4OR
https://doi.org/10.1007/s10288-022-00510-8

INVITED SURVEY

Matheuristics: using mathematics for heuristic design

Marco Antonio Boschetti[1] · **Vittorio Maniezzo**[2]

Received: 15 December 2021 / Revised: 24 March 2022 / Accepted: 4 April 2022

Abstract
Matheuristics are heuristic algorithms based on mathematical tools such as the ones provided by mathematical programming, that are structurally general enough to be applied to different problems with little adaptations to their abstract structure. The result can be metaheuristic hybrids having components derived from the mathematical model of the problems of interest, but the mathematical techniques themselves can define general heuristic solution frameworks. In this paper, we focus our attention on mathematical programming and its contributions to developing effective heuristics. We briefly describe the mathematical tools available and then some matheuristic approaches, reporting some representative examples from the literature. We also take the opportunity to provide some ideas for possible future development.

Keywords Matheuristics · Mathematical programming · Heuristics

Mathematics Subject Classification 90-00 · 90-02 · 90C11

1 Introduction

The term "*matheuristic*" does not identify any new approach for solving optimization problems but it is a *label* very useful to promote an opportunity that is well-known in the literature since the early years of operations research. When a first international workshop was organized in Bertinoro, Italy, to discuss the use of mathematical tools for the design of heuristics (Maniezzo 2006), we needed a name to denote the specific focus of the event. We decided to use the term *matheuristic* because it is nice, it points

✉ Marco Antonio Boschetti
marco.boschetti@unibo.it

Vittorio Maniezzo
vittorio.maniezzo@unibo.it

[1] Department of Mathematics, University of Bologna, Bologna, Italy

[2] Department of Computer Science, University of Bologna, Bologna, Italy

Published online: 09 May 2022

very well to the focus on mathematics for developing heuristics, and Google returned at the time 0 found pages under that heading. Starting from that first event, many other conferences, sessions, special issues, etc., have been promoted under the name matheuristics and at the time of writing Google and Google Scholar return 72,400 and 5460 results for this term, respectively.[1] Clearly, in this survey we cannot give a full account of this wealth of contributions, but we will try to focus on some points we consider of particular interest for the readers of 4OR.

In our recent book (Maniezzo et al. 2021) we survey matheuristics from the viewpoint of metaheuristics, i.e., abstract, problem agnostic heuristic solving approaches. We provide both a description of how mathematics can be used within well-known metaheuristics, defining *metaheuristic hybrids* and we also show that some mathematical approaches, for example the decomposition methods, can be used to define *original matheuristics*. All algorithms in the book are proposed as general, problem-independent methods, but they are also detailed in a specific application, that is a common Generalized Assignment Problem (GAP) instance.

In this survey, we follow a different approach. We introduce the main mathematical tools and then we describe how to use them for designing heuristics, reporting some contributions in the literature that provide interesting insights. We focus our attention mainly on the use of Mathematical Programming (MP), even if it is not the only possible option.

We begin by giving a general overview of MP in Sect. 2, where we introduce a general mixed integer linear programming model, its LP-relaxation and the corresponding dual. We consider the use of dual variables and reduced costs in constructive heuristics and some general design frameworks, such as Kernel Search. We also showcase the use of mixed integer programming in heuristic algorithms, where some approximated problems or subproblems have been modeled and solved by different heuristic frameworks. For example, in many metaheuristic applications we can explore a *large neighborhood* or we need to obtain feasible integer solutions from some fractional solutions, these are both cases that we can model by a suitable mathematical model.

We describe some approaches for solving mixed integer problems, in particular the branch and bound in Sect. 3, where we report two examples of heuristic frameworks exploiting it, namely Beam Search and ANTS. In Sect. 4, we consider some variants of the branch and bound and we focus our attention on the branch and cut along with its use in diving heuristics and Corridor Methods. In Sect. 5, we consider the dynamic programming and we show how it could be used in Dynasearch and Fore and Back.

In Sect. 6, we discuss different heuristic algorithms based on decomposition methods and their potential for solving difficult and large-scale instances and for developing fully-distributed and parallel heuristic algorithms. In particular, we consider the Lagrangian, Dantzig–Wolfe, Benders' and surrogate relaxations and we provide some insights. For each approach, we move to describe matheuristics designed to use the corresponding approach to obtain high quality feasible solutions.

