Studies in Computational Intelligence 434

For further volumes:
http://www.springer.com/series/7092

El-Ghazali Talbi (Ed.)

Hybrid Metaheuristics

Springer

Editor
Prof. Dr. El-Ghazali Talbi
University of Lille 1
CNRS, INRIA
France

ISSN 1860-949X
e-ISSN 1860-9503
ISBN 978-3-642-30670-9
e-ISBN 978-3-642-30671-6
DOI 10.1007/978-3-642-30671-6
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2012938580

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To my dear wife Keltoum. She always comforts and consoles me. I am such a lucky guy to have found such a wonderful woman.

To my two star sons Anis and Chahine. Every time that I saw your smile it lights me up inside.

To my daughter Besma, you are my sweet girl with an impressive intelligence.

To my mother Zehour for her infinite sacrifice.

To my father Ammar who continue to support me in my academic research.

Preface

Importance of This Book

Applications of optimization is countless. Every process has a potential to be optimized. There is no company which is not involved in solving optimization problems. Indeed, many challenging applications in science and industry can be formulated as optimization problems. Optimization occurs in the minimization of time, cost, risk, or the maximization of profit, quality, efficiency. For instance, there are many possible ways to design a network to optimize the cost and the quality of service; there are many ways to schedule a production to optimize the time; there are many ways to predict a 3D structure of a protein to optimize the potential energy, and so on.

A large number of real-life optimization problems in science, engineering, economics and business are complex and difficult to solve. They cannot be solved in an exact manner within a reasonable amount of time. Using hybrid algorithms is the main alternative to solve this class of problems.

Purpose of This Book

The main goal of this book is to provide a state of the art of hybrid metaheuristics. The book provides a complete background that enables readers to design and implement hybrid metaheuristics to solve complex optimization problems in a diverse range of application domains. Readers learn to solve large scale problems quickly and efficiently. Numerous real-world examples of problems and solutions demonstrate how hybrid metaheuristics are applied in such fields as telecommunication, logistics and transportation, bioinformatics, engineering design, scheduling, etc.

Audience

One of the main audience of this book is **advanced undergraduate and graduate students** in computer science, operations research, applied mathematics, control, business and management, engineering, etc. Many undergraduate courses on optimization throughout the world would be interested in the contents thanks to the introductory part of the book.

In addition, the **postgraduate** courses related to optimization and complex problem solving will be a direct target of the book. Hybrid metaheuristics are present in more and more postgraduate studies (computer science, business and management, mathematical programming, engineering, control, etc).

The intended audience is also **researchers** in different disciplines. Researchers in computer science and operations research are developing new optimization algorithms. Many researchers in different application domains are also concerned by the use of hybrid metaheuristics to solve their problems.

Many **engineers** are also dealing with optimization in their problem solving. The purpose of the book is to help engineers to use hybrid metaheuristics for solving real-world optimization problems in various domains of application. The application part of the book will deal with many important and strategic domains such as computational biology, telecommunication, engineering design, data mining and machine learning, transportation and logistics, production systems, etc.

Outline

The book is organized following 17 different chapters organized in 5 parts :

- Unified view of hybrid metaheuristics for mono and multi-objective optimization, and optimization under uncertainty.
- Combining metaheuristics with (complementary) metaheuristics.
- Combining metaheuristics with exact methods from mathematical programming approaches which are mostly used in operations research.
- Combining metaheuristics with constraint programming approaches developed in the artificial intelligence community.
- Combining metaheuristics with machine learning and data mining techniques.

Lille,
March 2012

Prof. Dr. El-Ghazali Talbi
University of Lille 1, CNRS, INRIA, France

Acknowledgements

Thanks to all contributors of this book for their cooperation in bringing this book to completion.

Thanks also go to all members of my research team DOLPHIN : D. Brockoff, L. Brotcorne, F. Clautiaux, B. Derbel, C. Dhaenens, L. Jourdan, A. Liefooghe, N. Melab, S. Verel.

Finally I should like to thank the team at Springer who gave me excellent support throughout this project, and especially for their patience.

Contents

List of Contributors

Enrique Alba
Universidad de Málaga, Spain
e-mail: eat@lcc.uma.es

Filipe Alvelos
Centro Algoritmi - DPS, Universidade do Minho, 4710-057 Braga, Portugal
e-mail: falvelos@dps.uminho.pt

Sarab Al-Muhaideb
Department of Computer Science, College of Computer and Information Sciences, King Saud University, P.O. Box 51178, Riyadh 11543, Saudi Arabia
e-mail: salmuhaideb@acm.org ·

Una Benlic
LERIA, University of Angers, 2 Bd Lavoisier, 49045 Angers Cedex 01, France
e-mail: benlic@info.univ-angers.fr

Christian Blum
ALBCOM Research Group, Universitat Polit‘ecnica de Catalunya, Barcelona, Spain
e-mail: cblum@lsi.upc.edu

Raffaele Cipriano
Dipartimento di Matematica e Informatica, Università degli Studi di Udine via delle Scienze 208, I-33100, Udine, Italy
e-mail: cipriano@dimi.uniud.it

Pedro J. Copado-Méndez
Departament d'Enginyeria Quimica, Universitat Rovira Virgili, Tarragona, Spain
e-mail: pedrojesus.copado@urv.cat

Frederico R. B. Cruz
Departamento de Estatística - ICEx - UFMG, Av. Antônio Carlos, 6627, 31270-901 - Belo Hori-zonte - MG, Brazil
e-mail: fcruz@est.ufmg.br

Patrick De Causmaecker
Katholieke Universiteit Leuven Campus Kortrijk Etienne Sabbelaan 53 8500 Kortrijk, Belgium
e-mail: Patrick.DeCausmaecker@kuleuven-kortrijk.be

Amaro de Sousa
Instituto de Telecomunicações / DETI, Universidade de Aveiro, 3810-193 Aveiro, Portugal
e-mail: asou@ua.pt

Luca Di Gaspero
Dipartimento di Ingegneria Elettrica, Gestionale e Meccanica, Università degli Studi di Udine via delle Scienze 208, I-33100, Udine, Italy
e-mail: l.digaspero@uniud.it

Julián Domínguez
Universidad de Málaga, Spain
e-mail: julian@lcc.uma.es

Agostino Dovier
Dipartimento di Matematica e Informatica, Università degli Studi di Udine via delle Scienze 208, I-33100, Udine, Italy
e-mail: dovier@dimi.uniud.it

Jérémie Dubois-Lacoste
IRIDIA, CoDE, Université Libre de Bruxelles (ULB), Brussels, Belgium
e-mail: jeremie.dubois-lacoste@ulb.ac.be

Christophe Duhamel
Institut Charles Delaunay (LOSI) and STMR (UMR CNRS 6279), Université de Technologie de Troyes, BP 2060, 10010 Troyes Cedex, France
e-mail: christophe.duhamel@isima.fr

Paola Festa
Department of Mathematics and Applications, University of Napoli Federico II, 80126 Napoli, Italy
e-mail: paola.festa@unina.it

Christophe Gouinaud
Laboratoire d'Informatique (LIMOS, UMR CNRS 6158), Campus des Cézeaux, 63177 Aubière Cedex, France
e-mail: christophe.gouinaud@isima.fr

G. Guillén-Gosálbez
Departament d'Enginyeria Quimica, Universitat Rovira Virgili, Tarragona, Spain
e-mail: gonzalo.guillen@urv.cat

Jin-Kao Hao
LERIA, University of Angers, 2 Bd Lavoisier, 49045 Angers Cedex 01, France
e-mail: hao@info.univ-angers.fr

Nur Insani
School of Mathematical Sciences, Monash University, Victoria 3800, Australia
e-mail: nur.insani@monash.edu

Laureano Jiménez
Departament d'Enginyeria Quimica, Universitat Rovira Virgili, Tarragona, Spain
e-mail: laureano.jimenez@urv.cat

Graham Kendall
School of Computer Science, University of Nottingham, Malaysia Campus, Jalan Broga, 43500, Semenyih, Selangor Darul Ehsan, Malaysia
e-mail: graham.kendall@nottingham.edu.my

Philippe Lacomme
Laboratoire d'Informatique (LIMOS, UMR CNRS 6158), Campus des Cézeaux, 63177 Aubière Cedex, France
e-mail: placomme@isima.fr

Leo Liberti
LIX, Ecole Polytechnique, Palaiseau, France
e-mail: liberti@lix.polytechnique.fr

Andrea Lodi
DEIS, University of Bologna and IBM-UniBo Mathematical Optimization Center of Excellence, Viale Risorgimento 2 – I-40136, Bologna, Italy
e-mail: andrea.lodi@unibo.it

Leo Lopes
School of Mathematical Sciences, Monash University, Victoria 3800, Australia
e-mail: leo.lopes@monash.edu

Manuel López-Ibáñez
IRIDIA, CoDE, Université Libre de Bruxelles (ULB), Brussels, Belgium
e-mail: manuel.lopez-ibanez@ulb.ac.be

Barry McCollum
School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, BT7 1NN, UK
e-mail: B.McCollum@qub.ac.uk

Paul McMullan
School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, BT7 1NN, UK
e-mail: p.p.mcmullan@qub.ac.uk

Mohamed El-Bachir Menai
Department of Computer Science, College of Computer and Information Sciences, King Saud University, P.O. Box 51178, Riyadh 11543, Saudi Arabia
e-mail: menai@ksu.edu.sa

Antonio Mucherino
IRISA, University of Rennes, Rennes, France
e-mail: antonio.mucherino@irisa.fr

Caroline Prodhon
Institut Charles Delaunay (LOSI) and STMR (UMR CNRS 6279), Université de Technologie de Troyes, BP 2060, 10010 Troyes Cedex, France
e-mail: caroline.prodhon@utt.fr

Jakob Puchinger
Mobility Department, Austrian Institute of Technology, Austria
e-mail: jakob.puchinger@ait.ac

Mauricio G.C. Resende
Algorithms and Optimization Research Department, AT&T Labs Research, Florham Park, NJ 07932 USA
e-mail: mgcr@research.att.com

Ulrike Ritzinger
Mobility Department, Austrian Institute of Technology, Austria
e-mail: ulrike.ritzinger_fl@ait.ac.at

Dorabella Santos
Instituto de Telecomunicações, 3810-193 Aveiro, Portugal
e-mail: dorabella@av.it.pt

Kate Smith-Miles
School of Mathematical Sciences, Monash University, Victoria 3800, Australia
e-mail: kate.smith-miles@monash.edu

Thomas Stützle
IRIDIA, CoDE, Université Libre de Bruxelles (ULB), Brussels, Belgium
e-mail: stuetzle@ulb.ac.be

El-Ghazali Talbi
University of Lille 1, CNRS, INRIA, Lille-France
e-mail: talbi@lifl.fr

Greet Vanden Berghe
CODeS (Combinatorial Optimisation and Decision Support) Industrial Sciences KAHO St.-Lieven, Belgium
e-mail: Greet.VandenBerghe@kahosl.be

Katja Verbeeck
Vakgroep ICT, University College, Katholieke Hogeschool Sint-Lieven, Ghent (KAHO Belgium)
e-mail: Katja.Verbeeck@kahosl.be

A. Villagra
Emerging Technologies Laboratory, Universidad Nacional de la Patagonia Austral, Argentine
e-mail: avillagra@uaco.unpa.edu.ar

Tony Wauters
CODeS, KAHO Sint-Lieven, Gebroeders Desmetstraat 1, 9000 Gent, Belgium
e-mail: tony.wauters@kahosl.be

Lyndon While
School of Computer Science and Software Engineering, The University of Western Australia, Perth WA 6009, Australia
e-mail: lyndon@csse.uwa.edu.au

Brendan Wreford
School of Mathematical Sciences, Monash University, Victoria 3800, Australia
e-mail: brendan.wreford@monash.edu

Chapter 11
Combining Column Generation and Metaheuristics

Filipe Alvelos, Amaro de Sousa, and Dorabella Santos

Abstract. In this Chapter, we consider the hybridization of column generation (CG) with metaheuristics (MHs) for solving integer programming and combinatorial optimization problems. We describe a general framework entitled "metaheuristic search by column generation" (for short, SearchCol). CG is a decomposition approach in which one linear programming master problem interacts with subproblems to obtain an optimal solution to a relaxed version of a problem. The subproblems may be solved by problem-specific algorithms. After CG is applied, a set of subproblem's solutions, optimal primal and dual values of the master problem variables and a lower bound to the optimal value of the problem are available. In contrast with enumerative approaches (e.g, branch-and-price), in SearchCol the information provided by CG is used in a MH search. The search is based on representing a solution (to the overall problem) as being composed by one solution from each subproblem. After a search is conducted, a perturbation for CG is defined and a new iteration begins. The perturbation consists in forcing or forbidding attributes of the subproblem's solutions and, in general, leads to the generation of new subproblem's solutions and different optimal primal and dual values of the master problem variables. In this Chapter, we discuss (i) which models are suitable for decomposition approaches as SearchCol, (ii) different alternatives for generating initial solutions for the search (with different degrees of randomization, greediness and influence of CG) (iii) different search approaches based on local search, (iv) different alternatives for perturbing CG (influenced by CG, based on the incumbent, and based on the memory of the search).

Filipe Alvelos
Centro Algoritmi / DPS, Universidade do Minho, 4710-057 Braga, Portugal
e-mail: falvelos@dps.uminho.pt

Amaro de Sousa
Instituto de Telecomunicações / DETI, Universidade de Aveiro, 3810-193 Aveiro, Portugal
e-mail: asou@ua.pt

Dorabella Santos
Instituto de Telecomunicações, 3810-193 Aveiro, Portugal
e-mail: dorabella@av.it.pt

E.-G. Talbi (Ed.): Hybrid Metaheuristics, SCI 434, pp. 285–334.
springerlink.com

11.1 Introduction

We describe a framework for obtaining approximate solutions for mixed integer programming (MIP) and combinatorial optimization (CO) problems. The framework, named metaheuristic search by column generation , or *SearchCol* for short, relies on the combination of column generation (CG) and metaheuristic (MH) search. The aim of a SearchCol algorithm is to obtain high quality solutions in reasonable amounts of time for a wide range of problems.

The core ideas of this work were first proposed in [2] and have their roots in [4]. In this Chapter, we further develop some concepts, provide additional details on others, and extend the SearchCol framework to a more general setting.

SearchCol is a decomposition approach in which a solution to a problem is composed by several components, which are themselves solutions to smaller problems. Its central idea is the exchange of information between CG and a MH. Basically, CG provides subproblem solutions which define a search space for the MH. The MH provides an incumbent solution and information on desired or avoidable attributes of the subproblem solutions to CG. This exchange of information is repeated until a stopping criterion is met.

A particular MH is not specified in SearchCol but a set of algorithmic components are defined such that different (hybrid) MHs can be used in an actual SearchCol algorithm. Those components use CG information, randomness, greediness, neighborhoods, and memory for implementing different strategies of building and improving solutions.

SearchCol can be applied in the vast majority of problems for which approaches based on Dantzig-Wolfe decomposition [19] or Lagrangean relaxation [30] were devised (even when the model to be solved does not result directly from those reformulation methods). Dantzig-Wolfe decomposition and Lagrangean relaxation are dual equivalent of each other and the corresponding solution approaches, CG and Kelley's cutting plane method [44], are also dual equivalent of each other (see [28] for a clear explanation). Having present this equivalence, in the remainder of this Chapter, we use the primal perspective of CG and Dantzig-Wolfe decomposition.

