GHz, 6 GB RAM, 8 MB cache). As expected, the main conclusion is that LocalSolver outperforms tree search solvers by several orders of magnitude when large-scale combinatorial optimization problems are addressed. Older results can be found in our paper [15] where our early prototype LocalSolver 1.1 is compared to IBM ILOG CPLEX 12.2[4].

Car sequencing

Here we outline the results obtained on the car sequencing problem, addressed in the previous chapter. LocalSolver and Gurobi are launched with their default parameter settings. Note that the results obtained by model-and-run CP or SAT solvers are not mentioned because not competitive (see for example results presented in [91, 92]). For each solver, we use a standard model adapted to the formalism of the solver. For both solvers, the assignment of cars to positions is modeled with boolean variables and the violations on each ratio constraint are summed (see [40] for details). Sample results are presented for 5 instances on Table 3.1 below: 10-93 (100 cars, 5 options, 25 classes), 200-01 (200 cars, 5 options, 25 classes), 300-01 (300 cars, 5 options, 25 classes), 400-01 (400 cars, 5 options, 25 classes), 500-08 (500 cars, 8 options, 20 classes). The first 4 instances are available in CSPLib[5]; the fifth comes from a benchmark generated by [92]. The line "state-of-the-art" corresponds to the state-of-the-art results obtained by our local search algorithm, described in [40]. The results presented in the table have been obtained with a time limit fixed to 10, 60 and 600 seconds respectively; the corresponding LSP file is given in the LocalSolver Example Tour[6]. The cost of the best solution found is given (the symbol X is used if no solution has been obtained within the time limit, the symbol * is added to indicate that the solution found was *proved* optimal). In summary, one can observe that LocalSolver outperforms Gurobi as the scale of instances grows.

[3] `http://www.gurobi.com`

[4] `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer`

[5] `http://www.csplib.org`

[6] `http://www.localsolver.com/exampletour.html?file=car_sequencing.zip`

Table 3.1: Sample results for the academic car sequencing problem.

Time limit: 10 s	10-93	200-01	300-01	400-01	500-08
Gurobi 5.5	140	274	X	429	513
LocalSolver 3.1	**6**	8	9	11	24
LocalSolver 4.0 beta	8	**5**	**8**	**10**	**19**
Time limit: 60 s	10-93	200-01	300-01	400-01	500-08
Gurobi 5.5	**3**	66	1	356	513
LocalSolver 3.1	6	**3**	**3**	7	10
LocalSolver 4.0 beta	6	4	**3**	**5**	**6**
Time limit: 600 s	10-93	200-01	300-01	400-01	500-08
Gurobi 5.5	**3**	2	**0***	**1**	20
LocalSolver 3.1	6	2	1	2	4
LocalSolver 4.0 beta	4	**0***	**0***	2	**0***
State of the art	3	0*	0*	1	0*
Binary decisions	500	1000	1500	2000	4000

Table 3.2: Sample results for Renault's car sequencing problem.

Time limit: 600 s	I1	I2	I3	I4
Gurobi 5.5	3,027,839	X	X	X
LocalSolver 3.1	3,120	**217,058**	11,423,006	166,632
LocalSolver 4.0 beta	**3,114**	240,075	**382,010**	**160,572**
State of the art	3,109	192,066	337,006	160,407
Binary decisions	44,184	259,560	374,596	278,880

More interesting are the results obtained on the real-world version integrating the constraints and objectives of the paint workshop proposed by Renault as subject of the ROADEF 2005 Challenge[7] which was discussed in Section 2.2. Table 3.2 contains sample results for four instances: I1 = 022-EP-ENP-RAF-S22-J1 (540 cars, 9 options, 14 colors), I2 = 023-EP-RAF-ENP-S49-J2 (1260 cars, 12 options, 13 colors), I3 = 024-EP-RAF-ENP-S49-J2 (1319 cars, 18 options, 15 colors), I4 = 025-EP-ENP-RAF-S49-J1 (996 cars, 20 options, 20 colors). The state of the art corresponds to the local search heuristic described in Section 2.2, which won the challenge [41, 109]. Note that the engineering of this dedicated algorithm required nearly 80 working days to the authors. In contrast, Gurobi finds no feasible solution after several hours of computation for instances I2, I3, I4. After one hour of running time, Gurobi obtains a solution whose cost is 194,161 for I1, which is far away of LocalSolver's result within 5 minutes only. For instance I3, the resulting LocalSolver model contains 516,936 expressions whose 374,596 are binary decisions (less than 50 MB of RAM are required for the execution). Localsolver explores nearly 1 million *feasible* solutions per minute. According to its results, LocalSolver would have been ranked among the top competitors of the ROADEF 2005 Challenge. In particular, LocalSolver outperforms the dedicated variable neighborhood search by [94] mixing classical moves and

[7]`http://challenge.roadef.org/2005/en`

large neighborhood search by MIP.

Machine scheduling

Another striking result was recently obtained in the context of the ROADEF/EURO 2012 Challenge[8]. This real-world problem, posed by Google, consists in reassigning processes to machines while respecting different kinds of constraints (resources, mutual exclusions, etc.). The aim of this challenge is to improve the usage of a set of machines. A machine has several resources, like CPU and RAM for example, and runs processes which consume these resources. Initially each process is assigned to a machine. In order to improve the machine usage, processes can be moved from one machine to another. Possible moves are limited by hard constraints, as for example resource capacity constraints, and have a cost. A solution to this problem is a new process-machine assignment which satisfies all hard constraints and minimizes a given objective cost.

Using a 100-line mathematical model[9], LocalSolver was able to qualify for the final round of the Challenge, with rank 25th over 82 participating teams. LocalSolver is able to tackle the 10 instances from set A (until 100,000 decisions) within 5 minutes on a standard computer with 4 GB RAM. For all instances except the toy A1-1 (400 binary decisions), no model-and-run solver is able to provide a feasible solution within the time limit, to our knowledge. LocalSolver is able to solve some of the ultra-large instances from set B (from 500,000 to 250 million 0-1 decisions). It depends of the amount of RAM available on the computer. With 4 GB of RAM, we can tackle B1, B2. With 40 GB of RAM, you can tackle all instances except B7, B9, B10. In the latter case, we can notably attack B4 instance with 1 million decisions, 114 million expressions, 1.6 million constraints. The results obtained within 5 minutes on our standard computer with 6 GB RAM are detailed on Table 3.3.

Table 3.3: Results on Google's machine reassignment with 6 GB RAM (300 seconds).

Instances	Expressions	Decisions	Constraints	LocalSolver 3.1	LocalSolver 4.0 beta	State of the art
A1-1	6,020	400	503	44,306,501	44,306,501	44,306,501
A1-2	1,812,044	100,000	100,595	797,041,378	791,337,884	777,532,896
A1-3	1,423,438	100,000	26,097	583,007,622	583,006,420	583,005,717
A1-4	753,404	50,000	9,913	295,785,797	277,578,924	252,728,589
A1-5	229,213	12,000	13,905	727,578,409	727,578,309	727,578,309
A2-1	1,415,324	100,000	102,300	8,176,319	4,513,099	198
A2-2	3,769,381	100,000	19,770	1,328,470,442	1,216,918,411	816,523,983
A2-3	3,843,977	100,000	20,213	1,738,822,512	1,566,648,617	1,306,868,761
A2-4	1,537,771	50,000	13,373	2,309,678,761	2,054,256,551	1,681,353,943
A2-5	1,556,017	50,000	13,260	601,850,679	551,397,584	336,170,182
B1	10,165,011	500,000	284,155	4,436,929,040	4,352,763,543	3,339,186,879
B2	8,117,249	500,000	273,799	1,475,082,345	1,412,787,354	1,015,553,800

[8]`http://challenge.roadef.org/2012/en`

[9]`http://www.localsolver.com/exampletour.html?file=google_machine_reassignment.zip`

Quadratic assignment problem

The quadratic assignment problem (QAP) [22] is a classics of the operations research literature. The problem involves n facilities and n locations. For each pair of locations, a distance is specified and for each pair of facilities a weight or flow is specified (for example, the amount of supplies transported between the two facilities). The problem consists in assigning all facilities to different locations with the goal of minimizing the sum of the distances multiplied by the corresponding flows. Intuitively, the objective encourages factories with high flows between each other to be placed close together. The problem statement is similar to the assignment problem, except that the cost function is composed of quadratic terms instead of linear ones.