We describe how to apply mathematics in some components of well-known metaheuristics in Sect. 7, obtaining metaheuristic hybrids. We close the paper in Sect. 8 discussing some possible future research directions.

[1] Accessed: 19 August 2021.

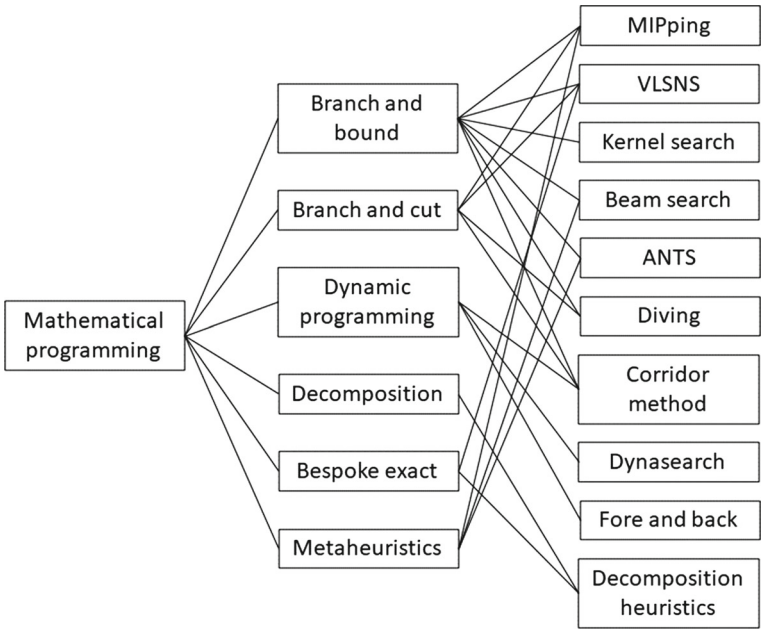

Fig. 1 MP solution methods and matheuristics

A comment is in order. For the need of structure, we listed each matheuristic after the MP method most commonly used as one of its modules. However, most matheuristics can be based on different types of solvers, Fig. 1 shows some of the most effective connections among MP methods, listed in the middle layer, and matheuristics listed in the third layer. The structure of the dependencies is rather messy, as several matheuristics have been implemented leveraging of different MP components, and indeed one can find in the literature other combinations, for example metaheuristics using diving or VLSNS, and other exist. We just draw here the connections most common in the literature, or those we review in this survey.

The topic of matheuristics has attracted significant interest since its proposal as a freestanding area of research, and it has already been surveyed several times, besides the textbook mentioned at the beginning (Maniezzo et al. 2021). The interested reader can find in the literature other surveys and special issues dedicated to Matheuristics (e.g., Boschetti et al. 2009b; Fischetti and Fischetti 2018; Maniezzo and Stützle 2020; Maniezzo et al. 2009).

2 Mathematical programming

Solving a real-world problem can be very challenging, and sometimes even finding a feasible solution can be very difficult. If we want to use some math for solving it, a first step requires defining a mathematical model and a possible option for this formalization is the use of mathematical programming. In this case, a model can have

the following general form:

$$\min\{f(\mathbf{x}) : \mathbf{x} \in X\} \tag{1}$$

where the *objective function* $f(\mathbf{x})$ can be a linear or non-linear function. In the case of multi-objective optimization, we have two or more functions. The *feasible region* X can be defined by linear or non-linear inequalities and some components of the solution $\mathbf{x}$ can be discrete (e.g., integer or binary).

A model *exactly* describes a real-world problem if we have $f(\mathbf{x}_1) < f(\mathbf{x}_2)$ when the solution $\mathbf{x}_1$ is better than $\mathbf{x}_2$ also in the real-world setting, and if the constraints allow all the real-world feasible solutions but forbid the unfeasible ones. Defining a mathematical model that exactly describes a real-world problem can be very challenging and, moreover, the resulting model can be too difficult to solve. In case, we can define an approximate model, accepting that the optimal solution found may not be optimal or feasible for the original problem, or we can consider the opportunity to heuristically solve the problem or to model only some subproblems within the heuristic algorithm.