In Dantzig-Wolfe decomposition , it is assumed that an original model exists. By defining a set of subproblems with some of the constraints of the original MIP model, a reformulated model, the decomposition model, is obtained. In many applications, a decomposition model can be formulated directly. In this Chapter, although we mention Dantzig-Wolfe decomposition when we think that this important particular case is worth to detail, we consider the more general decomposition model.

11.1.1 Motivations

Decomposition approaches are appealing for MIP and CO problems for several reasons:

- Decomposition models may capture the decomposable structure already present in the problem (which in fact, seems to exist in most pratical problems [49]) being the simpler and more direct way to model it.
- Decomposition is attractive for large-scale problems when a non-decomposition model is too large to be dealt with computationally.
- Decomposition approaches can be parallelized easier than direct approaches.
- A problem may involve complex constraints or objective functions which are not easily addressed by MIP (where linearity is assumed - disregarding the integrality constraints of the variables) but can be efficiently addressed by other modelling approaches such as dynamic programming [22] and constraint programming [35]. These approaches can be used in subproblems, but still framed by MIP / linear programming.
- Decomposition approaches can be faster than the other available approaches. There are two main reasons for this:
  - The Linear relaxation of a decomposition model may provide a better gap than other models. In particular, when a Dantzig-Wolfe decomposition is applied, if the subproblem does not possess the integrality property (i.e, not all its extreme points are integers), the lower bound (assuming a minimization problem, as will be done throughout the Chapter) provided by the linear relaxation of the decomposition model is greater than or equal to the one provided by the linear relaxation of the original model. Good quality lower bounds are a major factor of the quality of implicit enumeration algorithms, e.g. branch-and-bound.
  - Very efficient problem-specific algorithms can be used in solving subproblems.

Decomposition approaches are usually applied in NP-hard problems. A MIP decomposition model (which linear relaxation can be solved by CG), has an exponential number of variables. Even when the set of variables is restricted, the restricted problem usually remains NP-hard (it is a general MIP). Therefore, the definition of a combinatorial model to explore a search space in an approximate manner is an attractive solution approach.

When using SearchCol, there is an implicit combinatorial model which is the base for MH search. In this model a solution is represented by a set of subproblem solutions, one from each subproblem. The rationale for using this combinatorial *decomposition* model relies on the following three aspects:

- The decomposition naturally defines a search space in a higher level which is easier to explore. In the combinatorial decomposition model, subproblems solutions are combined to form a solution to the overall problem. These subproblems solutions can be seen as higher level components when compared to subproblem variables. Note that the feasibility in the subproblems constraints is assured in subproblem solutions.
- If the linear relaxation of the MIP decomposition model associated with the combinatorial decomposition model provides high quality lower bounds, it is

reasonable to expect that the search space is of better quality (i.e. poor solutions are excluded) than search spaces based on other models.
- The decomposition of the structure of a solution naturally leads to the definition of operators for MH search.

11.1.2 Literature Review

SearchCol combines decomposition approaches mainly based on mathematical programming and MHs. We now give a necessarily brief overview of relevant works on the first type, on the second type, and also on the combination of both types of approaches.

For providing approximate solutions (as in the so called Lagrangean heuristics) or optimal solutions (as in branch-and-price), decomposition approaches based on Dantzig-Wolfe decomposition [19] or Lagrangean relaxation [30] for MIP and CO have been a major topic of research in the last decades [41]. For example, two of the most influential ten papers in one of the most relevant journals of the Institute for Operations Research and the Management Sciences [39] are papers in decomposition approaches: CG [27] and Lagrangean relaxation [24].

The roots of Lagrangean relaxation in MIP and CO lie in [37, 38, 30, 24]. More recent references can be found in [10, 25]. The roots of Dantzig-Wolfe decomposition and CG lie in [27, 19, 31, 32, 21]. A modern perspective on CG for integer programming is given in the surveys [9, 65, 46, 64] and in the book [20].

MHs are the most successfull approaches for addressing large instances of many CO problems. There is a vast literature since the 1980s where significant research on this area begun (recent books are [33, 63, 29]). A comprehensive survey paper is [14]. A landmark book is [34].

In recent years, hybrid MHs (taken as the combination of different MHs or taken as the combination of MHs with other approaches) emerged as an important area of research. In [62], a taxonomy of hybrid MHs is proposed and an annotated bibliography classifying several MHs according to the taxonomy is provided. Recent surveys on hybrid MHs are [55, 13]. Surveys and applications of hybrid MHs are in the book [11].

Among the hybrid MHs, the combination of heuristics and mathematical programming , resulting in the so-called *Matheuristics*, has received an increasing interest (see, for example, [56, 12, 47, 7] and sections 4 and 5 of [13]). SearchCol belongs to this research stream.

SearchCol is a framework which combines a decomposition method (CG) and MH search. The next two references also propose frameworks for this combination.

In [18], a framework for combining branch-and-price with local search is proposed. In that work, local search is applied in the nodes of the branch-and-price tree for attempting the improvement of the incumbent solution and generating new columns. The resulting hybrid approach is applied to a vehicle routing problem where the master problem is a set partitioning model. A central difference between

this framework and Searchcol is that the latter intentionally avoids an implicit enumeration strategy, i.e. (heuristic) branch-and-price.

A framework based on Dantzig-Wolfe decomposition is proposed in [15] in which the primal and dual solutions are used to construct a problem solution in a problem-specific manner. In the same reference a framework for Lagrangean heuristics is also provided. The central step (building a solution from the subproblem solutions and from the dual variables values) is also problem dependent. In SearchCol, the subproblems are also solved in different dual points (as in the frameworks of the reference) given by the restricted master problems (RMPs) of CG (note that a subgradient method could also be used), but then the solution is obtained in a problem independent manner. Furthermore, this solution can be used to perturb CG leading to the consideration of new subproblem solutions. This perturbation is also problem independent.

Other examples and references on the combination of CG based approaches and heuristics are provided in Section 11.4 after the description of the MIP decomposition model used in SearchCol.

We now classify SearchCol according to the classification of [54]. The kind of algorithms that are hybridized define a first differentiation criterion in that classification scheme. In SearchCol, at least two types of algorithms are combined: a linear programming algorithm for solving the RMP of CG and a (hybrid) MH for the search. A problem-specific algorithm for solving the subproblem is also usual. The second differentiation criterion is the strength of combination. In SearchCol, CG influences a MH and the reverse is also true. As the algorithms retain their own identities, there is a low level of hybridization. Note that this weak coupling is a consequence of the generality of the approach. The order of execution is interleaved and the combination is collaborative: the CG and the MH exchange information but none is strongly subordinated to the other.

11.1.3 Contributions

Decomposition models are powerful modelling tools, but efficient solution methods are required for their success in problem solving. The main contribution of this work is the proposal of a framework, SearchCol, which has the purpose of solving efficiently a wide range of decomposition models. The key issue is that SearchCol explores the combinatorial structure of a problem which has been mostly tackled by mathematical programming. Furthermore, the combinatorial perspective is rather natural as we are given a number of sets, each one with a large number of elements, and we are interested in selecting one element from each such that a function is minimized.

There are two main strenghts in this framework. The first one is its ability to deal with complex problems, since the subproblems of CG can be approached with any optimization technique and their complexity is hidden from the search which

is conducted in a higher level. The second one is its generality, since the problem-specific components of a SearchCol algorithm are limited to a subproblem solver and its interaction with a linear programming model. In fact, if the model results from a Dantzig-Wolfe decomposition and the subproblems are solved with a general purpose solver, SearchCol is totally problem-independent.

The type of models which may be addressed by SearchCol are the ones where the linear relaxation may be solved by CG. In this sense, SearchCol is an alternative to branch-and-price, not based on enumeration. Rigorously, we consider the case where the coefficients of a column in the rows of the master problem are a function of binary variables of the subproblem (the subproblem may have additional binary/integer/continuous variables but they are not used for this purpose). This is the most usual case in CG based approaches.

Besides the usual variables associated with the subproblem solutions, our framework also admits binary / general integer / continuous variables in the master problem, which provides flexibility when modelling a problem through decomposition.

Two related contributions of this work is the combinatorial model used in SearchCol and the proposal of a large number of algorithmic components for the definition of MHs to address it. These components can be combined originating particular MHs. We provide the examples of (advanced) multi-start local search and variable neighborhood search. Additional algorithmic components, which are easily conceivable, allow the extension to other approaches, in particular, population-based MHs.

11.1.4 Chapter Structure

This Chapter is organized as follows. In Section 2 we introduce the decomposition MIP model and give examples for different problems. In Section 3 we describe CG and perturbed CG which are base ingredients of SearchCol. CG is used to solve the linear relaxation of the MIP model. In Section 4 we review solution methods for the decomposition MIP model. In Section 5 we propose a different perspective on the problem being solved which is based on a combinatorial decomposition model. We discuss the solution representation, the search space, and how solutions are evaluated. We provide a description of Searchcol in Section 6. In Section 7, the MH search phase is detailed and examples of actual SearchCol algorithms are provided. In Section 8 we discuss the main conclusions of the Chapter.

11.2 A Decomposition Mixed Integer Programming Model

We consider problems where a set of decisions, each one from a finite but large set of alternatives, must be taken. The alternatives are modelled through binary variables and a cost value might be associated with each one. In the most general case, the relation between the decisions can be modelled by binary, general integer and

continuous variables, linear constraints, and may imply additional costs. We intend to minimize the total cost.

In subsection 11.2.1 a general decomposition model for these problems is introduced. This model relies on associating a subproblem solution with each decision. In the same subsection, we define the notation used and state the assumptions made. The decomposition model is introduced independently of any solution method or previous model, what we believe can be interesting as a starting point for a fresh look to decomposition approaches.

In subsection 11.2.2 we provide several examples of decomposition models.

11.2.1 Model Statement

We consider the following MIP model (D), named decomposition model, which may be derived directly for a problem or may be the result of the application of a Dantzig-Wolfe decomposition or of a Lagrangean relaxation.

$$Min \sum_{t\in T} c_t y_t + \sum_{k\in K}\sum_{s\in S^k} c_s^k y_s^k \qquad (D)$$

subject to

$$\sum_{s\in S^k} y_s^k = 1 \qquad k \in K \qquad (11.1)$$

$$\sum_{t\in T} a_{it} y_t + \sum_{k\in K}\sum_{s\in S^k} a_{is}^k y_s^k \{\leq, =, \geq\} b_i \qquad i \in I \qquad (11.2)$$

$$y_t \in \{0,1\} \qquad t \in T^b \qquad (11.3)$$

$$y_t \geq 0 \text{ and integer} \qquad t \in T^i \qquad (11.4)$$

$$y_t \geq 0 \qquad t \in T^c$$

$$y_s^k \in \{0,1\} \qquad k \in K, s \in S^k \qquad (11.5)$$

The decomposition model (D) has two sets of decision variables. Variables y_t, which we name *static* variables, are not present in the first set of constraints and are grouped by type: binary ($t \in T^b$), general integer ($t \in T^i$), and continuous ($t \in T^c$). The second set of decision variables, y_s^k, $k \in K$, $s \in S^k$, are the *selection* variables and are partitioned in $|K|$ subsets, indexed by k. Their general definition is the following.

$$y_s^k = \begin{cases} 1 \text{ if the } s\text{-th element of the subset } k \text{ is selected} \\ 0 \text{ otherwise} \end{cases} \quad k \in K,\ s \in S^k$$

Parameters c_t, c_s^k, a_{it}, a_{is}^k, and b_i are coefficients of the variables in the objective function, coefficients of the variables in some of the constraints and the right-hand sides of some of the constraints, as can be seen directly in (D).

Constraints (11.1) state that one element from each subset k must be selected, therefore they are named *selection* constraints. Constraints (11.2) may include any decision variable. We name them *global* constraints.

We are interested in the case where the number of selection variables is so large that addressing model (D) directly, for reasonable size instances, is out of the question. The growth of $|S^k|$ with respect to the size of the data defining the problem to be solved is *not* bounded by a polynomial. On the contrary, the number of static variables and constraints is polynomial.

Furthermore, we consider that each partition k is associated with a subproblem, which allows treating the selection variables implicitly as they become associated with solutions of the subproblem. The subproblem is defined such that each subproblem solution, x_s^k, is associated with one selection variable, y_s^k. Each subproblem k has a set J^k of binary decision variables represented by the vector x^k with components x_j^k. We represent the s-th feasible solution by x_s^k. The subproblem may have auxiliary variables, w_r^k not necessarily binary. Representing the set of indexes of the auxiliary binary variables of subproblem k by R^{bk}, the set of indexes of general integer variables by R^{ik}, and the set of indexes of continuous variables by R^{ck}, the feasible region of the subproblem, which we represent by Q^k, is defined by

$$
\begin{aligned}
&(x^k, w^k) \in W^k && (Q^k)\\
&x^k \in \{0,1\}^{|J^k|}\\
&w_r^k \in \{0,1\}, r \in R^{bk}\\
&w_r^k \geq 0 \text{ and integer}, r \in R^{ik}\\
&w_r^k \geq 0, r \in R^{ck}
\end{aligned}
$$

where W^k is the set of possible values for the decision variables x^k and w^k.

We represent the determination of the coefficients of the selection variables in (D) for each subproblem solution x_s^k, $k \in K$, $s \in S^k$, by the following functions:

$$
\begin{aligned}
&c_s^k = f_0^k(x_s^k, w^k)\\
&a_{is}^k = f_i^k(x_s^k) && i \in I
\end{aligned}
$$

Note that the variables w_r^k do not appear in the f_i^k functions which implies that they do not participate in the definition of the feasible region of (D). If two solutions have the same value in the x^k variables but different values in the w^k variables, then a single selection variable is associated with solution x^k, the one with a lower value (given by f_0^k).

A solution in terms of the selection variables can be translated into a subproblem solution. In order to establish this relation we define x_{js}^k as the value of the j-th variable of the subproblem k in solution s (which takes values 0 or 1).

Given a set of values for the selection variables y_s^k, the value of a subproblem variable is given by:

$$x_j^k = \sum_{s \in S^k} x_{js}^k y_s^k. \tag{11.6}$$

There are other types of models closely related to model (D) and for which all the approaches discussed and proposed in this Chapter can also be applied.

In the first type, there is only one subproblem and we want m different solutions from it. The selection constraints (11.1) are replaced by a single constraint

$$\sum_{s \in S} y_s = m$$

The second type of models are set covering, partitioning, and packing. In these models a parameter m representing an upper bound to the number of sets in an optimal solution is introduced (note that a trivial upper bound is the number of elements). The selection constraints (11.1) are again replaced by a single constraint

$$\sum_{s \in S} y_s \leq m$$

In the most general case, the selection constraints take the form:

$$y_0^k + \sum_{s \in S^k} y_s^k = m^k, k \in K$$

where m^k is the number of different solutions required for subproblem k, which may include null solutions as modelled by the slack (general integer) variables y_0^k. If the null solution of the subproblem is not feasible then $y_0^k = 0$.

In general, different decomposition models can be formulated for the same problem. As virtually all the solution approaches for solving (D) are based on linear relaxations, a first criterion for the choice of a particular decomposition model instead of another is the quality of the lower bound given by its linear relaxation. A second criterion is the easiness of dealing with the subproblem. The third criterion is the difficulty to solve the decomposition model itself: in principle, the less the number of global constraints, the better.

In the next subsection, we give examples of decomposition models for different types of problems.