On Table 3.4, we present the results obtained by LocalSolver 3.1 using a basic model[10], within 5 minutes of running time. The state of the art corresponds to the best result known today, as referenced on the webpage[11] maintained by P. Hahn. Since mixed-integer (linear or quadratic) programming solvers are not competitive on QAP, we have omitted their results (no feasible solution is found within the time limit on the majority of instances).

Table 3.4: Results on the QAPLIB (300 seconds).

Instances	n	Decisions	Expressions	LocalSolver 3.1	State of the art	Gap
esc32a	32	1,024	124,299	136	130	5 %
esc32b	32	1,024	180,872	188	168	12 %
esc32c	32	1,024	219,153	642	642	0 %
esc32d	32	1,024	150,922	200	200	0 %
esc32e	32	1,024	11,148	2	2	0 %
esc32g	32	1,024	16,138	6	6	0 %
esc32h	32	1,024	235,791	444	438	1 %
kra30a	30	900	288,166	92,130	88,900	4 %
kra30b	30	900	288,166	93,020	91,420	2 %
kra32	32	1,024	328,558	94,440	88,700	6 %
lipa30a	30	900	732,736	13,399	13,178	2 %
lipa30b	30	900	731,258	177,255	151,426	17 %
lipa40a	40	1,600	2,374,581	31,998	31,538	1 %
lipa40b	40	1,600	2,368,797	565,314	476,581	19 %
lipa50a	50	2,500	5,885,226	62,859	62,093	1 %
lipa50b	50	2,500	5,876,130	1,431,681	1,210,244	18 %
nug30	30	900	510,877	6,238	6,124	2 %
sko42	42	1,764	2,078,709	16,282	15,812	3 %
sko49	49	2,401	3,817,588	24,146	23,386	3 %
ste36a	36	1,296	435,141	10,488	9,526	10 %
ste36b	36	1,296	435,628	18,066	15,852	14 %
ste36c	36	1,296	435,822	8,612,782	8,239,110	5 %
tai30a	30	900	745,196	1,914,966	1,818,146	5 %
tai30b	30	900	484,316	638,762,469	637,117,113	0 %
tai35a	35	1,225	1,384,849	2,515,194	2,422,002	4 %
tai35b	35	1,225	737,819	287,227,702	283,315,445	1 %
tai40a	40	1,600	2,382,379	3,284,344	3,139,370	5 %
tai40b	40	1,600	1,341,640	689,131,580	637,250,948	8 %
tai50a	50	2,500	5,905,491	5,230,902	4,938,796	6 %
tai50b	50	2,500	2,911,505	481,156,669	458,821,517	5 %
tho30	30	900	379,599	153,286	149,936	2 %
tho40	40	1,600	976,311	249,326	240,516	4 %
wil50	50	2,500	5,387,864	49,526	48,816	1 %
average						5 %

[10] `http://www.localsolver.com/exampletour.html?file=qap.zip`

[11] `http://www.seas.upenn.edu/qaplib`

MIPLIB 2010

We recently developed a tool to translate a MIP model (LP or MPS format) into a LocalSolver model (LSM format). Since the LocalSolver formalism is richer than the MIP formalism, this transformation is heuristic, trying to distinguish decision variables from intermediate variables and to recover some nonlinear expressions from matrix-oriented linear constraints. This tool allows to translate and solve successfully a number of the hardest and largest MIP instances of MIPLIB 2010[12], the well-known benchmark for MIP solvers. The results are given on Table 3.5 below. This experiment also shows that LocalSolver is able to provide quality solutions to large-scale problems, even if these ones are not suitably formulated for LocalSolver.

Table 3.5: Results on some of the hardest instances of the MIPLIB 2010 (300 seconds).

Instances	Status	Variables	LocalSolver 3.1	Gurobi 5.5	Optimum
opm2-z10-s2	hard	6,250	-25,719	-19,601	-33,826
opm2-z11-s8	hard	8,019	-33,028	-21,661	-43,485
opm2-z12-s14	hard	10,800	-46,957	-11,994	-64,291
opm2-z12-s7	hard	10,800	-46,034	-12,375	-65,514
pb6	hard	462	-62	-62	-63
queens-30	hard	900	-38	-36	-40
dc1l	open	37,297	11,100,000	21,300,000	unknown
ds-big	open	6,020	9,844	62,520	unknown
ex1010-pi	open	25,200	249	251	unknown
ivu06-big	open	1,812,044	479	9,416	unknown
ivu52	open	1,423,438	4,907	16,880	unknown
mining	open	753,404	-65,720,600	902,969,000	unknown
ns-1853823	open	213,440	2,820,000	4,670,000	unknown
pb-simp-nonunif	open	23,848	90	140	unknown
ramos3	open	2,187	223	274	unknown
rmine14	open	32,205	-3,469	-170	unknown
rmine21	open	162,547	-3,657	-184	unknown
rmine25	open	326,599	-3,052	-161	unknown
siena1	open	13,741	256,620,000	315,186,152	unknown
sts405	open	405	342	342	unknown
sts729	open	709	648	648	unknown

To conclude, we invite the readers to have a look to the LocalSolver Example Tour[13] to find more results obtained using LocalSolver on difficult problems. In particular, in the recent versions 3.0 and 3.1 of LocalSolver, some moves dedicated to combinatorial structures encountered in scheduling or routing problems have been designed. This allows to obtain good-quality results in short running times on traveling salesman problems[14] and vehicle routing problems[15].

[12]`http://miplib.zib.de`
[13]`http://www.localsolver.com/exampletour.html`
[14]`http://www.localsolver.com/exampletour.html?file=tsp.zip`
[15]`http://www.localsolver.com/exampletour.html?file=vrp.zip`

Chapter 4

Toward an optimization solver based on neighborhood search

In the previous chapter, we have shown that 0-1 nonlinear programming can be efficiently tackled by local search. Exploiting pure and direct local search allows us to tackle combinatorial problems which are out of scope of the state-of-the-art tree search solvers (MIP or CP). This was made possible by designing fast, scalable local moves tending to maintain the search into feasible regions. The core idea behind these local moves is to modify the current value of some decisions while repairing violated constraints. We conclude this dissertation by outlining our current work on LocalSolver and more generally by presenting the roadmap of the LocalSolver project toward a generalized, all-in-one, hybrid mathematical programming solver based on neighborhood search.