We have a huge literature on effective mathematical tools for solving the model when the objective function and the inequalities defining the feasible region are linear functions. Moreover, in the last decades even for non-linear models we have an increasing set of mathematical tools available, in particular for the quadratic programming.

Focusing on linear programming, given a problem (1), where the objective function $f(\mathbf{x})$ and the inequalities defining X are linear, if variables $\mathbf{x}$ must be integers, the model becomes an *Integer Linear Programming* (ILP) model, and it can be written as follows:

$$z_{ILP} = \min \ \mathbf{cx} \tag{2}$$
$$\text{s.t.} \quad \mathbf{Ax} \geq \mathbf{b} \tag{3}$$
$$\mathbf{x} \geq \mathbf{0}, \ \text{integer} \tag{4}$$

where $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{A} \in \mathbb{R}^{mn}$, and $\mathbf{x} \in \mathbb{Z}^n$. If only some of the variables must be integers, we have a *mixed integer linear programming* (MILP) model. For the sake of simplicity, hereafter we only consider MILP models, since an ILP model is a special case where the set of continuous variables is empty.

If we relax the integrality constraints of a MILP model, we have its *LP-relaxation* that can be written as follows:

$$z_{LP} = \min \ \mathbf{cx} \tag{5}$$
$$\text{s.t.} \quad \mathbf{Ax} \geq \mathbf{b} \tag{6}$$
$$\mathbf{x} \geq \mathbf{0} \tag{7}$$

where $\mathbf{x} \in \mathbb{R}^n$. Its dual problem is:

$$z_D = \max \ \mathbf{wb} \tag{8}$$
$$\text{s.t.} \quad \mathbf{wA} \leq \mathbf{c} \tag{9}$$
$$\mathbf{w} \geq \mathbf{0} \tag{10}$$

where $\mathbf{w} \in \mathbb{R}^m$ are the dual variables corresponding to constraints (6).

The complementary slackness theorem states that an optimal LP variable x_j can be positive only if its *reduced cost* $c'_j = c_j - \sum_{i=1}^{m} a_{ij} w_i$ is equal to zero (see Bazaraa et al. 1990). This property can be used to drive heuristic algorithms to feasible near-optimal solution.

Useful tools for solving a MILP problem, both exactly and heuristically, are *branch and bound methods*, *dynamic programming*, and *decomposition methods*, possibly including *surrogate relaxation*, that are described in the next sections.

In this survey, we focus our attention on linear programming, and in this case a preliminary point we need to put forth is the increasing effectiveness of mixed-integer programming (MIP) solvers, i.e., the well-known software solutions that take in input the problem model in MIP format and calculate its solution.

2.1 Mipping

Modern MIP solvers include a wide variety of advanced techniques to attack hard problems, which include, for example, *strong branching*, i.e., the solution of LP models to control the branching strategy, *lift-and-project* for cut generation, reduced costs-based heuristics, among others. It has been ascertained that the solution of very hard MIPs can take advantage of the solution of a series of "collateral" linear programs, for example solving knapsacks to separate valid inequalities, whose solutions permit to guide the main steps of the MIP solver (Fischetti et al. 2009). Moreover, for easy MIPs, finding good-quality MIP solutions may require a computing time that is comparable to that needed to solve its LP relaxation. It is therefore computationally worthwhile to use these MIP models, instead of problem linear relaxations, to guide the MIP solvers in its most crucial steps, or anyway to solve to optimality MIP subproblems in order to speed up the solution of the whole problem we are interested in.

Fischetti et al. (2009) proposed to use the verb "*MIPping*" to denote the activity of translating into a MIP model some crucial decisions to be taken within a MIP algorithm. MIPping can be effective both for exact and for heuristic solutions. In this review, we are specifically interested in heuristic design, thus in the possible benefits deriving from the use of a MIP solver to produce heuristic primal solutions for a generic MIP. Several contributions have been proposed along this line, many of which intersect with other approaches presented in other sections of this review. This is in fact the case for local branching or RINS, which are paradigms that use the black-box MIP solver to explore large solution neighborhoods defined through the introduction in the MIP model of simple invalid linear inequalities. These approaches are described in Sect. 4.1. Here, mentioning just a few contributions that closely stick to the idea of including MIP models in order to construct large-scale neighborhoods that are effectively explored by a black-box MIP solver to obtain high-quality solutions, we point out the works of De Franceschi et al. (2006), Hewitt et al. (2010) and Salari et al. (2010).