11.2.2 Examples

11.2.2.1 Time Constrained Shortest Path Problem

We first consider the time constrained shortest path problem. In this problem, a directed network is given. Associated with each arc there are two parameters: its length and the time it takes to be traversed. The objective is to find the shortest path between two nodes not exceeding a given time limit. We define c_j as the length of arc j, t_j as the travel time in arc j, and T as the time limit. In order to obtain

a decomposition model in the form of the general one introduced in the previous subsection, we define the following decision variables.

$$y_s = \begin{cases} 1 \text{ if path } s \text{ is the chosen path} \\ 0 \text{ otherwise} \end{cases}, s \in S$$

where S is the set of all paths. A decomposition model is:

$$Min \sum_{s \in S} c_s y_s$$
subject to
$$\sum_{s \in S} y_s = 1,$$
$$\sum_{s \in S} a_s y_s \leq T, j \in A$$
$$y_s \in \{0,1\}, s \in S$$

Note that in this model, there are no static variables and we only define one partition (we do not use the index k).

As the number of paths in a network grows exponentially with the dimension of the network, we define a subproblem where a solution corresponds to a path and the decision variables are associated with arcs (J is the set of arcs of the network). There are no auxiliary subproblem variables. The subproblem solution of path s is given by

$$x_{js} = \begin{cases} 1 \text{ if arc } j \text{ belongs to path } s \\ 0 \text{ otherwise} \end{cases} \quad k \in K, j \in J$$

The coefficients of the selection variables in the objective function and in the constraint are given by

$$c_s = f_0^k(x_s^k) = \sum_{j \in J} c_j x_{js}$$
$$a_s = f_1^k(x_s^k) = \sum_{j \in J} t_j x_{js}$$

In this model, the relation between the values of the selection variables and the subproblem variables established by (11.6) let us obtain the flows on the arcs (the values of the subproblem variables) based on the flows on paths (the values of the selection variables). This is a well known result for network flow problems [1].

We give a numerical example with an instance from [1] in Figure 11.1 where the values on each arc are the cost and time by that order. It is intended to obtain the shortest path between nodes 1 and 6 with the time limit of 14 time units.

For this very small instance, all paths can be enumerated easily. The set of all paths, S, their lenghs and times, are given in Table 11.1. Since the decomposition model is very small, it can be solved directly by a general purpose solver. The optimal solution corresponds to the path $1-3-2-4-6$ with length 13.

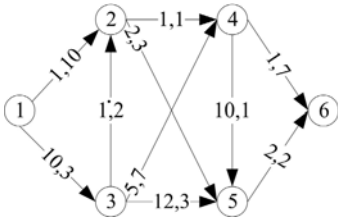

Fig. 11.1 An instance of the time constrained shortest path problem

Table 11.1 Parameters of the decomposition model for the time constrained shortest path example

s	(1,2,4,6)	(1,2,4,5,6)	(1,2,5,6)	(1,3,2,4,6)	(1,3,2,4,5,6)	(1,3,2,5,6)	(1,3,4,6)	(1,3,4,5,6)	(1,3,5,6)
c_s	3	14	5	13	24	15	16	27	24
a_s	18	14	15	13	9	10	17	13	8

11.2.2.2 Single Path Routing Problems

A single path routing problem is defined over a network in which a set of commodities, K, must be routed from their origin nodes to their destination nodes. We address the problem in which each commodity has a given demand, d^k, corresponds to an origin/destination pair, and there are capacities associated with the arcs, $b_{ij}, ij \in A$, where A is the set of arcs of the network. Furthermore, each commodity must use a unique path to route its demand. This problem is also known as the unsplittable multicommodity flow problem. In here, we consider two variants of this problem: in the first, there are costs (per unit of demand) associated with the arcs that may vary by commodity and the objective is to minimize the total cost (min cost problem); in the second, the objective is to minimize the load (i.e, the used proportion of the capacity of the arc) of the arc with the maximum load (min-max problem). We discuss the cases of directed networks but the models and solution approaches to be discussed later can be easily transformed for undirected ones. This problem have been addressed in [8, 3, 4].

Min cost problem

A decomposition model for the minimum cost single path routing problem is known as the arc-path formulation (see, for example, [1]). A selection variable, y_s^k, is associated with path s of commodity k, therefore it will take the value 1 if path s is the path selected for commodity k, and 0, otherwise. There are no static variables and there is one global constraint for each arc ij, $ij \in A$, stating that the capacity of the arc cannot be exceeded:

$$\sum_{k \in K} \sum_{s \in S^k} a^k_{(ij)s} y^k_s \leq b_{ij}$$

As in the time constrained shortest path example, the subproblem variables are associated with arcs. For commodity k:

$$x^k_{(ij)s} = \begin{cases} 1 \text{ if arc } ij \text{ belongs to path } s \text{ of commodity } k \\ 0 \text{ otherwise} \end{cases} \quad k \in K, ij \in A$$

Denoting the unitary cost for commodity k of arc ij by c^k_{ij}, then cost of the selection variable y^k_s is given by

$$c^k_s = f^k_0(x^k_{(ij)s}) = \sum_{ij \in A} c^k_{ij} x^k_{(ij)s}$$

The coefficients of the variables in the global constraints are given by:

$$a^k_{(ij)s} = f^k_{ij}(x^k_{(ij)s}) = d^k x^k_{(ij)s}$$

Min max problem

A decomposition model for the min max single path routing problem can be obtained by considering one *static* continuous variable, y_1, corresponding to the maximum load of an arc. The objective function becomes

$$Min\, y_1$$

and the global constraint for arc ij, $ij \in A$:

$$-b_{ij} y_1 + \sum_{k \in K} \sum_{s \in S^k} a^k_{(ij)s} y^k_s \leq 0$$

Note that, in this case, there are no costs associated with the subproblem solutions.

11.2.2.3 Vehicle Routing Problems

We consider the example of a decomposition model for a classical vehicle routing problem [22, 43]. We consider a network with nodes associated with customers and with the depot (one node is the origin depot and another is the destination depot). We consider that each customer has a given demand and a given time window (the demand of the client must be delivered not earlier than the beginning of the time window and not later than its end). Associated with each arc there is a cost and a time. We consider a fleet of homogeneous vehicles - all the vehicles have the same capacity. The problem is to define the route of each vehicle such that all customers are served within the time windows prescribed and minimizing the total distance travelled by the fleet. The decomposition model has no static variables, each selection variable is associated with a specific route (a path starting at the origin depot and ending at the destination depot), and the global constraints state that each customer must be visited:

$$\sum_{s \in S^k} a_{is}^k y_s^k = 1.$$

The subproblem variables are associated with arcs:

$$x_{(ij)s} = \begin{cases} 1 \text{ if arc } ij \text{ belongs to route } s \\ 0 \text{ otherwise} \end{cases} \quad ij \in A$$

Denoting the length of arc ij by c_{ij}^k, then the cost of the selection variable y_s is given by

$$c_s^k = f_0^k(x_{(ij)s}^k) = \sum_{ij \in A} c_{ij}^k x_{(ij)s}^k$$

The coefficients of the selection variables in the global constraints are given by:

$$a_{is}^k = f_i^k(x_{(ij)s}^k) = \sum_{ij \in A} x_{(ij)s}^k$$

A subproblem solution is a path from the origin depot to the destination depot which obeys the time windows of the customers associated with the nodes of the path. Auxiliary subproblem variables must be required to define feasible paths (paths obeying the time windows).

11.2.2.4 Machine Scheduling Problems

We consider the example of decomposition models for parallel machine scheduling where there are a given number of machines and a given number of jobs to be processed on them with some time-related objective. For each job, a processing time in each machine, a priority or weight, possibly a release date, and possibly a due date, are given. Machines can be identical or different, if they process the jobs with different speeds, or have different availabilities during the time horizon. Usual decomposition models [5, 16, 6, 45] have no static variables, each selection variable is related with a specific schedule, the selection constraints state that one schedule for each machine must be selected (non-identical machines) or that a given number of schedules must be selected (identical machines), and there is one global constraint for each job i stating that the job must be processed (in the case of identical machines, we drop the superscript k):

$$\sum_{k \in K} \sum_{s \in S^k} a_{is}^k y_s^k = 1.$$

As the number of schedules grows exponentially with the number of jobs, we define a subproblem for each machine where a solution corresponds to a schedule and the decision variables are associated with the assigment of jobs to the machine. The subproblem solution of schedule s of machine k is given by

$$x_{js}^k = \begin{cases} 1 \text{ if job } j \text{ belongs to schedule } s \text{ of machine } k \\ 0 \text{ otherwise} \end{cases} \quad k \in K,\ j \in J$$

The coefficients of the selection variables in the global constraint of job i are:

$$a_{is}^k = f_i^k(x_s^k) = x_{js}^k, \text{ with } i = j.$$

The determination of the objective coefficient of the selection variables involves a set of auxiliary subproblem variables which are related with the completion times of the jobs. For example, in [5, 16], the objective is to minimize the total weighted completion time and therefore, the cost of a schedule depends on the completion time of the jobs it includes. Representing the completion time of job j in schedule s of machine k by w_{rs}^k, then

$$c_s^k = f_0^k(x_s^k, w_r^k) = \sum_{j \in J} W_j w_{rs}^k x_{js}^k$$

where W_j is the weight of job j. The completion time of a job depends on which jobs are processed in the same machine. Therefore function f_0 is non linear.

11.2.2.5 Multiple Spanning Tree Routing Problems

Multiple spanning tree routing is a particular case of single path routing. This problem is defined in undirected networks and is motivated by the spanning tree routing protocol for switched Ethernet networks [40]. In multiple spanning tree routing, a set of spanning trees is defined and each commodity is to be routed over the edges of one of the spanning trees. Given the number of spanning trees that may be used, the problem is to define the specific spanning trees to be used and which commodities are assigned to each spanning tree. This problem and a related problem have been addressed in [58, 59].

The tree and path decomposition models are based on selection variables corresponding to paths and corresponding to spanning trees (other decompositions are studied in [58]). We describe the min cost problem, noting that a min max problem can be modelled in a similar way.

We divide partitions K in two subsets: $K1$ is related with paths and $K2$ is related with spanning trees. Therefore, a variable y_s^k with $k \in K1$ is associated with a path and a variable y_s^k with $k \in K2$ is associated with a spanning tree. Naturally, $K = K1 \cup K2$. For $k \in K1$, we define $x_{\{i,j\}s}^k$ equal to 1 if path s, which is associated with commodity, includes edge $\{i, j\}$ and equal to 0, otherwise. For $k \in K2$, we define $x_{\{i,j\}s}^k$ equal to 1 if spanning tree s, which is associated with k-th spanning tree, includes edge $\{i, j\}$ and equal to 0, otherwise.

As fixed variables, we define ϕ_{k1}^{k2} equal to 1 if path $k1 \in K1$ is assigned to spanning tree $k2 \in K2$.

A decomposition model based on paths and trees for the minimum cost multiple spanning tree routing problem is:

$$Min \sum_{k \in K1} \sum_{s \in S^k} c_s^k y_s^k \qquad (Dmctrp)$$

subject to

$$\sum_{s \in S^k} y_s^k = 1 \qquad k \in K \quad (11.7)$$

$$\sum_{s \in S^k 1} x_{\{i,j\}s}^{k1} y_s^{k1} \leq \sum_{s \in S^k 2} x_{\{i,j\}s}^{k2} y_s^{k2} + (1 - \phi_{k1}^{k2}) \quad \{i,j\} \in A, k1 \in K1, k2 \in K2 \quad (11.8)$$

$$\sum_{s \in S^{k2}} \phi_{k1}^{k2} = 1 \qquad k1 \in K1, k2 \in K2 \quad (11.9)$$

$$\sum_{k \in K1} \sum_{s \in S^k} a_k x_{\{i,j\}s}^k y_s^k \leq b_{\{i,j\}} \qquad \{i,j\} \in A \quad (11.10)$$

$$\phi_{k1}^{k2} \in \{0,1\} \qquad k1 \in K1, k2 \in K2$$

$$y_s^k \in \{0,1\} \qquad k \in K, s \in S^k$$

Constraints (11.7) force one path for each commodity and one spanning tree for each allowed spanning tree to be selected. Constraints (11.8) state that if a commodity uses edge $\{i,j\}$ and is assigned to the t-th spanning tree then this spanning tree must include edge $\{i,j\}$. Constraints (11.9) state that each commodity is assigned to one spanning tree. Constraints (11.10) are the capacity constraints. Note that in this problem the capacities are given by edge.

11.3 Column Generation

Column generation (CG) has been successfully applied in solving linear programming problems with a huge number of variables (many of those being relaxations of integer models). In this Section, we discuss how the linear relaxation of the decomposition model introduced in the previous section can be solved by CG.

In subection 11.3.1 we provide a general overview of CG. In subsection 11.3.2 we address the subproblem in detail. In subsections 11.3.3 and 11.3.4 we provide a CG algorithm and an example of its application. In subsection 11.3.5 we present how subproblem variables can be fixed, which is an important topic to the solution methods to be discussed in the remainder of this Chapter.

11.3.1 Overview

CG is a method that allows obtaining an optimal solution to the linear relaxation of the decomposition model (D). The linear relaxation of (D) (LRD) is obtained by replacing the integrality constraints, (11.3), (11.4), and (11.5) by:

$$0 \leq y_t \leq 1 \qquad t \in T^b$$
$$y_t \geq 0 \qquad t \in T^i$$
$$y_s^k \geq 0 \qquad k \in K, s \in S^k$$

Note that each y_s^k cannot exceed 1 because of the selection constraints (11.1). In many CG applications the (relaxed) static variables do not appear. However, their presence extends the ability to model relevant problems with decomposition models, as illustrated by some of the examples of subsection 11.2.2.

CG relies on the definition of restricted master problems (RMPs) which are linear programming problems where not all the variables of the overall problem are considered. In the case of LRD, in a RMP, not all the selection variables are considered. Consider the case where LRD has only one subproblem, which is easily extendable for $|K|$ subproblems as will be done later. A CG algorithm begins with the definition of a first RMP, which may include artificial variables for assuring feasibility. In each iteration, the current RMP is optimized and its optimal dual solution is used in evaluating the reduced costs of the selection variables outside the RMP. This is done by solving the subproblem where the objective function is related with the reduced costs of the selection variables. The smallest reduced cost is obtained and, according to linear programming theory, if it is non negative the RMP solution is optimal for the overall problem LRD and the algorithm stops. Otherwise, the selection variable associated with the subproblem solution (the one with the most negative reduced cost) is introduced in the RMP (by using functions f_0 and f_i, $i \in I$) and a new iteration begins.

11.3.2 Subproblem

11.3.2.1 General Considerations

In order to identify selection variables with negative reduced costs, a subproblem for each k, $k \in K$, is solved. The subproblem k resolution aims at identifying a selection variable y_s^k with negative reduced cost.

By definition, representing the dual variables of constraints (11.1) by π^k, $k \in K$, and the dual variables of constraints (11.2) by ω_i, $i \in I$, the reduced cost of variable y_s^k is given by

$$\bar{c}_s^k = c_s^k - \pi^k - \sum_{i \in I} a_{is}^k \omega_i.$$

The relation established between the selection variables and subproblem solutions (given by the functions f_0 and f_i, $i \in I$), allows to obtain the selection variable y_s^k with the most negative reduced cost by solving an optimization subproblem for each partition k:

$$Min - \pi^k + f_0^k(x^k, w^k) - \sum_{i \in I} \omega_i f_i^k(x^k)$$

subject to

$$(x^k, w^k) \in Q^k$$

CG is of practical interest if there is an efficient exact algorithm able to solve this (sub)problem. Typically, this algorithm is problem-specific as we are interested in exploring the common structure of the selection variables of the same partition. In many cases, such as the machine scheduling and vehicle routing models discussed in 11.2.2, those algorithms are based on dynamic programming which allows dealing with non linear objective functions and non linear constraints. Examples of subproblems solved by constraint programming can be found in [52, 23]. For the network routing examples given in the previous subsection, shortest path algorithms (e.g. Dijkstra's) and minimum spanning tree algorithms (e.g. Kruskal's or Primm's) can be used. Other usual subproblems are binary knapsacks (for example, [60]).