4.1 Using neighborhood search as global search strategy

As explained previously, pure tree search approaches suffer from several drawbacks, which makes them ineffective for large-scale optimization. Let us cite Fischetti and Monaci [47] about tree search: “Our working hypothesis is that erraticism is in fact just a consequence of the exponential nature of tree search, that acts as a chaotic amplifier, so it is largely unavoidable.” This explains the reason why today tree search solvers (MIP, CP, SAT) integrate more and more heuristic ingredients related to local search [36, 44, 45, 46, 107]. In modern solvers, tree search is mainly used to perform large neighborhood exploration around some promising solutions (see for example [44] or [107]). Since relaxation or propagation are expensive to compute, particularly for large-scale problems, variable-fixing heuristics [17] are intensively used at each node of the tree search to find feasible integer solutions.

We propose a general-purpose optimization solver founded on a radically different architecture. Instead of embedding local search ingredients into tree search, we propose to use neighborhood search as global search strategy. Nevertheless, neighborhood search does not mean to restrict the search to small neighborhoods and to abandon tree search. Very large-scale neighborhoods, namely neighborhoods of exponential size, can be explored through tree search or specific algorithms (see our papers [40, 41] for example). This way of searching follows the methodology exposed in Section 2.1. The type and the size of the neighborhood explored at each iteration are dynamically adapted along the search. As shown in the previous chapter, small-neighborhood moves – mainly running in $O(1)$ time – are critical for the velocity of the search: they speed up the convergence toward good-quality feasible solutions and allows to scale when facing problems with millions of variables. Then, if exploring small neighborhoods fails to improve the best incumbent or to diversify the search (which occurs when the search is trapped into a local optimum or confined to a hardly constrained subspace), then larger neighborhoods can be explored to escape the current solution. As explained in Section 2.1, this can be done by using compound moves or even large neighborhood moves based on tree search [101] or specific algorithms [3]. Such moves are more expensive in running time, so they have to be used parsimoniously, ideally only when it is necessary. Note that when the number of decisions is not so large, some large neighborhood moves based on tree search can handle all decisions of the problem: in this case, the neighborhood explored corresponds to the entire solution space, making the search complete.

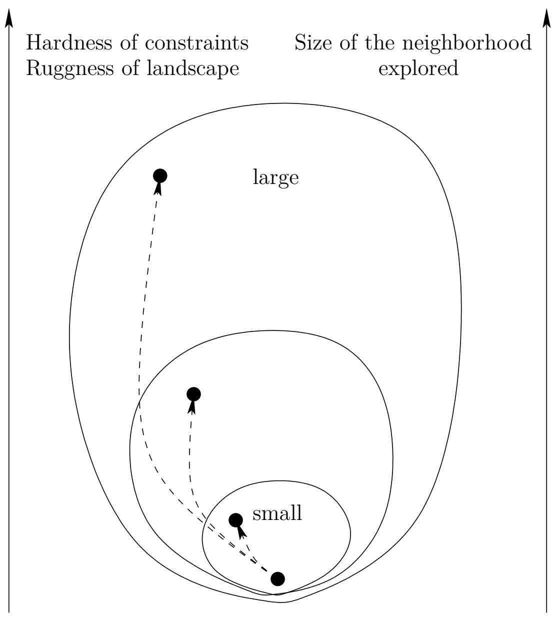

Figure 4.1: Enlarging or retracting the neighborhood to explore, during the search.

An important ingredient is the randomization of the whole search process, in particular the exploration of the neighborhoods. Even if small neighborhood moves are sufficient to progress, some large neighborhood moves can also be called to diversify the search, with a low probability. Moves are applied randomly following a distribution evolving during the search. This distribution is changed based on the acceptance and improvement rates of each kind of moves and of the time complexity of these ones. Then, the top-layer heuristic which pilots the search allows several levels of diversification in addition to the moves. A strategy mixing simulated annealing with thresholds [1] is used to escape from local optima for small neighborhoods. In case of very chaotic landscapes, the search can be restart from feasible points (reheating) or from infeasible ones (restart). Finally, the search is multithreaded. It consists in running different searches in parallel and synchronizing them regularly.

In this way, the neighborhood search strategy implemented in LocalSolver generalizes and unifies many known ideas and concepts: variable neighborhood search [58], multiple neighborhood search [111], autonomous search [56]. Coupling simulated annealing, multiple neighborhood search, and hyper-heuristic learning mechanism, the search strategy recently proposed by Bai et al. [6, 7] is close to the one implemented in LocalSolver. Note that this kind of local/global search strategy is heavily used in global optimization to practically solve non-convex continuous or mixed-variable problems (see [72] for more details).

4.2 Extension to continuous and mixed optimization

Local search is a technique which is actually used in continuous optimization, under another naming: *direct search*. The technique was introduced in the sixties for unconstrained nonlinear programming (with continuous decisions) through the "other simplex method", namely the Nelder-Mead simplex algorithm [85, 118]. Direct search [70] refers to *derivative-free or zeroth-order methods*, in contrast to first-order methods requiring the computation of gradients like quasi-Newton methods, or second-order methods requiring the computation of the Hessian matrix like Newton methods (see [79] for more details). Reading the recent survey by Kolda et al. [70], the parallel between local search methods in combinatorial optimization and direct search methods in continuous optimization is striking: the apparent conceptual simplicity of the approach, the notion of "moves" applied at each iteration of the search (see [70, p. 389]), the fast evaluation of each iteration because directly based on the objective, as well as the lack of mathematical analysis explaining its practical efficiency. Surprisingly, the similarity does not stop to technicalities: "Mathematicians hate it because you can't prove convergence; engineers seem to love it because it often works" as commented by John Nelder himself [118, p. 274].

On the other hand, similarly to local search in combinatorial optimization, direct search appears to be limited to tackle constrained problem, even if some progresses have been made recently [70]. We are currently working on *unifying local search for combinatorial optimization and direct search for continuous optimization*. In particular, we have adapted the local search moves described in the previous chapter to handle continuous decisions. The ideas behind these local continuous (or even mixed-variable) moves remain the same: we tend to maintain the feasibility of the move by repairing violated constraints, we exploit incremental calculations to evaluate the move quickly. In summary, we are working to make LocalSolver able to tackle *large-scale mixed-variable non-convex optimization problems* involving both 0-1 and continuous decisions. In the same way than for combinatorial optimization, some moves exploring larger neighborhoods will be designed based on first-order (linear) or second-order (convex) approximations, as done in interior point methods or sequential linear/quadratic programming for nonlinear programming [71].

More generally, our goal with this research line is *to promote the use of direct/local search, and more generally of approximate computing techniques [83, 84], as primary techniques to search and optimize efficiently in any kind of solution spaces, in particular to cope with very large-scale, highly non-convex problems encountered in real-world optimization.* This approach is radically different from the mainstream in mathematical programming.