We now discuss two usual subproblem particular cases.

11.3.2.2 The Linear Case

In the linear case, the functions relating the subproblem solutions and the selection variables f_0^k and f_i^k, $i \in I$, $k \in K$, are linear. We define them, for $k \in K$, as

$$f_0^k(x^k, w^k) = \sum_{j \in J^k} d_{0j}^k x_j^k + \sum_{r \in R^k} d_r^k w_r^k$$

$$f_i^k(x^k) = \sum_{j \in J^k} d_{ij}^k x_j^k, i \in I$$

where d_{0j}^k, d_{ij}^k, $i \in I, j \in J^k$, $d_r, r \in R$ are scalars.

Using the example of the time constrained shortest path problem, d_{0j} corresponds to the cost of arc j and d_{1j} corresponds to the time of arc j (there is only one global constraint). For min cost single path routing, d_{0j}^k is the cost of arc j for commodity k and d_{ij}^k is the demand of commodity k if $i = j$ (the arc of the global constraint i is the arc of the subproblem j) and is 0 otherwise. X^k is set the of all paths between the origin and the destination of k.

Given the duals of the global constraints of the current RMP, π^k, $k \in K$, and ω_i, $i \in I$, the subproblem k is:

$$\bar{c}^k = min \sum_{j \in J^k} d_{0j}^k x_j^k - \pi^k - \sum_{i \in I} \omega_i \sum_{j \in J^k} d_{ij}^k x_j^k + \sum_{r \in R} d_r^k w_r^k$$

$$subject\ to$$

$$(x^k, w^k) \in Q^k$$

or, the objective function can be rearranged to

$$\bar{c}^k = min - \pi^k + \sum_{j \in J^k} (d_{0j}^k - \sum_{i \in I} \omega_i d_{ij}^k) x_j^k + \sum_{r \in R} d_r^k w_r^k$$

These decomposition models based on linear functions can be directly constructed or may result from a Dantzig-Wolfe decomposition. When resulting from a Dantzig-Wolfe decomposition, the subproblem variables are the original variables (the ones of the compact model before the decomposition is applied) and the parameters d_{0j}^k and d_{ij}^k, $i \in I, k \in K, j \in J^k$, correspond to the coefficients of the original variables in the compact model (in the objective function and in the constraints that are kept in the master problem when the decomposition is applied). The auxiliary subproblem variables correspond to original variables which do not appear in the linking constraints but only in constraints which were sent to the subproblem. Static variables correspond to original variables which appear in the linking constraints but not in constraints which were sent to the subproblem. In this context, it is relevant to note, we are restricting ourselves to the case where the subproblem variables are binary (the auxiliary variables may be of any type as they are not relevant for the definition of the feasible region of (D)).

11.3.2.3 Fixed Costs for Non-null Subproblem Solutions

We also consider the particular case where a non null subproblem solution has an additional cost. Its value may depend on the subproblem but is equal to all the non null solutions of the same subproblem. We represent it by e^k. As an example, let us consider a decompositiom model for a vehicle routing problem where the decision variable y_s^k is associated with route s for vehicle k and the null solution for k corresponds to not using vehicle k. If we want to model the cost of using a vehicle, it makes sense to consider it independently of the distance travelled by the vehicle, as long as the vehicle is used (i.e., y_s^k is not associated with the null solution). This cost is represented by e^k. Another component of the cost of route s may be a linear cost associated with the arcs included in route s.

Another example is for bin packing problems where a subproblem is defined for each bin and we intend to minimize the number of bins which are used to pack a set of items. The decision variable y_s^k is associated with packing pattern s for bin k and the null solution for k corresponds to not using bin k. In this case, $e^k = 1$, $k \in K$. In general, for $k \in K$, the function for computing the objective coefficient of the selection variable y_s^k becomes:

$$f_0^{k'}(x^k) = \begin{cases} 0 \text{ if } x^k = 0 \\ e^k + f_0^k(x^k, w^k) \text{ if } x^k \neq 0 \end{cases}$$

11.3.3 Algorithm

We now introduce the notation used in the presentation of a CG algorithm.

Given that π is constant, the subproblem k, which we represent by SP^k is

$$\begin{aligned} z_{SP^k} = & Min\, f_0^k(x^k, w^k) - \sum_{i \in I} \omega_i f_i^k(x^k) \\ & \text{subject to} \\ & (x^k, w^k) \in Q^k \end{aligned} \tag{11.11}$$

If the optimal solution of SP^k is the null solution, then

$$\bar{c}^k = -\pi^k + z_{SP^k}$$

otherwise it is

$$\bar{c}^k = e^k - \pi^k + z_{SP^k}$$

Of course, $\bar{c}^k < 0$ means the current dual solution is not optimal to the overall problem LRD and the selection variable corresponding to the solution obtained when solving the subproblem must be inserted in the RMP and this problem must be re-optimized.

Initialize the *RMP*, possibly with artificial variables for assuring feasibility
Initialize the SPs SP^k, $k \in K$
If the null solution is feasible for SP^k, insert the corresponding selection variable in the *RMP*
repeat
  Optimize the *RMP*
  $\pi, \omega \leftarrow$ optimal duals from the *RMP*
  $end \leftarrow true$
  for all $k \in K$ **do**
    Modify the objective function of SP^k to $f_0^k(x^k, w^k) - \sum_{i \in I} \omega_i f_i^k(x^k)$
    Optimize SP SP^k
    $z_{SP^k} \leftarrow$ optimal value of SP^k
    if $e^k - \pi^k - z_{SP^k} < 0$ **then**
      $s \leftarrow$ optimal solution of SP^k
      Obtain the RMP column associated with s by calculating the corresponding coefficients $c_s^k = e^k + f_0^k(x^k, w^k)$ and $a_{is}^k = f_i^k(x^k), \forall i \in I$
      Update the RMP by adding that column
      $end \leftarrow false$
    end if
  end for
until $end = true$

Fig. 11.2 Column generation algorithm

The CG algorithm is given in Figure 11.2. After the initializations, all the selection variables associated with null subproblem solutions are inserted in the RMP (if they are feasible). This excludes the necessity during the rest of the algorithm to test if the solution returned by the subproblem is the null solution or not. Then the main

cycle begins: the RMP is optimized, all the subproblems are solved and for each of them if the smaller reduced cost, given by $e^k - \pi^k - z_{SP^k}$, is negative the coefficients of the selection variable are calculated and the RMP updated. If that happens for any subproblem more iterations are required (*end* is false).

11.3.4 Example

We give an example of a CG algorithm for the linear relaxation of the instance of the time constrained shortest path problem introduced in Subsection 11.2.2.1.

The subproblem is a shortest path problem in a network with modified costs. This problem can be solved efficiently by a specific algorithm, e.g. Dijkstra, but here we consider the linear programming model to turn explicit the relation between the selection variables and the subproblem variables. A subproblem variable x_{ij} is equal to one if the arc ij is included in the path, and equal to 0 otherwise. The subproblem is:

$$
\begin{aligned}
& Min -\pi + \sum_{ij \in A} (c_{ij} - \omega t_{ij}) x_{ij} \\
& subject\,to \\
& x_{12} + x_{13} = 1 \\
& x_{12} + x_{32} = x_{24} + x_{25} \\
& x_{13} = x_{32} + x_{34} + x_{35} \\
& x_{24} + x_{34} = x_{45} + x_{46} \\
& x_{25} + x_{35} + x_{45} = x_{56} \\
& x_{46} + x_{56} = 1 \\
& x_{ij} \in \{0,1\}, ij \in A
\end{aligned}
$$

In order to initialize the RMP we consider path $1-3-5-6$ (with index $s=1$). The first RMP is then:

$$
\begin{aligned}
& Min\, 24 y_1 \\
& subject\,to \\
& y_1 = 1 \\
& 8 y_1 \leq 14 \\
& y_1 \geq 0
\end{aligned}
$$

The optimal primal solution of the RMP is $y_1 = 1$ and an optimal dual solution of the RMP is $\pi = 24$ and $\omega = 0$. Using this dual solution in the subproblem, its optimal solution is path $1-2-4-6$ with reduced cost -21. Given that the reduced cost is negative, this path $(s=2)$ is inserted in the RMP and a new iteration begins. The optimal primal solution of the RMP is $y_1 = 0.4, y_2 = 0.6$ and an optimal dual solution

of the RMP is $\pi = 40.8$ and $\omega = -2.1$. Using this dual solution in the subproblem, its optimal solution is path $1-3-2-5-6$ with reduced cost -4.8. Given that the reduced cost is negative, this path $(s = 3)$ is inserted in the RMP and a new iteration begins. The optimal primal solution of the RMP is $y_1 = 0, y_2 = 0.5, y_3 = 0.5$ and an optimal dual solution of the RMP is $\pi = 30$ and $\omega = -1.5$. Using this dual solution in the subproblem, its optimal solution is path $1-2-5-6$ with reduced cost -2.5. Given that the reduced cost is negative, this path $(s = 4)$ is inserted in the RMP and a new iteration begins.

The current RMP is:

$$
\begin{aligned}
&Min\, 24y_1 + 3y_2 + 15y_3 + 5y_4 \\
&subject\, to \\
&y_1 + y_2 + y_3 + y_4 = 1 \\
&8y_1 + 18y_2 + 10y_3 + 15y_4 \leq 14 \\
&y_s \geq 0, s = 1,2,3,4
\end{aligned}
$$

The optimal primal solution is $y_1 = y_2 = 0, y_3 = 0.2, y_4 = 0.8$. An optimal dual solution is $\pi = 35$ and $\omega = -2$. An optimal solution of the subproblem is path $1-3-2-5-6$ with reduced cost 0. As the reduced cost is non negative the optimal solution of the current RMP is an optimal solution to the overall (relaxed) problem.

11.3.5 Perturbed Column Generation

We now discuss how variables can be fixed without disrupting the CG algorithm, which is a major issue in SearchCol and in branch-and-price algorithms.

We define *perturbation* as the fixing of a subproblem variable to 0 or 1. When CG is applied to a model with perturbations, we use the term *perturbed CG*. The implementation of perturbed CG is based on relation (11.6), which allows imposing values on the subproblem variables without disrupting the CG method.

We implement perturbed CG by adding constraints to the RMP and modifying accordingly the objective coefficients of the subproblem variables in the subproblems. Although this is not the only way of doing it, it has the advantage that it does modify the subproblem feasible region, being less demanding for the subproblem than other types of perturbations.

In this type of perturbed CG, constraints that fix the desired subproblem variables are added to the RMP (named perturbation constraints). The subproblem is not modified except for taking into account the duals of the perturbation constraints in the objective coefficients of the subproblem variables.

We consider a *perturbed* CG RMP which differs from the regular RMP because it has a set P of additional constraints (indexed by p). Each additional constraint forces a subproblem variable to be 1 or 0:

$$x_j^k = b,$$

where b is 0 or 1, and appears in the RMP as

$$\sum_{s \in S^k} x_{js}^k y_s^k = b.$$

We associate a dual variable σ_p to each of these constraints, $p \in P$. The general objective function of the subproblem becomes:

$$Min - \pi^k + f_0^k(x^k, w^k) - \sum_{i \in I} \omega_i f_i^k(x^k) - \sum_{p \in P} \sigma_p x_{j(p)}^k$$

where $j(p)$ represents the index of the subproblem variable in the additional constraint p.

As an example, consider that we want a solution to the linear relaxation of the time constrained shortest path in which the arc $1-2$ is not used and the arc $2-5$ is used. Adding the corresponding constraints to the last RMP, we obtain:

$$\begin{aligned}
&Min\, 24y_1 + 3y_2 + 15y_3 + 5y_4 \\
&subject\, to \\
&y_1 + y_2 + y_3 + y_4 = 1 \\
&8y_1 + 18y_2 + 10y_3 + 15y_4 \leq 14 \\
&y_2 + y_4 = 0 \\
&y_3 + y_4 = 1 \\
&y_s \geq 0, s = 1,2,3,4
\end{aligned}$$

The objective function of the subproblem is now:

$$Min - \pi + \sum_{ij \in A} (c_{ij} - \omega t_{ij}) x_{ij} - \sigma_1 x_{12} - \sigma_2 x_{25}$$

The optimal primal solution is $y_1 = y_2 = 0, y_3 = 1, y_4 = 0$. An optimal dual solution is $\pi = 13$, $\omega = 0$, $\sigma_1 = -10$, and $\sigma_2 = 2$. An optimal solution of the subproblem is path $1-3-2-5-6$ with reduced cost 0. As the reduced cost is non negative the optimal solution of the perturbed RMP is an optimal solution to the overall perturbed problem.

Although this was not the case in this example, when perturbations are considered, additional solutions, generated by the subproblem, may be required to achieve an optimal solution.

11.4 Solving the MIP Decomposition Model

In this Section we provide an overview of solution methods to the decomposition MIP model. We first mention methods not based on column generation (CG) (subsection 11.4.1), then a usual heuristic which is based on applying CG once and

then obtaining an integer solution based on the columns of the last restricted master problem (RMP) solved (subsection 11.4.2). Then we describe the branch-and-price method (subsection 11.4.3), branch-and-price heuristics (subsection 11.4.4), and, finally, Lagrangean heuristics (subsection 11.4.5). From all these approaches to solve the decomposition MIP model of Section 11.2, only branch-and-price is an exact method.

11.4.1 Static Approaches

So far, we have discussed the decomposition MIP model and CG methods to solve its linear relaxation. We now describe an approach, which we name static approach, where the subproblem is used to generate a subset of selection variables, $\bar{S}$, $|\bar{S}| << |S|$ which defines a restricted problem - the problem restricted to the selection variables of the subset $\bar{S} \in S$. This problem, or its linear relaxation, is then optimized. If the integer restricted problem is considered then its optimal solution is the one given by the static approach. If the linear relaxation is solved then an additional procedure is needed to attempt an integer solution, for example, a rounding procedure. Of course, in any case, the obtained solution is an approximate solution to the overall problem (D).

A natural way of generating the subset of selection variables in a static approach is the application of heuristics. For example, in [50], each selection variable is associated with a subset of rectangular items packed in a rectangular bin. In order to generate the subset of selection variables, greedy procedures and fast constructive heuristic algorithms from the literature are applied. Another example where this type of approach was applied successfully is in airline crew scheduling problems [65]. In these problems, the subproblem is responsible for generating rotations (series of flights) which must obey several types of legal restrictions and have complex cost structures what makes difficult to turn it into an optimization problem with the dual variables of the RMP as parameters. However, more recent approaches usually model the subproblem through a network where the optimization is possible [42].

In [7], partial CG is described. This is a general concept that includes solving the subproblem such that only ”reasonable” columns are generated. It also includes the case where columns are generated by fast heuristics executed multiple times (with some different input).

11.4.2 MIP Based CG Heuristic

Instead of using a static set of selection variables, CG may be applied to define the set of selection variables - the ones present in an *optimal* RMP (a RMP for which its optimal solution is optimal for the overall *linear* problem). An *integer* solution can then be found by solving the *integer* RMP or by a problem-specific procedure

(again, rounding is the simplest example). We name the former as MIP based CG heuristic.

11.4.3 Branch-and-Price

So far, we have discussed approximate methods to solve the decomposition model (D). The previous methods are approximate because a selection variable with value 1 in an optimal solution to (D) may not be present in the set of selection variables generated by a static procedure or by CG (as CG solves the linear relaxation of the problem).

Branch-and-price is the combination of CG and branch-and-bound for obtaining an optimal solution to (D). In branch-and-price, each node of the branch-and-bound tree is solved by CG.