Despite some attempts like the use of pattern search or variable neighborhood search in global optimization [70, 72, 73], most of the works in combinatorial or continuous optimization were concentrated on the design of sophisticated techniques providing *guarantees* in terms of convergence toward optimal solutions or locally optimal ones. Such mathematical guarantees have a price: these techniques involve heavy computations at each step, leading to slow or even impractical algorithms for large-scale real-world optimization. For example, many state-of-the-art optimization algorithms follow a best-improvement approach, in the sense that they search *at each step* the best next iterate (in particular, the best direction to follow). For example, at each iteration of the simplex algorithm for linear programming, the best swap of columns (one entering the basis and another one leaving it) requires to invert the basis matrix and to compute the reduced costs for all non-basic variables, basically involving $O(m^2 + nm)$ multiplications with n (resp. m) the number of variables (resp. constraints) [34]. In convex programming, while the number of iterations of many interior point algorithms is reduced to roughly $O(\sqrt{n})$ in the worst case, the practical complexity of each iteration is still much more expensive than one of the simplex algorithm. Indeed, the computation of the Newton direction requires $O(n^3)$ time with n the number of variables [10, pp. 396–397]. In combinatorial optimization, algorithms for maximum flow or maximum matching computes at each iteration the best augmenting path, requiring a $O(m)$-time breadth-first search with m the number of edges in the graph [34]. In integer programming, tree search approaches rely on the computation of a lower bound at each node, based on the *exact* linear relaxation reinforced by cut generation. Table 4.1 gives a more explicit comparison of the practical efficiency of some of the search paradigms described above.

Table 4.1: Comparison of the "velocity" of different search techniques to tackle a real-life continuous linear problem (LP) involving nearly 100,000 variables and 1 million nonzeros.

	Iterations per minute	Speedup factor
LP interior point	≈ 10	-
LP simplex	$\approx 10,000$	$\times 1000$
Small-neighborhood search	$\approx 10,000,000$	$\times 1,000,000$

Our idea is to use direct local search to lower the practical time complexity of one iteration, average and amortized over all the search, in any kind of solution space. Indeed, if a lightweight move suffices to make a step in the right direction (that is, to improve the incumbent), why not performing it? The idea is to use heavier moves exploring larger neighborhoods (possibly changing more coordinates of the solution vector) to diversify or intensify the search, in particular when the lightweight moves become ineffective. As explained above, this can be made adaptively and randomly all along the search, leading to two major advantages in practice. First, it speeds up the convergence toward good-quality feasible solutions. Then, it allows to scale when facing ultra-large instances, as encountered today in the practice of optimization.

4.3 Separating the computation of solutions and bounds

According to us, finding feasible solutions (that is, computing upper bounds) and proving optimality (that is, computing lower bounds) are different tasks, requiring different approaches. Modern MIP solvers use primal feasibility heuristics [17] to quickly produce feasible solutions *before* starting the tree search. Then, these two tasks are handled *separately* in LocalSolver. On one hand, feasible solutions are searched and optimized through a local/global search approach, as described in the previous section. On the other hand, lower bounds are improved through relaxation and inference techniques embedded in a divide-and-conquer scheme. Information is exchanged between the two calculations to enhance both. Our goal is to obtain exponential-inverse convergence patterns for both computations, even when facing very large-scale problems.

Relaxation techniques take their roots in the field of mathematical programming. The idea is to relax the original problem such that the relaxed problem can be solved efficiently, to obtain a lower bound of the optimal solution to the original problem. So far, (continuous) linear problems are the ones we practically solve the better today. This explains the success of mixed-integer *linear* programming solvers, relying on the computation of the linear relaxation by the simplex algorithm. A line of research has been initiated twenty years ago to take advantage of convex relaxations, in particular semidefinite ones [113], for combinatorial optimization [77] or more generally mixed-variable non-convex optimization [72]. On the other hand, inference and constraint propagation techniques have been developed in the field of artificial intelligence. Inference techniques are the heart of SAT solvers (see [57] for a survey on conflict learning). Extending inference techniques to problems having decisions over arbitrary domains (and not only boolean ones), constraint propagation is the heart of CP solvers (see [19] for a survey). Many ideas developed in the SAT and CP fields can be integrated in a general-purpose mathematical optimization solver like LocalSolver. It is a good way to improve the optimality gap when relaxation techniques are failing, which is generally the case when the problem is highly combinatorial [18]. As pointed in [57], resolution techniques in SAT are related to cutting plane generation techniques in MIP.

We have started to integrate both relaxation and inference techniques in LocalSolver to compute lower bounds. In the previous chapter, some results presented on Table 3.1 related to the car sequencing problem show the ability of LocalSolver to prove optimality. First, a dual linear relaxation of the input model is automatically generated and heuristically solved by the local/global search techniques presented in the previous section. Using a dual relaxation instead of a primal one allows to exploit an iterative improvement solution approach, since every dual feasible solution induces a lower bound. The idea is to follow the same scheme than the one advocated for computing feasible solutions of the original problem: using approximate resolution techniques to speed up the bounding and to be able to scale when necessary. Similar ideas have been recently introduced through the so-called primal-dual variable neighborhood search [59, 60]. In the spirit of the line of research initiated by Malick and Roupin [77], the next step will be to reinforce the linear bounds by using dual convex relaxations. On the other hand, we have integrated inference techniques relying on identifying richer discrete structures (global constraints). This bounding process – relaxation and propagation – will be embedded in a divide-and-conquer process (that is, a branching process dealing with discrete and continuous domains), defining an iterative improvement scheme to reduce the optimality gap. To go further, resolution techniques (cutting plane generation or conflict learning) have to be integrated in this process to tighten relaxation and propagation, in particular for combinatorial optimization. To deal with continuous domains, we plan to apply interval propagation and filtering techniques combining interval analysis and constraint programming features (see [31, 66] for details).

4.4 A new-generation math programming solver

In this last section, we outline the target architecture of LocalSolver over the long term. Our vision of mathematical programming and then the future of LocalSolver can be summarized by citing the conclusion of a prospective paper by John N. Hooker [63] written in 2007 about the future of constraint programming and operations research: "Since modeling is the master and computation the servant, no computational method should presume to have its own solver. This means there should be no CP solvers, no MIP solvers, and no SAT solvers. All of these techniques should be available in a single system to solve the model at hand. They should seamlessly combine to exploit problem structure. Exact methods should evolve gracefully into inexact and heuristic methods as the problem scales up."

Following the previous discussions, the target architecture of LocalSolver is drawn on Figure 4.2. Through this architecture, *we unify heuristic and exact optimization approaches in two ways*: first by adopting a *generalized (that is, multiple, variable, adaptive) neighborhood search approach* – from small neighborhoods explored in a heuristic fashion to large neighborhoods explored in an exact fashion – then by *dissociating the search and the optimization of feasible solutions (search into the primal solution space) from the computation of lower bounds aiming at proving optimality or infeasibility (search into the dual solution space).* This way of unifying mathematical optimization techniques extends the recent ideas independently suggested by Hooker in a forthcoming paper [65], while being consistent with his (and our) vision of the future of mathematical programming described above.

Planned for the end of 2013, the next version 4.0 will be a first step toward this new-generation mathematical programming solver for *large-scale mixed-variable non-convex optimization*, hybridizing all appropriate optimization techniques : mixed-integer and nonlinear programming techniques, constraint programming and satisfiability techniques, local and direct search techniques. Indeed, the next versions will progressively offer several important features from both functional and technical points of view: small-neighborhood moves to optimize over continuous or mixed decisions; exploration of large, exponential-size neighborhoods over 0-1 or mixed decisions using tree search techniques (for example, rounding heuristics based on linear relaxation); exploration of large neighborhoods over continuous decisions by revisiting successive linear programming techniques for nonlinear programming (based on a simplex algorithm); computation of lower bounds combining constraint propagation and dual linear relaxation.

As example, we provide some preliminary results on the unit commitment problem [88] as addressed in Benoist [13, pp. 25–39], which is a celebrated mixed-decision nonlinear (convex quadratic) optimization. The unit commitment problem is an optimization problem used to determine the operation schedule of the generating units at every hour interval with varying loads under different constraints and environments. This problem is traditionally solved using Lagragian relaxation approaches [88]. On Table 4.2, we give the results obtained after 5 minutes of running time using a straightforward LocalSolver model (in version 4.0 beta). The state-of-the-art results presented on this table have been obtained by a heuristic based on Lagragian relaxation [13, pp. 21–33].