The main issue in a branch-and-price algorithm is how the branching is performed. It is well known that branching strategies based on the subproblem variables are preferable to branching strategies based on the selection variables.

A branching strategy based on the subproblem variables relies on the fact that the integrality of the selection variables may be imposed through the integrality of the subproblem variables. After a node of the (branch-and-price) search tree is optimized, values for the selection variables, y_s^k, are known. If they are not all integers, then a solution in the subproblem variables can be obtained through (11.6). Note that if all x_j^k are integer then the solution y_s^k was integer in the first place, because no duplicate selection variables are allowed. Note also that if y_s^k were all integers, then the subproblem solution is also integer. If there is a fractional variable y_s^k, there is at least one fractional variable x_j^k which may be used for creating the branches $x_j^k = 0$ and $x_j^k = 1$. Perturbed CG is used to solve the nodes of the search tree and the search is performed as in standard branch-and-bound.

11.4.4 Branch-and-Price Heuristics

Assuming a branch-and-price algorithm exists for problem (D) which is based on branching on the subproblem variables, the extension of branch-and-bound heuristics to branch-and-price heuristics is straightforward.

For example, the only difference in dive-and-fix (for example, [66]), relax-and-fix (for example, [66]), fix-and-relax [17], and beam search [51] heuristics, consists in how the solution of the linear programming model with some variables fixed is obtained - in a branch-and-price version of these heuristics, perturbed CG is used (with the perturbations fixing subproblem variables). For other examples of heuristics based on branch-and-bound that may be modified in a straightforward manner to accomodate CG, see [7]. In the same reference, iterative CG is described. In this approach the linear relaxation of a MIP is first solved by CG, then branch-and-bound is applied but additional columns are generated only when an heuristic criterion says

so (the example given is that the value of the node is too far from the optimum of the linear relaxation). Combinations of decomposition approaches and MHs are proposed in [53], where a review on integer programming and MHs hybrids is done and hybrid approaches based on Lagrangean relaxation and CG are applied to the knapsack contrained maximum spanning tree problem and to a periodic vehicle routing problem.

11.4.5 Lagrangean Heuristics

It is well known that Dantzig-Wolfe decomposition and Lagrangean relaxation can be seen as dual decomposition techniques [28]. Therefore, for each problem in which Lagrangean relaxation has been applied (and in which the variables present in the subproblem and present in the dualized constraints are all binary), a decomposition model (D) can be defined. Inversely, the linear relaxation of model (D) can be approached by Lagrangean methods (e.g. subgradient methods) and problem (D) by Lagrangean heuristics.

Lagrangean heuristics have been applied widely. In these approaches, the decomposition model (D) is not considered explicitly. Subproblem solutions are generated at different dual points (for example, by a subgradient method) and it is attempted to modify them in such a way that the global constraints become feasible. This is done in a problem-specific manner (for references on these approaches, see [7], for a general description, see [15]).

11.5 A Combinatorial Decomposition Model

In this Section we introduce the combinatorial oprimization (CO) decomposition model which is the base for the metaheuristic (MH) search conducted in SearchCol. In subsection 11.5.1 we describe the CO decomposition model and the associated concepts of how a solution is represented and what is the search space. In 11.5.2 we describe how solutions are evaluated.

11.5.1 Solution Representation and Search Space

The general problem being addressed may be seen as a CO problem consisting in selecting one element from each partition, such that the sum of the cost associated with each element is minimum and the global constraints are satisfied. If this problem can be modelled through the decomposition MIP model (D) (see Section 11.2), it can also be modelled through the following CO decomposition model (C):

$$
\begin{aligned}
& Min\, f(s) + \lambda g(s) && (C)\\
& subject\, to\\
& s(k) \in S^k, k \in K
\end{aligned}
$$

where $s(k)$, $k \in K$, represents a subproblem solution of partition k. An overall solution is represented by s, which is a vector $s = (s(1), s(2), ..., s(|K|))$, where each component is a solution from a subproblem. A feasible solution s belongs to $S = S^1 \times S^2 \times ... \times S^{|k|}$. The function $f(s)$ is the cost of the solution with the most favorable values for the static variables. Function $g(s)$ provides a measure of the infeasibility of solution s with respect to the global constraints (11.2). The parameter λ takes a value such that feasible solutions have a lower value than infeasible solutions. Note that a solution s where $g(s) > 0$ is infeasible for the problem but is feasible for this model and, in fact, can even be an incumbent solution at some point in the execution of an algorithm to solve (C). We give the details on how solutions are evaluted in the next subsection.

An important issue in (C) is that the representation of solutions is problem independent. For example, for a problem with four subproblems, a solution $s = (2,4,1,1)$ means that the second subproblem solution is chosen for the first subproblem, the fourth subproblem solution is chosen for the second subproblem, the first subproblem solution is chosen for the third subproblem, and the first subproblem solution is chosen for the fourth subproblem. These subproblems solutions can correspond to paths, trees, machine schedule, routes, ot others, depending on the problem.

Three fundamental characteristics on the design of a MH [63] are assured by this representation: completeness (all solutions can be represented), connexity (it is possible to go from any solution to any other - see neighborhood structures in Section 11.7), and efficiency of the search operators. A drawback is that when there are static variables (in particular, binary or general integer) the evaluation (see next subsection) may become too heavy.

A feasible solution in model (D) can be translated into a solution in model (C) and *vice-versa*. A variable $y_s^k = 1$ in a solution of (D) implies the component $s(k) = s$ in a solution of (C). A component $s(k) = s$ in (C) implies that $y_s^k = 1$, if $s = s(k)$, and $y_s^k = 0$, otherwise, in (D).

The number of solutions in the search phase is $n_1 \times n_2 \times ... \times n_{|K|}$ where n_k is the number of subproblem solutions associated with subproblem k, $k \in K$. The number of solutions of the combinatorial model is exponential. Therefore, model (C) is useful only if the search space S can be restricted, not excluding "good" subproblem solutions (the ones present in optimal or near optimal solutions). In SearchCol, CG is used to restrict the search space of model (C). The search is conducted only over subproblem solutions generated with (perturbed) CG.

We now extend the solution representation for the cases mentioned in the last paragraphs of subsection 11.2.1. When the subproblems are identical, a solution is represented by a set of subproblem solutions. For example, $s = \{7,4,1,3\}$. If the number of subproblem solutions have an upper bound, empty positions / null

subproblem solutions are considered. For example, for an upper bound of 4, a solution made of two subproblem solutions is $s = \{0,4,0,3\}$, where 0 is the index for null subproblem solutions.

11.5.2 Evaluating Solutions and Moves

In this subsection, we describe how solutions (including partial solutions) and moves are evaluated in SearchCol. A move corresponds to the addition or removal of a component of a solution. We describe several alternatives which are controlled through parameters which are detailed in subsection 11.6.2. All evaluations comprise two values: the feasibility value and the infeasibility value. A solution/move with a lower value of infeasibility is always better than a solution/move with a larger value. If the infeasibility values of two solutions/moves are equal, the feasibility value is used to compare them. In the next subsubsections, we detail what are the alternatives for determining the feasibility and infeasibility values of a solutions and moves.

11.5.2.1 Models without Static Variables

Full Solutions

When there are no static variables, we evaluate the feasibility of a solution by using one of three functions:

- $evalNumberViolated(s)$ returns the number of violated constraints in solution s;
- $evalAmountViolation(s)$ returns the total amount of violation of the constraints in solution s;
- $evalWeightedViolation(s) = 1000 evalNumberViolated(s) + evalAmountViolation(s)$.

The amount of violation of a constraint is calculated based on the value of the slack which is defined as the right-hand side value minus the left-hand side value. If it is an equality constraint, the amount of violation is the absolute value of the slack. If it is a less han or equal to constraint, if the slack is negative than the amount of violation is the symmetric of the slack, otherwise it is zero. If it is a greater-than or equal to constraint, if the slack is positive than the amount of violation is the slack, otherwise it is zero.

The feasibility value of a solution s is given by

$$evalOriginalCosts(s) = \sum_{k \in K} c^k_{s(k)}.$$

Partial solutions

In a partial solution, the subproblem solutions of some subproblems are known (we represent the set of these subproblems by K^1) and the subproblem solutions of the

other subproblems are unknown (we represent the set of these subproblems by K^0). Partial solutions are the building blocks of constructive heuristics.

Partial solutions can be evaluated by the four previous functions by considering only the components that belong to the partial solution (replacing the K by K^1). Additionally, the feasibility value of a partial solution may also be obtained by

$$evalReducedCosts(s) = \sum_{k \in K^1} \bar{c}^k_{s(k)}$$

where $\bar{c}^k_{s(k)}$ is the reduced cost of the selection variable $y^k_{s(k)}$.

The rationale behind the use of reduced costs is that when evaluating partial solutions, a solution with a lower actual objective function but higher reduced cost may be worse than one with higher actual objective function but lower reduced cost, because the reduced cost takes into account the influence of the variable in the constraints. An evaluation based on reduced cost is less greedy than one based on the actual cost. This idea was proposed in [26] for set covering and set partitioning problems.

Moves

There are two elementary moves in SearchCol: adding a subproblem solution to a partial solution and removing a subproblem solution from a partial or full solution. The add move is represented by the function $s'' = addMove(s', s(k))$. Assuming s' has not a subproblem solution associated with k. The remove move is represented by the function $s'' = dropMove(s', s(k))$. Assuming s' has a subproblem solution associated with k.

In both moves, the feasibility value of the solution can be obtained by summing (subtracting) the cost or reduced cost of the subproblem solution being added (removed). The update of the infeasibilities value requires, for each global constraint where the subproblem solution has a non zero coefficient, the calculation of the new slack, if the constraint is violated, and its amount of violation.

Based on these two elementary moves another one can be defined which replaces the solution of a subproblem by another:

$$s'' = replaceMove(s, s(k)) = addMove(DropMove(s, s'(k)), s(k))$$

Where $s'(k)$ is the solution of subproblem k in solution s. If $s(k) = s'(k)$ the same solution s is obtained, i.e. $s'' = s$.

11.5.2.2 Models with Static Variables

Full solutions

In the most general decomposition model, a solution s is evaluated by solving the MIP (E):

$$Min \sum_{t\in T} c_t y_t + \sum_{k\in K} c^k_{s(k)} + M \sum_{i\in I^=} (a_i^+ + a_i^-) + M \sum_{i\in I^\leq} a_i^- + M \sum_{i\in I^\geq} a_i^+ \qquad (E)$$

subject to

$$a_i^+ - a_i^- + \sum_{t\in T} a_{it} y_t = b_i - \sum_{k\in K} a^k_{is(k)} \qquad i \in I^=$$

$$-a_i^- + \sum_{t\in T} a_{it} y_t \leq b_i - \sum_{k\in K} a^k_{is(k)} \qquad i \in I^\leq$$

$$a_i^+ + \sum_{t\in T} a_{it} y_t \geq b_i - \sum_{k\in K} a^k_{is(k)} \qquad i \in I^\geq$$

$$y_t \in \{0,1\} \qquad t \in T^b$$

$$y_t \geq 0 \text{ and integer} \qquad t \in T^i$$

$$y_t \geq 0 \qquad t \in T^c$$

$$a_i^+, a_i^- \geq 0 \qquad i \in I^=$$

$$a_i^- \geq 0 \qquad i \in I^\leq$$

$$a_i^+ \geq 0 \qquad i \in I^\geq$$

Model (E) includes the artificial variables, represented by a_i^+ and a_i^-, for equality constraints (set $I^=$), less than or equal to constraints (set $I^\leq$), and greater than or equal to constraints (set $I^\geq$). Artificial variables have a large objective function coefficient, M. This allows still evaluating solution s even if it is not feasible in (D) (if there are no values for the static variables such that all constraints are obeyed). Value M must be such that that an optimal solution to (E) has all artificials with value zero, unless there are no feasible solutions with all artificials with value zero.

The infeasibility value of a solution is given by the sum of the values of the artificial variables. The feasibility value is defined only when the sum of the value of the artificial variables is zero and is equal to the optimal value of (E).

Note that model (E) is much easier to solve than model (D) since it does not have selection variables: $\sum_{k\in K} c^k_{s(k)}$ and $\sum_{k\in K} a^k_{is(k)}$ are constants. Model (E) is a linear programming model if all static variables are continuous.

When there are static variables, it may be possible to evaluate a solution in a more efficient manner if problem-specific characteristics are used. For example, in the min-max problem for single path routing, given the set of paths (one for each commodity) the maximum load of an arc can be obtained much more efficiently by computing the highest value among all arc loads than by solving a linear programming model.

Partial solutions

The evaluation of full solutions is easily extendable for partial solutions. In this case, the selection variables associated with the subproblems belonging to K^0 are set to 0.

Moves

The two types of elementary moves, *addMove* and *dropMove*, defined for the case when there are no static variables are not efficient for the case where there are static variables. In this case, a MIP (or LP) must be solved for each move. To alleviate this issue, we define the concept of optimal move. An optimal move is associated with one subproblem k and results from the optimization of the MIP problem where the selection variables associated with the other subproblems are fixed to 1 or 0. All the the selection variables associated with subproblem solutions in the current solution are fixed to 1 (subproblems in set K^1). All the others selection variables, except the ones associated with the subproblem defining the move, are fixed to 0 (subproblems in set K^0). After the optimization, the subproblem solution for subproblem k is obtained directly from the values of the selection variables. We represent this function by

$$s' = optimalMove(s,k)$$

An extension of the optimal move concept for defining a set of subproblems solutions is straighforward by defining the set K^* of those subproblems instead of only one subproblem.

11.6 SearchCol

In this Section, we detail the main concepts of the SearchCol framework. In subsection 11.6.1 we give an overview of SearchCol including the description of its main parameters. In subsection 11.6.2, the parameters related with evaluating solutions are introduced. In the following subsections, we detail the two components involved in the exchange of information between column generation (CG) and metaheuristic (MH) search: initial solutions (subsection 11.6.3) and perturbations (subsection 11.6.4). In the last subsection, subsection 11.6.5, we describe the stopping criteria. A fundamental component of SearchCol is (perturbed) CG which was already detailed in Section 11.3. The other fundamental component is the MH search which will be detailed in Section 11.7.

11.6.1 Overview

The starting point for SearchCol is a MIP model with an exponential number of decision variables, each of them associated with a solution of a subproblem, i.e. the MIP decomposition model (D) of Section 11.2. The first step in a SearchCol algorithm is solving the linear relaxation of the MIP model by CG. In the second step, the subproblem solutions generated by CG are seen as the components of an overall solution in the combinatorial decomposition model (C). A MH search based on the representation of a solution as being made of one solution from each subprob-

lem is conducted. This representation, and all the algorithmic components defined for the MH search, are problem independent. In the next step, a perturbation for CG is defined. A perturbation consists in a set of additional constraints in the restricted master problem (RMP) of CG, each one forcing one decision variable of the subproblem to have the value 1 or 0. This is accomplished by representing the subproblem variable through variables in the RMP as detailed in subsection 11.3.5. The aim of the perturbated CG is the generation of new subproblem solutions, that, hopefully, will improve the incumbent solution. After perturbed CG, a new search begins and then another iteration starts with the definition of a perturbation for CG. This cycle is repeated until one of the stopping criteria is met, as can be seen in Figure 11.3.

```
1: Column generation
2: Search
3: repeat
4:     Define column generation perturbation
5:     Optimize perturbed column generation
6:     Search
7: until Stopping criterion fullfilled
```

Fig. 11.3 SearchCol algorithm

In each iteration of SearchCol, a different search space is defined. The first search space corresponds to the subproblem solutions obtained during the linear relaxation. In each iteration, this search space is enlarged by the subproblem solutions generated through perturbed CG. After a perturbed CG step, the subproblem solutions with higher reduced costs (in the linear relaxation and not in the current iteration) may be removed from CG and from the search space. As many aspects of SearchCol, the number of columns that triggers the removal of subproblem solutions is given by a parameter, *PARDECmaxnumcols*. The dimension of the basis of the linear relaxation of (D), $|K|+|I|$ (the number of selection constraints plus the number of global constraints), is used as the unit for this parameter. If the number of subproblem solutions exceeds $PARDECmaxnumcols \times (|K|+|I|)$, a purge of columns is performed. Parameter *PARDECmaxnumcols* must be ≥ 1. Note that if the reduced cost of a variable (in CG with no perturbations) is greater than the difference between the value of the incumbent solution and the lower bound provided by CG, any solution with the corresponding subproblem solution will always be worst than the incumbent (by definition of reduced cost). Removal of these type of subproblem solutions can only improve the efficiency of the search and never worsten its effectiveness.