Table 4.2: Results on a unit commitment problem.

Instances	Time steps	Units	Decisions	Expressions	LocalSolver 4.0 beta	State of the art	Gap
ucp0	8	4	64	850	75,037	74,988	0,07 %
ucp3	24	10	480	5,946	572,558	563,719	1,57 %
ucp4	24	20	960	11,709	1,160,470	1,122,042	3,42 %
ucp5	24	40	1,920	23,230	2,403,470	2,237,824	7,40 %
ucp6	24	60	2,880	34,748	3,544,590	3,355,191	5,64 %
ucp7	24	80	3,840	46,270	4,765,150	4,470,455	6,59 %
ucp8	24	100	4,800	57,788	5,984,610	5,587,788	7,10 %

Over the long term, we plan to add *integer decisions and set operators* in the LocalSolver formalism to make the modeling of routing and scheduling problems easier (and the resolution more efficient, in particular about space complexity). Nevertheless, mixing this richer formalism with the classical 0-1 decision formalism is delicate for users (see for example the proposals by Hooker et al. [64, 119] on this topic). Indeed, integers can be used as quantitative variables or as indexers for set operators.

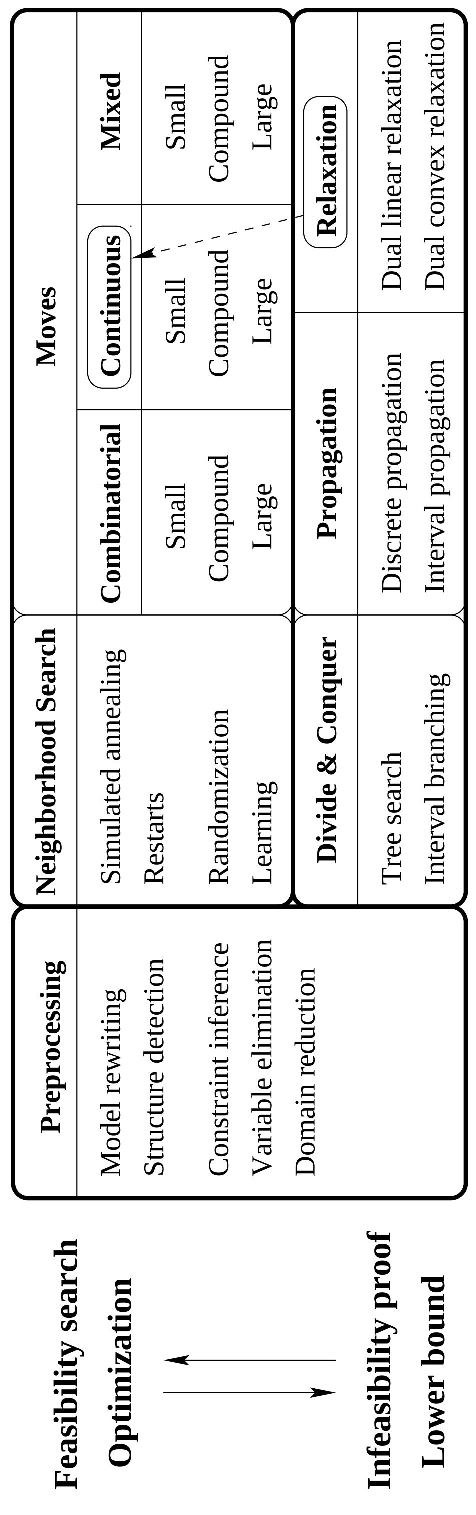

Figure 4.2: Target architecture of LocalSolver.

Bibliography

[1] E. Aarts, J.K. Lenstra (2003). *Local Search in Combinatorial Optimization*. Princeton University Press. (2nd edition)

[2] D. Abramson, M. Randall (1999). A simulated annealing code for general integer linear programs. *Annals of Operations Research* 86, pp. 3–21.

[3] R.K. Ahuja, Ö. Ergun, J.B. Orlin, A.P. Punnen (2002). A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics* 123, pp. 75–102.

[4] D.L. Applegate, R.E. Bixby, V. Chvátal, W. Cook, D.G. Espinoza, M. Goycoolea, K. Helsgaun (2009). Certification of an optimal TSP tour through 85,900 cities. *Operations Research Letters* 37, pp. 11–15.

[5] C. Archetti, M. Savelsbergh (2009). The trip scheduling problem. *Transportation Science* 43(4), pp. 417–431.

[6] R. Bai, J. Blazewicz, E.K. Burke, G. Kendall, E.K. Burke, B. McCollum (2012). A simulated annealing hyper-heuristic methodology for flexible decision support. *4OR* 10(1), pp. 43–66.

[7] R. Bai, T. Van Woensel, G. Kendall, E.K. Burke (2013). A new model and a hyper-heuristic approach for two-dimensional shelf space allocation. *4OR* 11(1), pp. 31–55.

[8] L. Barnett (2001). Netcrawling: optimal evolutionary search with neutral networks. In *Proceedings of the 2001 IEEE Congress on Evolutionary Computation*, pp. 30–37. IEEE Press.

[9] M.S. Bazaraa, H.D. Sherali, C.M. Shetty (2006). *Nonlinear Programming: Theory and Algorithms*. John Wiley & Sons.

[10] A. Ben-Tal, A. Nemirovski (2001). *Lectures on Modern Convex Optimization*. MPS-SIAM Series on Optimization. SIAM.

[11] W. Bell, L. Dalberto, M. Fisher, A. Greenfield, R. Jaikumar, P. Kedia, R. Mack, P. Prutzman (1983). Improving the distribution of industrial gases with an on-line computerized routing and scheduling optimizer. *Interfaces* 13(6), pp. 4–23.

[12] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, A. Mahajan (2013). Mixed-integer nonlinear optimization. *Acta Numerica* 22, pp. 1–131.

[13] T. Benoist (2007). Décomposition combinatoire et applications industrielles. Collection Programmation par Constraintes. Hermes Science - Lavoisier.

[14] T. Benoist, B. Estellon, F. Gardi, A. Jeanjean (2009). Randomized local search for real-life inventory routing. *Transportation Science* 45(3), pp. 381–398.

[15] T. Benoist, B. Estellon, F. Gardi, R. Megel, K. Nouioua (2011). LocalSolver 1.x: a black-box local-search solver for 0-1 programming. *4OR* 9(3), pp. 299-316.

[16] T. Benoist, F. Gardi, A. Jeanjean (2009). Lessons learned from 15 years of operations research for French TV channel TF1. *Interfaces* 42(6), pp. 577–584.

[17] T. Berthold (2006). Primal heuristics for mixed integer programs. Diploma Thesis, Technische Universität Berlin. Konrad-Zuse-Zentrum für Informationstechnik Berlin.

[18] T. Berthold, S. Heinz, M.E. Pfetsch (2009). Nonlinear pseudo-Boolean optimization: relaxation or propagation?. In *Proceedings of SAT 2009*, LNCS 5584, pp. 441–446. Springer.

[19] C. Bessière (2006). Constraint propagation. Technical Report LIRMM 06020. CNRS/University of Montpellier, France.