The search space of a given iteration is influenced by perturbations. The subproblem variables with value 1 can be seen as attributes (features) of a solution that are desired or avoidable. If, in a perturbation, a subproblem variable is fixed to 1 then, in general, more subproblem solutions with the corresponding attribute will be generated by (perturbed) CG. If a subproblem variable is fixed to 0 then, in general,

subproblem solutions without the corresponding attribute will be generated by (perturbed) CG. All previous perturbations are disregarded when defining the perturbed CG for an iteration.

In order to conduct the search in the overall search space, there are several alternatives for defining perturbations. When the incumbent solution is infeasible, we use perturbations based on subproblem variables which contribute to the constraints being violated. When the incumbent solution is feasible there are the following alternatives for perturbations: (i) some subproblem variables in the constraints with higher (absolute) duals, (ii) some subproblem variables with values close to 1 in the last (perturbed) CG step, (iii) some subproblem variables in the incumbent, (iv) some subproblem variables based on the memory of the search.

To be used in memory perturbations, the search step keeps two types of memory. The recency memory value of a subproblem variable is incremented each time the solution variable is present in a good quality solution (e.g. a local optimum). The frequency memory of a subproblem variable is incremented each time the solution variable is present in a representative solution (e.g. a current solution in local search).

One way of achieving intensification and diversification is by applying different perturbations at different iterations. We define two type of iterations: in a *plus* iteration (related to intensification) the incumbent was improved in the previous iteration and in a *minus* iteration (related to diversification) that did not happen. In a plus iteration, good attributes are reinforced (for example, subproblem variables with value 1 in the incumbent, or with value 1 in many local optima are fixed to 1). In a minus iteration, attributes already explored are forbidden (roughly speaking, solutions with that attribute were already explored and the incumbent was not improved, therefore the corresponding subproblem variable is set to 0). Perturbations are described in detail in subsection 11.6.4.

Perturbations may be seen as an input for CG provided by the search. Information for defining an initial solution may be seen as an input for search provided by CG (of course, another input is the search space itself). In SearchCol, there are several alternatives for defining how the initial solution is obtained: (i) randomly, (ii) based on the values of the primal variables associated with the last RMP solved (deterministically or randomly), (iii) based on when the subproblem solutions were generated, (iv) through constructive greedy heuristics (deterministic or randomized). The alternatives for obtaining initial solutions are described in detail in subsection 11.6.3.

SearchCol defines four stopping criteria: (i) a given time limit is reached, (ii) a given number of iterations without incumbent improvement is reached, (iii) a desired quality of the incumbent is achieved, (iv) a given number of search steps is reached. Stopping criteria are described in detail in subsection 11.6.5.

Note that CG provides a lower bound (in minimization problems) that may be used to calculate the quality of the incumbent solution to be used in the third stopping criterion. This lower bound also allows the assessment of the quality of the solution obtained, which is not usually possible with usual MHs.

In Table 11.2, the main parameters of SearchCol and their purpose are provided. The algorithm with a detail in which these parameters are present is given in Figure 11.4.

We remind that CG was already addressed in Section 11.3 and search (shaded in Figure 11.4) will be detailed in the next Section. All the main parameters and other parameters are detailed in the following subsections.

Table 11.2 Main parameters of SearchCol

Parameter	Purpose
PARSCinfeasobj	Evaluation of infeasible solutions
PARSCfeasobj	Evaluation of feasible partial solutions
PARSCinitialfirst	Determination of an initial solution the first time a search is conducted
PARSCinitialother	Determination of an initial solution *not* in the first time a search is conducted
PARSCmakefeasible	Specification of the perturbation to use when the incumbent is infeasible
PARSCperturbationplus	Specification of the perturbation to use when there was an improvement in the previous iteration
PARSCperturbationminus	Specification of the perturbation to use when there was no improvement in the previous iteration

11.6.2 Evaluating Solutions

In Table 11.3 and in Table 11.4 the possible values and corresponding functions, introduced in subsection 11.5.2, used for the two parameters related with evaluating solutions are given, parameters *PARSCinfeasobj* and *PARSCfeasobj*. These parameters define the evaluation of solutions only for models *without* static variables. The parameter *PARSCfeasobj* only influence the evaluation of partial solutions (in particular in greedy procedures). For full solutions, the evaluation is based on the real costs because it is the only meaningful value since it gives the true objective function value. For models with static variables, the evaluation is done by solving a MIP and a parameter *PARSCevalmipmaxtime*(>0) sets the maximum time allowed for its optimization.

Table 11.3 Possible values and associated functions for SearchCol parameter *PARSCinfeasobj*

Value	Function
2	evalNumberViolated(s)
3	evalAmountViolation(s)
4	evalWeightedViolation(s)

Table 11.4 Possible values and associated functions for SearchCol parameter *PARSCfeasobj*

Value	Function
2	evalOriginalCosts(s)
3	evalReducedCosts(s)

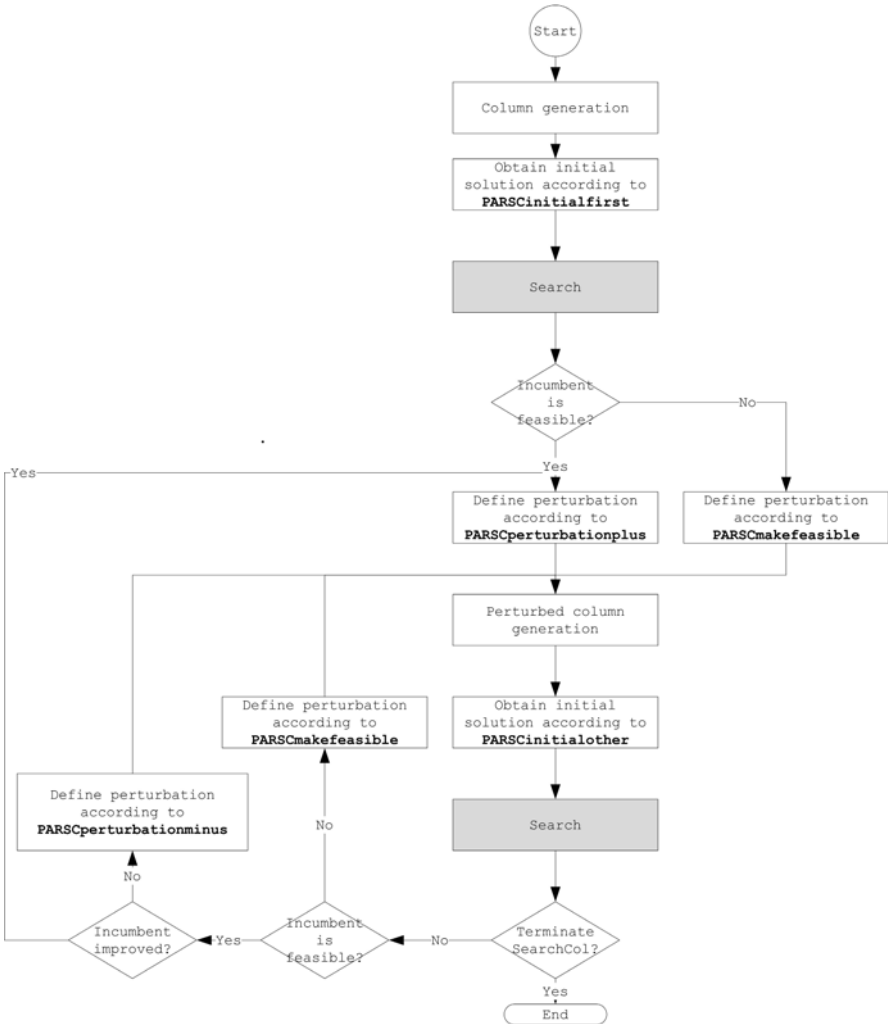

Fig. 11.4 Detailed algorithm for SearchCol

11.6.3 Initial Solutions

11.6.3.1 Uniform Random

In this alternative, a solution is defined randomly, based on the uniform distribution. In this case, each subproblem picks a solution randomly with all the subproblem solutions having the same probability of being chosen. We represent the associated function by $genSolutionRandomUniform()$.

11.6.3.2 Based on the CG Optimal Solution

In the first alternative, the subproblem solution selected is the one associated with the selection variable with higher solution value in the last RMP solved. We represent the associated function by $genSolutionRounding()$. In the second alternative, a solution is defined randomly, but now the probability of a subproblem solution being chosen is given by its solution value in the last RMP solved. We represent the associated function by $genSolutionRandomBiased()$.

11.6.3.3 Based on the CG History

In the first alternative, the selected subproblem solution is the last one that was generated by the subproblem. Note that this subproblem solution may have a value 0 at the last RMP. Since after this subproblem solution no more solutions from the subproblem were generated, the solution can be viewed as the one which stabilized the subproblem. We represent the associated function by $getLastCreatedCGSolution()$. In the second alternative, the selected subproblem solution is the first one that was generated by the subproblem. In general, this subproblem solution corresponds to an optimal solution of the subproblem if the global constraints were removed from the model. In that sense, is the best solution that the subproblem may provide. We represent the associated function by $getFirstCreatedCGSolution()$. In the third alternative, the selected subproblem solution is the one obtained when the subproblem was solved for the last time, i.e. when the dual solution was optimal. For that reason, most Lagrangean heuristics start from these subproblem solutions. We represent the associated function by $getCGSolution()$. Note that the first and third alternatives are different because the last subproblem solution generated by the subproblem is not necessarily the one obtained in the last iteration of the RMP. In fact, in the last iteration of CG no subproblem solutions are generated, otherwise at least one more iteration was required.

11.6.3.4 Greedy Construction

A solution can be constructed based on greedy procedures. In the first alternative, a deterministic greedy procedure is used. In each step, the subproblem solution which

conducts to a better (partial) solution is chosen. Of course, solutions from subproblems for which there are already a solution are excluded from this candidate list. We represent the associated function by $constructDeterministicGreedy()$. The other alternative is the randomized constructive phase of GRASP [57], where in each iteration a subproblem solution is randomly chosen from a restricted candidate list which size is controlled by a parameter $\alpha \in [0,1]$. We represent the associated function by $constructRandomGreedy(\alpha)$. For $\alpha = 1$ the randomized greedy construction is the same as $genSolutionRandomUniform()$ and for $\alpha = 0$, the the randomized greedy construction is the same as $constructDeterministicGreedy()$.

For models without static variables, the constructive step is straightforward through the use of the functions *addMove* (for evaluating potential components and perform the move) and $dropMove$ (for restoring the current partial solution after an add move was done for evaluation). For models with static variables, the elements of the restricted candidate list are the best solution from each subproblem (with no solution in the current iteration). The best solution from each subproblem is obtained by applying the optimal move described in subsubsection 11.5.2.2.

11.6.3.5 Initial Solution Parameters

We make a distinction between how an initial solution is obtained for the first search step and for the other search steps inside the loop (see Figure 11.4). For example, $getCGSolution()$ can be used in the first situation and $constructDeterministicGreedy()$ in the second. We define the parameters $PARSCinitialfirst$ and $PARSCinitialother$ for each of these situations. For $PARSCinitialother$ an adiditional value, 8, is available, corresponding to use the incumbent solution as the initial solution. In this case, CG was used in the last iteration only for modifying the search space. Their possible values and corresponding functions are given in Table 11.5.

Table 11.5 Possible values and associated functions for SearchCol parameters $PARSCinitialfirst$ and $PARSCinitialother$

Value	Function
0	genSolutionRounding()
1	genSolutionRandomUniform()
2	genSolutionRandomBiased()
3	getLastCreatedCGSolution()
4	getFirstCreatedCGSolution()
5	getCGSolution()
6	constructDeterministicGreedy()
7	constructRandomGreedy($PARSCalpha$)

In Table 11.5, $PARSCalpha$ belongs to $[0,1]$ and corresponds to the α parameter of the constructive phase of GRASP.

11.6.4 Defining Perturbations

As defined in subsection 11.3.5, a perturbation is a constraint inserted in the RMP that fixes a subproblem variable to 0 or to 1. We define different alternatives for defining perturbations depending on the feasibility of the incumbent solution.

11.6.4.1 Infeasible Incumbent

When the incumbent solution is infeasible, we define two alternatives for defining the set of perturbations. In both of them, we identify the set of subproblem variables that imply a non null coefficient in the constraints that are being violated. Then we use Table 11.6 to decide if the subproblem variable is fixed to 0 or is fixed to 1. As an example, consider the second and third lines of that table which correspond to less than or equal to violated constraints. Since the constraint is violated, the slacks are negative (by definition, the value of the slack is right-hand side minus the left-hand). In the first line of the table, a subproblem variable has a positive coefficient (more rigorously, the selection variables associated with subproblem solutions where the variable has value 1 have a positive coefficient) in the constraint. Therefore, it has a positive contribution to the violation of the constraint, which we attempt to avoid by fixing its value to zero. In the second line of the table, a subproblem variable has a negative coefficient in the constraint. Therefore, it has a negative contribution to the violation of the constraint, which we attempt to reward by fixing its value to one.

Table 11.6 Determining if a subproblem variable is fixed to 0 or 1

Sense of the constraint	Signal of the slack	Sign of the coefficient of the subproblem variable	Fixed to
$\leq$	-	+	0
$\leq$	-	-	1
$\geq$	+	+	1
$\geq$	+	-	0
=	-	+	0
=	-	-	1
=	+	+	1
=	+	-	0

If a subproblem variable is forced to 0 and simultaneously to 1 by two different perturbations, no perturbations on that variable are considered. In many models, this situation does not occur because global constraints have some type of structure. For example, in the min cost single path routing, all the constraints are less than or equal to constraints and the contributions of the variables are always positive.

The subproblem variables considered can be the ones of the incumbent solution or all the subproblem variables. The functions *makeFeasibleInc*() and

makeFeasibleAll() represent these two alternatives. For example, if the capacity of an arc is being violated in a min cost single path routing problem, and the second perturbation is used, the arc will be forbidden for all the commodities. If the perturbation based on the incumbent is used, the arc is forbidden only for the commodities that use the arc in the incumbent solution.

When the incumbent is infeasible the perturbation to used is given by the parameter *PARSCmakefeasible* with the possible values and associated functions given in Table 11.7.

Table 11.7 Possible values and associated functions for SearchCol parameter *PARSCmakefeasible*

Value	Function
0	makeFeasibleInc()
1	makeFeasibleAll()

11.6.4.2 Feasible Incumbent

If the current solution is feasible, we define the following perturbations.

Perturbation based on the duals

A parameter $p \in [0,1]$ defines the proportion of global rows that will be considered for the perturbation. The ones with higher duals in absolute value are selected. The subproblem variables appearing on those rows and belonging to the incumbent are fixed to 1 or 0 depending on argument b. The rationale behind the choice of the rows with higher values of the duals is using the dual values to estimate the importance of the rows, in the sense of the sensitivity analysis of linear programming. For example, in the min cost single path routing problem, the dual of a row corresponds to the marginal value of the capacity associated with the arc (of course in the linear relaxation of the problem). Arcs with higher duals are more attractive arcs for the commodities than the others, as the arcs may be seen as scarce resources. In the limit, a row with a dual variable equal to zero could be removed from the model without modifying the (linear relaxation) optimal solution. Function *perturbBasedDuals*(p,b) represent this perturbation.

Perturbations based on column generation

In this type of perturbation the optimal solution of the last CG solved is used as well as a real parameter $p \in [0,1]$. The subproblem variables with a value greater than or equal to $1-p$ in the fractional solution given by the last CG are fixed to b,

which may take value 1 or value 0. Function $perturbBasedCG(p,b)$ represent this perturbation.

Perturbation based on the incumbent

In this type of perturbation the incumbent solution is used. Some subproblem variables with value 1 in the incumbent solution are fixed to b, which may take value 1 or value 0. Function $perturbBasedIncumbent(p,b)$ represents this perturbation where p stands for the proportion of variables, randomly selected, to be fixed.

Perturbation based on memory

The memory for defining perturbations is based on subproblem variables. The recency memory value of a subproblem variable is incremented each time the solution variable is present in a good quality solution (e.g. a local optimum). The frequency memory of a subproblem variable is incremented each time the solution variable is present in a representative solution (e.g. a current solution in local search). The recency memory is used for fixing to b, which may take the value 1 or the value 0, all the variables with a recency value higher than a threshold (given as a proportion by a parameter p). When using $b = 1$, we intend to preserve good quality attributes of the subproblem solutions. The frequency memory is used for fixing to b, which may take the value 1 or the value 0, all the variables with a frequency value higher than a threshold (given as a proportion by a parameter p). When using $b = 0$, we intend to avoid attributes of subproblem solutions that were already present in many portions of the search space explored so far. We represent these perturbations by functions by $perturbBasedRecency(p,b)$ and $perturbBasedFrequency(p,b)$

Parameters for perturbations

The actual perturbation used in an iteration of SearchCol depends on two parameters. If the there was an improvement in the last iteration, the perturbation is defined

Table 11.8 Possible values, associated functions and arguments for SearchCol parameters *PARSCperturbationplus* and *PARSCperturbationminus*

Value	Function	p
1	$perturbBasedDuals(p,1)$	PARSCproportionrows1
2	$perturbBasedDuals(p,0)$	PARSCproportionrows0
3	$perturbBasedCG(p,1)$	PARSCbinarythreshold1
4	$perturbBasedCG(p,0)$	PARSCbinarythreshold0
5	$perturbBasedIncumbent(p,1)$	PARSCproportionvariables1
6	$perturbBasedIncumbent(p,0)$	PARSCproportionvariables0
7	$perturbBasedRecency(p,1)$	PARSCrecencythreshold1
8	$perturbBasedRecency(p,0)$	PARSCrecencythreshold0
9	$perturbBasedFrequency(p,1)$	PARSCfrequencythreshold1
10	$perturbBasedFrequency(p,0)$	PARSCfrequencythreshold0

by the value of *PARSCperturbationplus*. Otherwise, the perturbation is defined by *PARSCperturbationminus*. For example, $perturbBasedMemory(0.5,1)$ can used in the first situation and $perturbBasedCG(0.1,0)$ in the second. The possible values, corresponding functions and arguments (which are themselves SearchCol parameters) are given in Table 11.8. The parameter p is a real belonging to the $[0,1]$ interval.

11.6.5 Termination

SearchCol has four stopping criteria: if one of them is verified, the algorithm stops and the best solution found so far is returned. The first criterion is a time limit. The second criterion is a maximum number of iterations without improvement of the incumbent solution. The third criterion is the maximum number of iterations. The last criterion is the relative gap being less than a given parameter.

$$|z_{inc} - z_{LR}|/|z_{inc}|$$

where z_{inc} is the value of the incumbent solution and z_{LR} is the value of the linear relaxation given by CG (without perturbations).

In Table 11.9 the parameters used for implementing the stopping criteria and their possible values are presented.

Table 11.9 Parameters and possible values for SearchCol stopping criteria

Parameter	Possible values
PARSCmaxtime	> 0
PARSCmaxnumiterwithoutimprov	> 0
PARSCmaxnumtotaliter	> 0
PARSCrelgap	$[0,1]$

11.7 Metaheuristic Search in SearchCol

In this Section we detail how the search steps of SearchCol are performed and provide examples of actual SearchCol algorithms. In subsection 11.7.1 we introduce additional algorithmic components that are used, together with the ones presented in the last section, for the definition of (hybrid) metaheuristics (MHs) for the search steps. Subsections 11.7.2, 11.7.3, and 11.7.4 are devoted to the description of Search algorithms with three different search approaches: multi-start local search (MSLS), variable neighborhood search (VNS) and MIP.

11.7.1 Additional Algorithmic Components

11.7.1.1 Local Search

We define the k-neighborhood of a solution as the set of solutions which are obtained by changing the subproblem solution of k or less subproblems. For example, in single path routing, a 1-neighbor is a solution which is obtained by changing the path of one commodity and a 2-neighbor is a solution which is obtained by changing the paths of one commodity or the path of two commodities.

The size of the 1-neighborhood is $\sum_{k\in K}(n_k - 1)$ where n_k is the number of subproblem solutions for subproblem k. For 1-neighborhood we define the function $neighbor1(s,d)$ which returns the best solution in the neighborhood or the first solution better than the current one found. The parameter d specifies the chosen alternative.

The size of the 2-neighborhood is $\sum_{k1\in K}(n_{k1} - 1)\sum_{k2\in K:k2>k1}(n_{k2} - 1)$. For 2-neighborhood we define the function $neighbor2(s,d)$ which has a similar interpretation to $neighbor1(s,d)$.

Based on the functions $neighbor1(s,d)$ and $neighbor2(s,d)$ four different local search algorithms can be defined according to the type of neighborhood (1-neighborhood or 2-neighborhood) and the descent strategy (best or first improvement). In Table 11.10 the value 0 to *PARSCdescentstrat* corresponds to best improvement and 1 to first improvement.

Table 11.10 Parameters and possible values for SearchCol local search functions

Parameter	Possible values
PARSCneigh	1 and 2
PARSCdescentstrat	0 and 1

11.7.1.2 Initial Solution Based on Other Solution

In this subsubsection, we introduce how a solution can be obtained based on other solution.

Random solution from the k-neigborhood

The first function of this type returns a randomly generated k-neigborhood solution. A proportion of $1-p$ subproblem solutions are kept from the base solution, where $p \in]0,1[$. The solutions for the remaining subproblems are randomly chosen. We represent this function by $perturbRandomly(s,p)$.

Solutions based on memory

The MH framework incorporates medium- and long-term memories based on subproblem solutions (note that this memory is different from the memory used in defining perturbations, which is based in subproblem *variables*). Both are based on having one integer (for each type of memory) associated with each subproblem solution counting how many times it appeared in representative solutions of the desired type of memory. For medium-term memory, we are interested in recording good quality solutions. An example of a representative set of solutions is a set of local optima solutions. For long-term memory, we are interested in recording solutions representative of portions of the search space already explored. An example of a representative set of solutions is the set of solutions that were current solutions at some time in a local search algorithm. Functions $perturbBasedRecencySols(s,p)$ and $perturbBasedFrequencySols(s,p)$ represent these two alternatives of getting a solution based on another one. Both functions sort the subproblem solutions by non increasing order of recency/frequency. For recency, the first $p \in]0,1[$ times the number of subproblems are kept. For frequency, the first $p \in]0,1[$ times the number of subproblems are changed. In both cases, the solutions to be changed are replaced randomly.

Parameters

In Table 11.11 the functions used for creating a solution based on other solution and its associated parameters are provided. Parameter p is a real belonging to the interval $[0,1]$.

Table 11.11 Functions and possible values for parameters for creating a solution based on other solution

Alternative	Function	p
8	$perturbRandomly(s,p)$	$PARSCperturbintensityrand$
9	$perturbBasedRecencySols(s,p)$	$PARSCperturbintensityrecency$
10	$perturbBasedFrequencySols(s,p)$	$PARSCperturbintensityfrequency$

11.7.1.3 Path Relinking

Path relinking is a procedure that attempts to find a better solution in the path between two good quality solutions (a starting solution and a target solution) (see, for example, [63]). A solution, we name it path solution, is initialized to be equal to the starting solution. All the components of the path solution with a different value in the target solution are evaluated in a greedy manner. The best one is fixed in the

path solution and the procedure is repeated until all components of the path solution are fixed. In the case of the Searchcol, a component of a solution is a subproblem solution and the evaluation is based on the functions discussed in subsection 11.5.2. We represent the path relinking function by $pathrelinking(s,s')$. If parameter $PARSCpathrelinking$ is 1 path relinking is used, otherwise path relinking is not used. Note that, as in the case of the other algorithmic components of this subsubsection, it is up to the actual MH to define where and how path relinking is used.

11.7.2 SearchCol with Multi-start Local Search

In a MSLS algorithm, local search is applied from different initial solutions. In Figure 11.5, an algorithm for SearchCol with MSLS is displayed. The shaded blocks correspond to the MSLS and replace the search blocks in 11.4.

The local search can be performed by one of the four procedures described in 11.7.1.1.

As in SearchCol main cycle, we distinguish between two types of iterations in MSLS: iterations where the MSLS incumbent was improved in the previous iteration and the others. Parameters $PARMSLSinitialplus$ and $PARMSLSinitialminus$ define how the initial solution is obtained in those two types of iterations. Each of these parameters can take values corresponding to the alternative 1, 2, or 7 presented in subsubsection 11.6.3.5 (the ones involving randomness) or one of three alternatives presented in subsubsection 11.7.1.2. The rationale behind the use of two parameters is exploring intensification and diversification concepts with the available algorithmic components.

The stopping criterion of MSLS is a maximum number of iterations without improvement, represented by $PARSCMSLSmaxiterwithoutimprov$. At the end of each local search step, a path relinking between the local optimum and the incumbent solution is attempted (if $PARSCpathrelinking = 1$). This additional procedure is not represented in Figure 11.5 for clarity.

According to [48], multi-start methods can be classified based on three key elements: memory, randomization, and degree of rebuild (related to the number of solutions components that remain fixed from one iteration to the next start). MSLS based on the available algorithmic components of SearchCol allows the implementation of an actual algorithms in all the spectrum of each of these elements. Different settings of parameters for SearchCol with MSLS result in different algorithms. For example, GRASP with path relinking is obtained by setting $PARMSLSinitialplus$ and $PARMSLSinitialminus$ to alternative 7 presented in subsubsection 11.6.3.5. MSLS also can be seen as an adaptive memory programming technique [61]. In the main cycle of this unifying perspective for MHs, a solution is generated using data in memory, then this solution is improved and the memory is udpated.

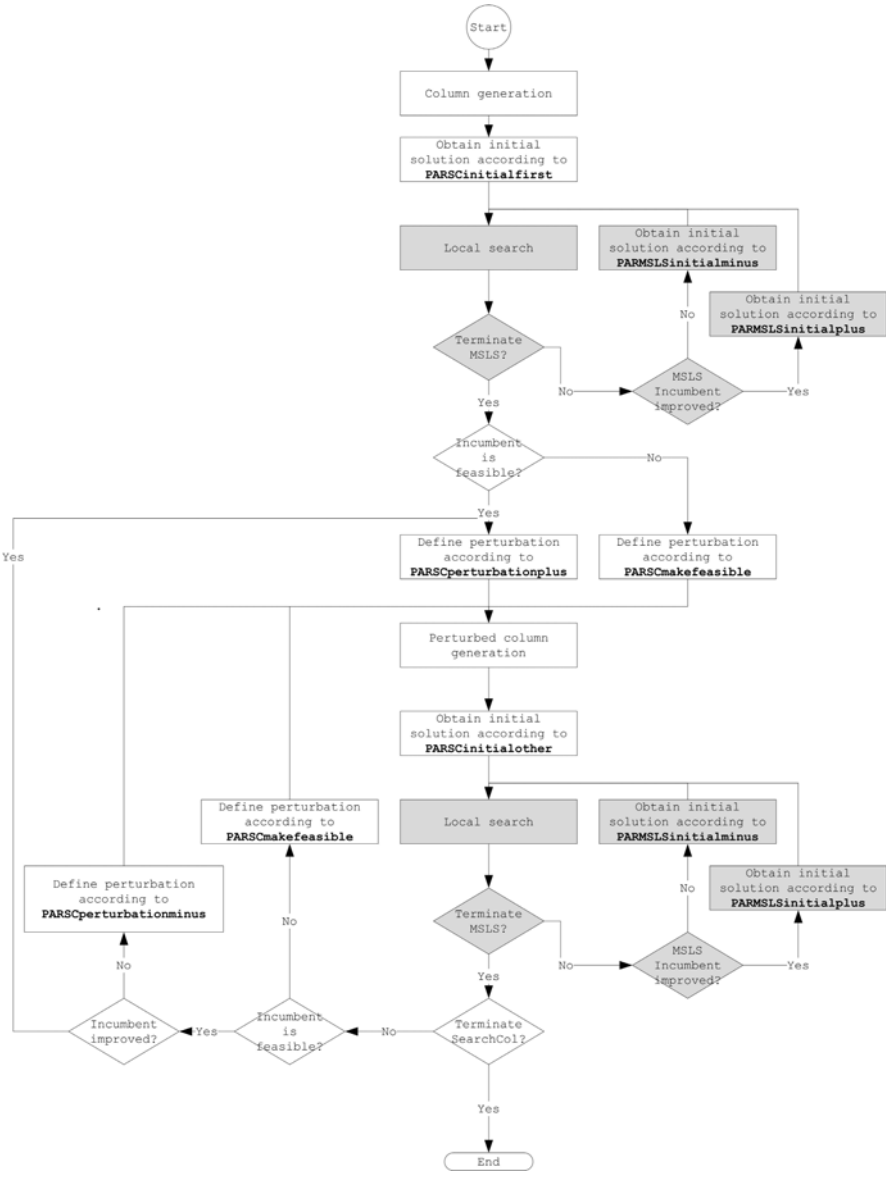

Fig. 11.5 Algorithm for SearchCol with multi-start local search

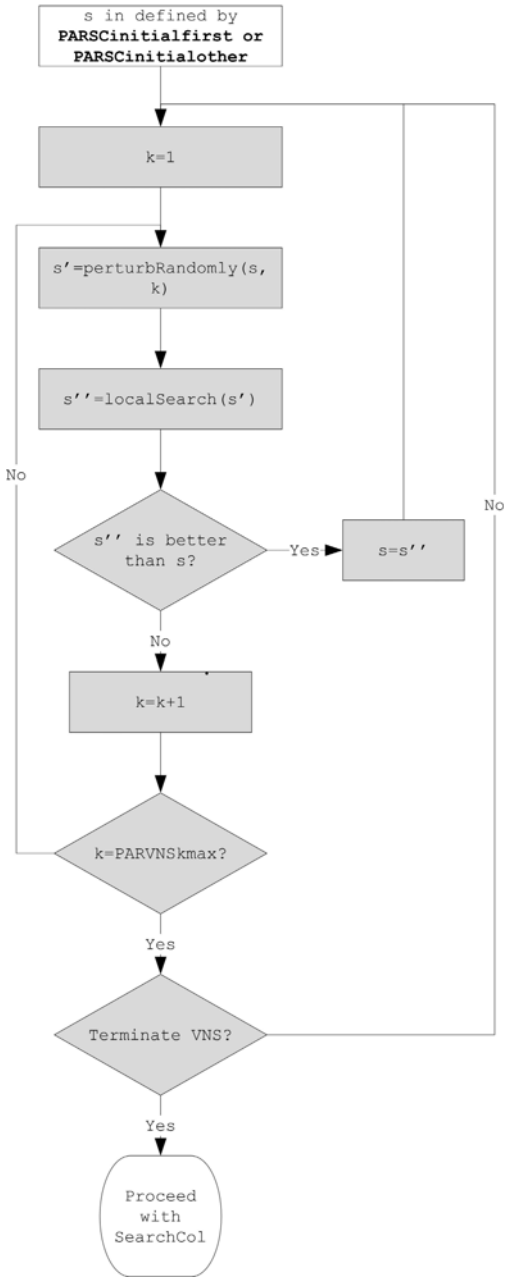

Fig. 11.6 Algorithm for SearchCol with variable neighborhood search

11.7.3 SearchCol with Variable Neighborhood Search

Variable neighborhood search (VNS) is a well known MH. For a recent description of VNS, variants, and applications see [36]. In Figure 11.6, an algorithm for VNS is displayed. In SearchCol with VNS, the search blocks in Figure 11.4 are replaced by the shaded blocks in Figure 11.6. In VNS algorithm, a hierarchy of neighborhoods are defined. For SearchCol with VNS, the hierarchy is given by k-neighborhoods defined in subsubsection 11.7.1.1. The current neigborhood is set to the first of the hierarchy ($k = 1$) and local search is applied from an initial solution (s') in the current neighborhood of the current solution (s). When the local optimum (s'') is better than the current solution (s) a new iteration starts from the $k = 1$ neighborhood, otherwise a more distant neighborhood becomes the current (by incrementing k). This cycle is repeated until k reaches parameter $PARVNSkmax$, ending an outer iteration. The stopping criterion is a maximum number of outer iterations without improvement of the VNS incumbent solution ($PARVNSiterwithoutimprov$).