[20] R.E. Bixby (2012). A brief history of linear and mixed-integer programming computation. In *Optimization Stories, 21st ISMP Berlin 2012*, pp. 107–121. Documenta Mathematica.

[21] J.F. Bonnans, J.C. Gilbert, C. Lemaréchal, C.A. Sagastizàbal (2000). *Numerical Optimization: Theoretical and Practical Aspects.* Springer.

[22] R.E. Burkard, S.E. Karisch, F. Rendl (1997). QAPLIB – a quadratic assignment problem library. *Journal of Global Optimization* 10(4), pp. 391–403. `http://www.seas.upenn.edu/qaplib`

[23] S. Burer, A.N. Letchford (2012). Non-convex mixed-integer nonlinear programming: a survey. *Surveys in Operations Research and Management Science* 17(2), pp. 97–106.

[24] M.R. Bussieck, S. Vigerske (2011). MINLP solver software. *Wiley Encyclopedia of Operations Research and Management Science*, pp. 95–113. John Wiley & Sons.

[25] A. Campbell, L. Clarke, A. Kleywegt, M. Savelsbergh (1998). The inventory routing problem. *Fleet Management and Logistics*, pp. 95–113. Kluwer Academic Publishers.

[26] A. Campbell, L. Clarke, M. Savelsbergh (2002). Inventory routing in practice. *The Vehicle Routing Problem*, SIAM Monographs on Discrete Mathematics and Applications, Vol. 9, pp. 309–330. SIAM.

[27] S. Cahon, N. Melab, E.-G. Talbi (2004). ParadisEO: a framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics* 10(3), pp. 357–380.

[28] A. Campbell, M. Savelsbergh (2004). A decomposition approach for the inventory-routing problem. *Transportation Science* 38(4), pp. 488–502.

[29] A. Campbell, M. Savelsbergh (2004). Delivery volume optimization. *Transportation Science* 38(2), pp. 210–223.

[30] A. Campbell, M. Savelsbergh (2004). Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science* 38(3), pp. 369–378.

[31] G. Chabert, L. Jaulin (2009). Contractor programming. *Artificial Intelligence* 173(11), pp. 1079–1100.

[32] D. Connolly (1992). General purpose simulated annealing. *Journal of the Operational Research Society* 43, pp. 495–505.

[33] W. Cook (2012). Markowitz and Manne + Eastman + Land and Doig = Branch and Bound. In *Optimization Stories, 21st ISMP Berlin 2012*, pp. 227–238. Documenta Mathematica.

[34] T. Cormen, C. Leiserson, R. Rivest, C. Stein (2004). *Introduction à l'Algorithmique.* Dunod. (French 2nd edition)

[35] C. D'Ambrosio, A. Lodi (2011). Mixed integer nonlinear programming tools: a practical overview. *4OR* 9(4), pp. 329–349.

[36] E. Danna, E. Rothberg, C. Le Pape (2005). Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming Series A* 102(1), pp. 71–90.

[37] L. Di Gaspero, A. Schaerf (2003). EasyLocal++: an object-oriented framework for flexible design of local search algorithms. *Software - Practice & Experience* 33(8), pp. 733–765.

[38] Dynadec Decision Technologies (2010). Comet 2.1 Tutorial, 581 pages.
`http://www.dynadec.com`

[39] B. Estellon, F. Gardi. Car sequencing is NP-hard: a short proof. (to appear in *Journal of the Operational Research Society*)

[40] B. Estellon, F. Gardi, K. Nouioua (2006). Large neighborhood improvements for solving car sequencing problems. *RAIRO Operations Research* 40(4), pp. 355–379.

[41] B. Estellon, F. Gardi, K. Nouioua (2008). Two local search approaches for solving real-life car sequencing problems. *European Journal of Operational Research* 191(3), pp. 928–944.

[42] B. Estellon, F. Gardi, K. Nouioua (2009). High-performance local search for task scheduling with human resource allocation. In *Proceedings of SLS 2009*. LNCS 5752, pp. 1–15. Springer.

[43] R. Fourer (2012). On the evolution of optimization modeling systems. In *Optimization Stories, 21st ISMP Berlin 2012*, pp. 377–388. Documenta Mathematica.

[44] M. Fischetti, A. Lodi (2003). Local branching. *Mathematical Programming Series B* 98(1-3), pp. 23–47.

[45] M. Fischetti, A. Lodi (2008). Repairing MIP infeasibility through local branching. *Computers and Operations Research* 35(5), pp. 1436–1445.

[46] F. Focacci, F. Laburthe, A. Lodi (2003). Local search and constraint programming. In *Handbook of Metaheuristics*, pp. 369–403. International Series in Operations Research and Management Science, Vol. 57. Kluwer Academic Publishers.

[47] M. Fischetti, M. Monaci (2012). Exploiting erraticism in search. (submitted to *Operations Research*)

[48] F. Gardi (2012). High-performance local search for TV media planning on TF1. In *EURO 2012*, Vilnius.

[49] F. Gardi, K. Nouioua (2003). Local search for mixed-integer nonlinear optimization: a methodology and an application. In *Proceedings of EvoCOP 2011*, LNCS 6622, pp. 167–178. Springer.

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

[51] F.W. Glover, G.A. Kochenberger (2003). *Handbook of Metaheuristics.* International Series in Operations Research and Management Science, Vol. 57. Kluwer Academic Publishers.

[52] A. Goel (2010). Truck driver scheduling in the European Union. *Transportation Science* 44(4), pp. 429–441.

[53] A. Goel, L. Kok (2009). Truck driver scheduling in the United States. *Transportation Science* 43(3), pp. 317–326.

[54] J. Gottlieb, M. Puchta, C. Solnon (2003). A study of greedy, local search and ant colony optimization approaches for car sequencing problems. In *Proceedings of EvoWorkshops 2003*, LNCS 2611, pp. 246–257. Springer.

[55] M. Gravel, C. Gagné, W.L. Price (2005). Review and comparison of three methods for the solution of the car sequencing problem. *Journal of the Operational Research Society* 56, pp. 1287–1295.

[56] Y. Hamadi, E. Monfroy, F. Saubion (2012). *Autonomous Search.* Springer.

[57] Y. Hamadi, S. Jabbour, L. Sais (2008). Learning from conflicts in propositional satisfiability. *4OR* 10(1), pp. 15–32.

[58] P. Hansen, N. Mladenović, J.A. Moreno Pérez (2008). Variable neighborhood search: methods and applications. *4OR* 6(4), pp. 319–360.

[59] P. Hansen, J. Brinberg, Dragan Uroševic, N. Mladenović (2007). Primal-dual variable neighborhood search for the simple plant-location problem. *INFORMS Journal on Computing* 19(4), pp. 552–564.

[60] P. Hansen, J. Brinberg, Dragan Uroševic, N. Mladenović (2009). Solving large p-median clustering problems by primal-dual variable neighborhood search. *Data Mining and Knowledge Discovery* 19(3), pp. 351–375.

[61] K. Helsgaun (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research* 126(1), pp. 106–130.

[62] K. Helsgaun (2009). General k-opt submoves for the Lin-Kernighan TSP heuristic. *Mathematical Programming Computation* 1, pp. 119–163.

[63] J.N. Hooker (2007). Good and Bad Futures for Constraint Programming (and Operations Research). *Constraint Programming Letters* 1, pp. 21–32.

[64] J.N. Hooker. Hybrid modeling. In *Hybrid Optimization: The Ten Years of CPAIOR*, pp. 11–62. Springer.

[65] J.N. Hooker. Toward unification of exact and heuristic optimization methods. (to appear in *International Transactions in Operations Research*

[66] L. Jaulin, M. Kieffer, O. Didrit, E. Walter (2001). *Applied Interval Analysis with Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer.

[67] A. Jeanjean (2011). Recherche locale pour l'optimisation en variables mixtes : méthodologie et applications industrielles. PhD thesis, École Polytechnique, Palaiseau.

[68] D.F. Jones, M. Tamiz (2010). *Practical Goal Programming*. International Series in Operations Research and Management Science, Vol. 141. Springer.

[69] T. Kis (2004). On the complexity of the car sequencing problem. *Operations Research Letters* 32, pp. 331–335.

[70] T.G. Kolda, R.M. Lewis, V. Torczon (2003). Optimization by direct search: new perspectives on some classical and modern methods. *SIAM Review* 45(3), pp. 385–482.

[71] S. Leyffer, A. Mahajan (2010). Nonlinear constrained optimization: methods and software. Preprint ANL/MCS-P1729-0310, Argonne National Laboratory, IL.

[72] L. Liberti, N. Maculan (2006). *Global Optimization: from Theory to Implementation*. Series in Nonconvex Optimization and Its Applications, Vol. 84. Springer.

[73] L. Liberti, N. Mladenović, G. Nannicini (2011). A recipe for finding good solutions to MINLPs. *Mathematical Programming Computation* 3(4), pp. 349–390.

[74] A. Lodi (2013). The heuristic (dark) side of MIP solvers. In *Hybrid Metaheuristics*, Studies in Computational Intelligence, Vol. 434, pp. 273–284. Springer.

[75] A. Løkketangen (2007). The importance of being careful. In *Proceedings of SLS 2007*, LNCS 4638, pp. 1–15. Springer.

[76] Z. Lü, F. Glover, J.-K. Hao. (2009). Neighborhood combination for unconstrained binary quadratic problems. In *Proceedings of MIC 2009*, pp. 49–61. Springer.

[77] J. Malick, F. Roupin. On the bridge between combinatorial optimization and nonlinear optimization: new semidefinite bounds for 0-1 quadratic problems leading to quasi-Newton methods. (to appear in *Mathematical Programming Series B*)

[78] L. Michel, P. Van Hentenryck (2000). Localizer. *Constraints* 5(1-2), pp. 43–84.

[79] M. Minoux (2007). *Programmation Mathématique : Théorie et Algorithmes.* Éditions Tec & Doc, Lavoisier. (2nd edition, in French)

[80] B.M.E. Moret (2002). Towards a discipline of experimental algorithmics. In *Data Structures, Near Neighbor Searches, and Methodology: 5th and 6th DIMACS Implementation Challenges*, DIMACS Monographs, Vol. 59, pp. 197–213. AMS.

[81] B.M.E. Moret, D.A. Bader, T. Warnow (2002). High-performance algorithm engineering for computational phylogenetics. *Journal of Supercomputing* 22(1), pp. 99–111.

[82] B.M.E. Moret, H.D. Shapiro (2001). Algorithms and experiments: the new (and old) methodology. *Journal of Universal Computer Science* 7(5), pp. 434–446.

[83] R. Nair (2008). Approximate computing. In *Symposium on Computing Challenges 2008*. Ithaca, NY.

[84] R. Nair, D.A. Prener (2008). Computing, approximately. In *Wild and Crazy Ideas VI, ASPLOS 2008*. Seattle, WA.

[85] J.A. Nelder, R. Mead (1965). A simplex method for function minimization. *Computer Journal* 7, pp. 308–313.

[86] J. Nocedal, S.J. Wright (2006). *Numerical Optimization.* Springer Series in Operations Research. Springer. (2nd edition)

[87] K. Nonobe, T. Ibaraki (1998). A tabu search approach to the constraint satisfaction problem as a general problem solver. *European Journal of Operational Research* 106(2-3), pp. 599–623.

[88] N.P. Padhy (2004). Unit commitment – a bibliographical survey. *IEEE Transactions on Power Systems* 19(2), pp. 1196–1205.

[89] J.-F. Puget (2004). Constraint programming next challenge: simplicity of use. In *Proceedings of CP 2004*, LNCS 3258, pp. 5–8. Springer.

[90] P. Pellegrini, M. Birattari (2007). Implementation effort and performance. In *Proceedings of SLS 2007*, LNCS 4638, pp. 31–45. Springer.

[91] L. Perron, P. Shaw (2004). Combining forces to solve the car sequencing problem. In *Proceedings of CPAIOR 2004*, LNCS 3011, pp. 225–239. Springer.

[92] L. Perron, P. Shaw, V. Furnon (2004). Propagation guided large neighborhood search. In *Proceedings of CP 2004*, LNCS 3258, pp. 468–481. Springer.

[93] D. Pisinger, S. Ropke (2010). Large neighborhood search. In *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, Vol. 146, pp. 399–419. Springer.

[94] M. Prandtstetter, G.R. Raidl (2008). An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem. *European Journal of Operational Research* 191(3), pp. 1004–1022.

[95] M. Puchta, J. Gottlieb (2002). Solving car sequencing problems by local optimization. In *Proceedings of EvoWorkshops 2002*, LNCS 2279, pp. 132–142. Springer.

[96] G. Ramalingam (1996). *Bounded Incremental Computation.* Lecture Notes in Computer Science, Vol. 1089. Springer.

[97] M. Randall, D. Abramson (2001). A general meta-heuristic based solver for combinatorial optimization problems. *Combinatorial Optimisation and Applications* 20, pp. 185–210.

[98] C. Rego, F. Glover (2002). Local search and metaheuristics. In *The Traveling Salesman Problem and Its Variations*, pp. 105–109. Kluwer Academic Publishers.

[99] C.M. Reidys and P.F. Stadler (2002). Combinatorial landscapes. *SIAM Review* 44(1), pp. 3–54.

[100] D.S. Rosenblum (1992). Towards a method of programming with assertions. In *Proceedings of ICSE 1992, the 14th International Conference on Software Engineering*, pp. 92–104. ACM Press.

[101] E. Rothberg (2007). An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing* 19(4), pp. 534–541.

[102] M. Savelsbergh, J.-H. Song (2007). Inventory routing with continuous moves. *Computers and Operations Research* 34(6), pp. 1744–1763.

[103] M. Savelsbergh, J.-H. Song (2008). An optimization algorithm for the inventory routing with continuous moves. *Computers and Operations Research* 35(7), pp. 2266–2282.

[104] G. Schrimpfa, J. Schneiderb, H. Stamm-Wilbrandta, G. Duecka (2000). Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics* 159(2), pp. 139–171.

[105] B. Selman, H.J. Levesque, D. Mitchell (1992). A new method for solving hard satisfiability problems. *Proceedings of AAAI 1992*, pp. 440–446. AAAI Press.

[106] B. Selman, H. Kautz, B. Cohen (1996). Local search strategies for satisfiability testing. In *Cliques, Coloring, and Satisfiability: 2nd DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol 26. AMS.

[107] P. Shaw (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of CP 1998*, LNCS 1520, pp. 417–431. Springer.

[108] C. Solnon (2000). Solving permutation constraint satisfaction problems with artificial ants. In *Proceedings of ECAI 2000*, pp. 118–122. IOS Press.

[109] C. Solnon, V.-D. Cung, A. Nguyen, C. Artigues (2008). The car sequencing problem: overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem. *European Journal of Operational Research* 191(3), pp. 912–927.