11.7.4 SearchCol with MIP

For problems where a general-purpose MIP solver is efficient in solving restricted versions of (D), or the number and type of static variables make the evaluation of solutions too heavy for a MH approach, a MIP solver can be used in the search phase, i.e. in Figure 11.3 the search steps return the optimal (if a time limit set by a parameter $PARSCMIPmaxtimemip$ is not reached) solution of the problem with the available subproblem solutions.

11.8 Conclusions

In this Chapter, we discussed a framework for the combination of column generation (CG) and metaheuristics (MHs), named metaheuristic search by column generation (SearchCol). A deep collaboration of CG and MHs is achieved through representing a solution to the overall problem as a set of solutions from the subproblems of CG. Basically, CG provides the search space to the MH and, in turn, the MH provides information on desired or avoidable attributes of the subproblem solutions to be generated by CG.

The combinatorial perspective suggested allows to define core algorithmic components for different MHs. We proposed several alternatives for an actual SearchCol algorithm, based on concepts as randomization, greediness, memory, duality, and neighborhoods. We also detailed three examples of actual SearchCol algorithms: SearchCol with multi-start local search, SearchCol with VNS, and SearchCol with MIP.

Although the uselfulness of heuristics based on decomposition models, as the ones resulting from Lagrangean relaxation or the ones using CG, has been proved for several decades in specific problems, their combination in a more general setting seldom has been tried. In this Chapter, we discussed such a general framework for combining CG and metaheuristics and highlighted its potential.

Acknowledgements. This work have been partially funded by FCT (Fundação para a Ciência e a Tecnologia - Portugal) through project "SearchCol: Metaheuristic search by column generation" (PTDC/EIA-EIA/100645/2008) and through the post-doc grant SFRH/BPD/41581/2007 of D. Santos.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network flows: theory, algorithms, and applications. Prentice Hall, Englewood Cliffs (1993)
2. Alvelos, F., de Sousa, A., Santos, D.: SearchCol: Metaheuristic Search by Column Generation. In: Blesa, M.J., Blum, C., Raidl, G., Roli, A., Sampels, M. (eds.) HM 2010. LNCS, vol. 6373, pp. 190–205. Springer, Heidelberg (2010)
3. Alvelos, F., Valério de Carvalho, J.M.: Comparing branch-and-price algorithms for the unsplittable multicommodity flow problem. In: Ben-Ameur, W., Petrowski, A. (eds.) Proceedings of the International Network Optimization Conference, INOC 2003, Evry/Paris, pp. 7–12 (October 2003)
4. Alvelos, F., Valério de Carvalho, J.M.: A Local Search Heuristic based on Column Generation Applied to the Binary Multicommodity Flow Problem. In: Proceedings of International Network Optimization Conference, INOC 2007, Spa, Belgium, p. 6 (April 2007)
5. Akker, J.M., van den Hoogeveen, J.A., van de Velde, S.L.: Parallel machine scheduling by column generation. Operations Research 47, 862–872 (1999)
6. Akker, J.M., van den Hoogeveen, H., van de Velde, S.L.: Appplying column generation to machine scheduling. In: Desaulniers, G., Desrosiers, J., Solomon, M.M. (eds.) Column Generation, ch. 11, Springer (2005)
7. Ball, M.O.: Heuristics based on mathematical programming. Surveys in Operations Research and Management Science 16, 21–38 (2006)
8. Barnhart, C., Hane, C.A., Vance, P.H.: Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. Operations Research 48, 318–326 (2000)
9. Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P., Vance, P.H.: Branch-and-price: column generation for solving huge integer programs. Operations Research 46, 316–329 (1998)
10. Beasley, J.E.: Lagrangian relaxation. In: Reeves, C.R. (ed.) Modern Heuristic Techniques for Combinatorial Problems. John Wiley and Sons (1993)
11. Blum, C., Aguilera, M.J.B., Roli, A., Sampels, M. (eds.): Hybrid metaheuristics: An emerging approach to optimization. Springer (2008)
12. Blum, C., Cotta, C., Fernandez, A.J., Gallardo, J.E., Mastrolilli, M.: Hybridizations of metaheuristics with branch-and-bound derivatives. In: Blum, C., Aguilera, M.J.B., Roli, A., Sampels, M. (eds.) Hybrid Metaheuristics: An Emerging Approach to Optimization. Springer (2008)
13. Blum, C., Puchinger, J., Raidl, G.R., Roli, A.: Hybrid metaheuristics in combinatorial optimization: A survey. Applied Soft Computing 11, 4135–4151 (2011)

14. Blum, C., Roli, A.: Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. ACM Computing Surveys 35, 268–308 (2011)
15. Boschetti, M., Maniezzo, V., Roffilli, M.: Decomposition Techniques as Metaheuristic Frameworks. In: Maniezzo, V., Stützle, T., Voß, S. (eds.) Matheuristics - Hybridizing Metaheuristics and Mathematical Programming, Annals of Information Systems, vol. 10, ch. 5. Springer (2009)
16. Chen, Z.-L., Powell, W.B.: Solving Parallel Machine Scheduling Problems by Column Generation. INFORMS Journal on Computing 11, 78–94 (1999)
17. Dillenberger, C., Escudero, L.F., Wollensak, A., Zhang, W.: On Practical Resource Allocation for Production Planning and Scheduling with Period Overlapping Setups. European Journal of Operational Research 75, 275–286 (1994)
18. Danna, E., Pape, C.L.: Branch-and-Price Heuristics: A Case Study on the Vehicle Routing Problem with Time Windows. In: Desaulniers, G., Desrosiers, J., Solomon, M.M. (eds.) Column Generation, ch. 4. Springer (2005)
19. Dantzig, G.B., Wolfe, P.: Decomposition principle for linear programs. Operations Research 8, 101–111 (1960)
20. Desaulniers, G., Desrosiers, J., Solomon, M.M. (eds.): Column Generation. Springer, New York (2005)
21. Desrosiers, J., Soumis, F., Desrochers, M.: Routing with time windows by column generation. Networks 14, 545–565 (1984)
22. Desrosiers, J., Dumas, Y., Solomon, M.M., Soumis, F.: Time Constrained Routing and Scheduling. In: Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L. (eds.) Network Routing, Handbooks in OR & MS, vol. 8, ch. 2. Elsevier Science B.V. (1995)
23. Fahle, T., Junker, U., Karisch, S.E., Kohl, N., Sellmann, M., Vaaben, B.: Constraint Programming Based Column Generation for Crew Assignment. Journal of Heuristics 18, 59–81 (2002)
24. Fisher, M.L.: The Lagrangian relaxation method for solving integer programming problems. Management Science 27, 1–18 (1981)
25. Fisher, M.L.: The Lagrangian relaxation method for solving integer programming problems. Management Science 50, 1872–1874 (2004)
26. Fisher, M.L., Kedia, P.: Optimal solutions of set covering/partitioning problems using dual heuristics. Management Science 36, 674–688 (1990)
27. Ford, L.R., Fulkerson, D.R.: A suggested computation for maximal multicommodity network flows. Management Science 5, 97–101 (1958)
28. Frangioni, A.: About Lagrangian Methods in Integer Optimization. Annals of Operations Research 139, 163–193 (2005)
29. Gendreau, M., Potvin, J.-Y. (eds.): Handbook of metaheuristics. Springer (2010)
30. Geoffrion, A.M.: Lagrangean relaxation for integer programming. Mathematical Programming Study 2, 82–114 (1974)
31. Gilmore, P.C., Gomory, R.E.: A linear programming approach to the cutting stock problem. Operations Research 9, 849–859 (1961)
32. Gilmore, P.C., Gomory, R.E.: A linear programming approach to the cutting stock problem - Part II. Operations Research 11, 863–888 (1963)
33. Glover, F., Kochenberger, G. (eds.): Handbook of metaheuristics. Kluwer (2003)
34. Glover, F., Laguna, M.: Tabu Search. Kluwer (1997)
35. Gualandi, S., Malucelli, F.: Constraint programming-based column generation. A Quarterly Journal of Operations 7, 113–137 (2009)
36. Hansen, P., Mladenovic, N., Brimberg, J., Perez, J.A.M.: Variable neighborhood search. In: Gendreau, M., Potvin, J.-Y. (eds.) Handbook of Metaheuristics. Springer (2010)
37. Held, M., Karp, R.M.: The traveling-salesman problem and minimum spanning trees. Operations Research 18, 1138–1167 (1970)
38. Held, M., Karp, R.M.: The traveling-salesman problem and minimum spanning trees: Part II. Mathematical Programming 1, 6–25 (1971)
39. Hopp, W.J. (Editor-in-Chief): Ten Most Influential Titles of "Management Science's" First Fifty Years. Management Science 50 (2004)

40. IEEE Standard 802.1s: Virtual Bridged Local Area Networks - Amendment 3: Multiple Spanning Trees (2002)
41. Jünger, M., Liebling, T.M., Naddef, D., Nemhauser, G.L., Pulleyblank, W.R., Reinelt, G., Rinaldi, G., Wolsey, L.A. (eds.): 50 Years of Integer Programming 1958-2008, From the Early Years to the State-of-the-Art. Springer (2010)
42. Klabjan, D.: Large-scale models in the airline industry. In: Desaulniers, G., Desrosiers, J., Solomon, M.M. (eds.) Column Generation, ch. 3, Springer (2005)
43. Kallehauge, B., Larsen, J., Madsen, O.B.G.: Vehicle Routing with Time Windows. In: Desaulniers, G., Desrosiers, J., Solomon, M.M. (eds.) Column Generation, ch. 3. Springer (2005)
44. Kelley, J.E.: The cutting-plane method for solving convex programs. Journal of the SIAM 8, 703–712 (1960)
45. Lopes, M.J.P., Valério de Carvalho, J.M.: A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. European Journal of Operational Research 176, 1508–1527 (2007)
46. Lübbecke, M.E., Desrosiers, J.: Selected topics in column generation. Operations Research 53, 1007–1023 (2005)
47. Maniezzo, V., Stutzle, T., Voss, S. (eds.): Matheuristics, hybridizing metaheuristics and mathematical programming. Springer (2009)
48. Marti, R., Moreno-Vega, J.M., Duarte, A.: Advanced multi-start methods. In: Gendreau, M., Potvin, J.-Y. (eds.) Handbook of Metaheuristics. Springer (2010)
49. Martin, R.K.: Large Scale Linear and Integer Optimization, A Unified Approach. Kluwer Academic Publishers (1999)
50. Monaci, M., Paolo, T.: A Set-Covering-Based Heuristic Approach for Bin-Packing Problems. INFORMS Journal on Computing 18, 71–85 (2006)
51. Ow, P.S., Morton, T.E.: Filtered beam search in scheduling. International Journal of Production Research 26, 35–62 (1988)
52. Pisinger, D., Sigurd, M.: Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. INFORMS Journal on Computing 19, 1007–1023 (2007)
53. Puchinger, J., Raidl, G.R., Pirkwieser, S.: MetaBoosting: enhancing integer programming techniques by metaheuristics. In: Maniezzo, V., Stutzle, T., Voss, S. (eds.) Matheuristics, Hybridizing Metaheuristics and Mathematical Programming. Springer (2009)
54. Raidl, G.R.: A Unified View on Hybrid Metaheuristics. In: Almeida, F., Aguilera, M.J., Blum, C., Moreno Vega, J.M., Perez, M., Roli, A., Sampels, M. (eds.) Hybrid Metaheuristics. Springer (2006)
55. Raidl, G.R., Puchinger, J., Blum, C.: Metaheuristic Hybrids. In: Gendreau, M., Potvin, J.-Y. (eds.) Handbook of Metaheuristics. Springer (2010)
56. Raidl, G.R., Puchinger, J.: Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In: Blum, C., Aguilera, M.J.B., Roli, A., Sampels, M. (eds.) Hybrid Metaheuristics: An Emerging Approach to Optimization. Springer (2008)
57. Resende, M., Ribeiro, C.: Greedy randomized adaptive search procedures: advances, hybridizations, and applications. In: Gendreau, M., Potvin, J.-Y. (eds.) Handbook of Metaheuristics, 2nd edn., ch. 10. Springer (2010)
58. Santos, D., de Sousa, A., Alvelos, F.: Traffic Engineering of Telecommunication Networks Based on Multiple Spanning Tree Routing. In: Valadas, R., Salvador, P. (eds.) FITraMEn 2008. LNCS, vol. 5464, pp. 114–129. Springer, Heidelberg (2009)
59. Santos, D., Sousa, A.F., Alvelos, F., Dzida, M., Pióro, M.: Optimization of link load balancing in multiple spanning tree routing networks. Telecommunication Systems 48, 109–124 (2011)
60. Savelsbergh, M.: A branch-and-price algorithm for the generalized assignment problem. Operations Research 45, 831–841 (2007)

61. Taillard, E., Gambardella, L., Gendreau, M., Potvin, J.-Y.: Adaptive memory programming: A unified view of metaheuristics. European Journal of Operational Research 135, 1–16 (2001)
62. Talbi, E.-G.: A taxonomy of hybrid metaheuristics. Journal of Heuristics 8, 541–564 (2002)
63. Talbi, E.-G.: Metaheuristics. John Wiley and Sons (2009)
64. Vanderbeck, F.: Implementing Mixed Integer Column Generation. In: Desaulniers, G., Desrosiers, J., Solomon, M.M. (eds.) Column Generation, ch. 12, Springer (2005)
65. Wilhelm, W.E.: A technical review of column generation in integer programming. Optimization and Engineering 2, 159–200 (2001)
66. Wolsey, L.A.: Integer Programming. John Wiley and Sons (1998)