[110] K. Sörensen (2012). Metaheuristics: the metaphor exposed. In *EURO 2012 Tutorials*, Vilnius.

[111] K. Sörensen, M. Sevaux, P. Schittekat (2008). "Multiple neighbourhood" search in commercial VRP packages: evolving towards self-adaptive methods. In *Adaptive and Multilevel Metaheuristics*, Studies in Computational Intelligence, Vol. 136, pp. 239–253. Springer.

[112] D.A. Spielman, S.-H. Teng (2004). Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. *Journal of the ACM* 51(3), pp. 385–463.

[113] L. Vandenberghe, S. Boyd (1996). Semidefinite programming. *SIAM Review* 38(1), pp. 49–95.

[114] P. Van Hentenryck, L. Michel (2005). *Constraint-based local search.* MIT Press.

[115] C. Voudouris, R. Dorne, D. Lesaint, A. Liret (2001). iOpt: a software toolkit for heuristic search methods. In *Proceedings of CP 2001*, LNCS 2239, pp. 716–730.

[116] J. Walser (1999). *Integer optimization by local search: a domain-independent approach.* Lecture Notes in Articial Intelligence, Vol. 1637. Springer.

[117] M.H. Wright (2004). The interior-point revolution in optimization: history, recent developments, and lasting consequences. *Bulletin of the American Mathematical Society* 42(1), pp. 39–56.

[118] M.H. Wright (2012). Nelder, Mead, and the other simplex method. In *Optimization Stories, 21st ISMP Berlin 2012*, pp. 271–276. Documenta Mathematica.

[119] T. Yunes, I. Aron, J.N. Hooker (2010). An integrated solver for optimization problems. *Operations Research* 58, pp. 342–356.

[120] L. Zhang, S. Malik (2003). Cache performance of SAT solvers: a case study for efficient implementation of algorithms. In *Proceedings of SAT 2003*, LNCS 2919, pp. 287–298. Springer.

Bouygues e-lab
32 avenue Hoche
75008 Paris, France

fgardi@bouygues.com
`http://e-lab.bouygues.com`
`http://www.bouygues.com`

Innovation 24 & LocalSolver
24 avenue Hoche
75008 Paris, France

fgardi@innovation24.fr
`http://www.innovation24.fr`

fgardi@localsolver.com
`http://www.localsolver.com`

Faculté d'Ingénierie (UFR 919)
Université Pierre et Marie Curie (Paris 6)
4 place Jussieu, case 175
75252 Paris cedex 5, France

École Doctorale Informatique, Télécommunications
et Électronique de Paris (EDITE – ED 130)
Université Pierre et Marie Curie (Paris 6)
4 place Jussieu, case 168
75252 Paris cedex 5, France

Abstract

This dissertation deals with *local search for combinatorial optimization* and its extension to mixed-variable optimization. Our goal is to present local search in a new light. Although not yet understood from the theoretical point of view, local search is the paradigm of choice to tackle large-scale real-life optimization problems. Today end-users ask for interactivity with decision support systems. For optimization software, it means obtaining good-quality solutions quickly. Fast iterative improvement methods, like local search, are suited to satisfy such needs.

When a solution space is gigantic, a complete search becomes unpractical. Given a (possibly infeasible) solution to the problem, local search consists in modifying some parts of this one – that is, some decision variables – to reach a new, hopefully better solution. Exploring a so-called *neighborhood* of the incumbent has a major advantage: the new solution can be evaluated quickly through *incremental calculation*. Then, local search can be viewed as an incomplete and nondeterministic but efficient way to explore a solution space.

First, an iconoclast *methodology* is presented to design and engineer local search algorithms. We show that the performance of a local search mainly relies on the richness of the neighborhoods explored, as well as on the efficiency of their exploration. Ultimately, implementing high-performance local search algorithms is a matter of expertise in *incremental algorithmics* and of dexterity in computer programming. Our concern to *industrialize* local search approaches shall be of a particular interest for practitioners. As examples, this methodology is applied to solve two industrial problems with high economic stakes.

Nevertheless, software applications based on local search induces extra costs in development and maintenance in comparison with the direct use of mixed-integer linear programming solvers. We present the *LocalSolver project* whose goal is to offer the power of local search through a model-and-run solver for *large-scale 0-1 nonlinear programming*. Having outlined its modeling formalism, the main ideas on which LocalSolver relies are described and some benchmarks are presented to assess its performance. We conclude the dissertation by presenting our ongoing and future works on LocalSolver *toward a full mathematical programming solver based on neighborhood search*.

Title: Toward a mathematical programming solver based on local search.
Keywords: combinatorial optimization, local/neighborhood search, mathematical programming.

Résumé

Ce mémoire traite de la *recherche locale en optimisation combinatoire* et de son extension à l'optimisation en variables mixtes. Notre but est de présenter la recherche locale sous un jour nouveau. Bien qu'encore mal cernée d'un point de vue théorique, la recherche locale est un paradigme de choix pour attaquer les problèmes d'optimisation de grande taille tels que rencontrés en pratique. Les utilisateurs demandent toujours plus d'interactivité avec les systèmes d'aide à la décision. Pour les logiciels d'optimisation, cela signifie obtenir de bonnes solutions vite. Les méthodes d'amélioration itérative véloces, comme la recherche locale, sont adaptées à ces besoins.

Lorsqu'un espace de solutions est gigantesque, une recherche complète devient impraticable. Étant donnée une solution (possiblement infaisable) au problème, une recherche locale consiste à modifier des parties de celle-ci – c'est-à-dire des variables de décisions – pour atteindre une nouvelle solution que l'on espère meilleure. Explorer un *voisinage* de la solution courante a un avantage majeur: la nouvelle solution peut être évaluée rapidement par *calcul incrémental*. Ainsi, la recherche locale peut être vue comme une façon incomplète et non déterministe mais efficace d'explorer un espace de solutions.

Tout d'abord, une *méthodologie* iconoclaste est présentée pour la conception et l'ingénierie d'algorithmes de recherche locale. Nous montrons que la performance d'une recherche locale repose essentiellement sur la richesse des voisinages explorés, ainsi que sur l'efficacité de leur exploration. En définitive, l'implémentation d'algorithmes de recherche locale performants est une question d'expertise en algorithmique incrémentale et de dextérité en programmation informatique. Notre souci d'*industrialiser* la recherche locale intéressera particulièrement les praticiens. À titre d'exemples, cette méthodologie est appliquée à la résolution de deux problèmes industriels aux enjeux économiques importants.

Néanmoins, les logiciels basés sur la recherche locale induisent des coûts additionnels de développement et de maintenance en comparaison avec l'utilisation directe de solveurs de programmation linéaire en nombres entiers. Nous présentons le *projet LocalSolver* dont le but est d'offrir la puissance de la recherche locale à travers un solveur *model-and-run* pour la *programmation non linéaire 0-1 à grande échelle*. Après une brève revue de son formalisme de modélisation, les idées maîtresses sur lesquelles reposent LocalSolver sont décrites et un échantillon de ses résultats est donné. Nous concluons le mémoire par une présentation de nos travaux en cours et futurs sur LocalSolver *vers un solveur de programmation mathématique complet basé sur une recherche par voisinage*.

Titre: Vers un solveur de programmation mathématique basé sur la recherche locale.
Mots-clés: optimisation combinatoire, recherche locale/par voisinage, programmation mathématique.