In de Franceschi et al. an auxiliary ILP model is used while solving the Vehicle Routing Problem (VRP). The subproblem permits to determine how to optimally reallocate sequences of clients that will be part of the whole solution.

The work of Hewitt et al. on the fixed-charge network flow problem relies on neighborhood search. Here again, nonpolynomial problems arise with neighborhoods that require solving carefully chosen integer programs derived from the arc-based formulation of fixed cost network flow instances, which are subproblem of the problem of interest.

Salari et al. consider the open VRP and implement an optimized destruct-and-repair approach, where the current solution is randomly destroyed (i.e., customers are removed in a random way) and repaired by specifically defined ILP models.

2.2 Very large-scale neighborhood search

Very large-scale neighborhood search, or VLSNS (Ahuja et al. 1999, 2000, 2002; Maniezzo et al. 2021), intersects much the idea of MIPping, and in fact, it could even be considered as denoting the area of matheuristics as a whole. It is not an algorithm, but a conceptual framework to be used when trying to design methods for solving combinatorial optimization problems. It suggests to "*concentrate on neighborhood search algorithms where the size of the neighborhood is 'very large' with respect to the size of the input data*", typically, exponentially large. However, VLSNS can be mentioned each time an algorithm works on neighborhoods that are too large for exhaustive search. Clearly, when the search of the large neighborhood is made by a MIP solver, VLNSN reduces to MIPping. However, specialized algorithms can be used for solving the arising subproblems, and this can give VLSNS a distinctive denotation.

To help make the class less indistinct, Ahuja et al. (2002) proposed a categorization of VLSNS methods into three classes:

1. *variable-depth methods*, which implement only a heuristic partial search of exponentially large neighborhoods.
2. *network flow based improvement algorithms*, local search methods which use network flow techniques to identify improving neighbors.
3. local search based on neighborhoods defined over *subclasses or restrictions* of NP-hard problems, that are solvable in polynomial time.

This categorization, allowing for exceptions, is largely accepted in the literature to discriminate what is properly VLSNS and what is, more in general, a local search over a very large, possibly exponential, neighborhood. We note that algorithms pertaining to these three classes have been proposed even before the introduction of VLSNS (which is true also for matheuristics in general), but the classification helps to quickly understand the fundamental working components.

At its core, VLSNS is a paradigm that can be used in designing local search heuristics where the best neighbor of the incumbent solution can be found solving a specific combinatorial problem. This secondary problem, to be solved at each local search step, must be solved efficiently, thereby supporting a full exploration even of exponential neighborhoods. In VLSNS mathematical programming is used to define and explore neighborhoods. When possible, this boosts local search algorithms, which produce better solutions when they are allowed to explore large neighborhoods, but the exhaustive exploration of the whole of large neighborhoods can be very time consuming, thus the time to get its local optimum very long. VLSNS permits to leverage

mathematical programming results to achieve polynomial time explorations of exponential neighborhoods.

The paradigmatic example of heuristic exploration of NP-hard neighborhoods is based on the correspondence between improving cyclic exchange and negative cost subset-disjoint cycle in an improvement graph (Thompson and Psaraftis 1993). When we deal with sequences of exchanges based on paths instead of cycles, contributions overlap with those presented under other names, such as dynasearch (see Sect. 5.1) or ejection chains (see Sect. 7.1). Moreover, specific approaches were presented as large neighborhood search (Pisinger and Ropke 2010) and later generalized by allowing multiple destroy and repair operators, to obtain adaptive large neighborhood search, ALNS (Ropke and Pisinger 2006). Anyway, the idea of path- or cycle-based exchange neighborhoods has been applied to very different problems, including the vehicle routing problem (Thompson and Psaraftis 1993), the minimum makespan machine scheduling problem (Gendreau et al. 2006), the graph coloring problem (Chiarandini et al. 2008), and timetabling problems (Meyers and Orlin 2006), among others.

An example of efficiently solvable subproblems was presented by Ahuja et al. (2002) using Halin graphs, but the idea was used even before that, for example for a matching neighborhood for the Traveling Salesman Problem (TSP) (Sarvanov and Doroshko 1981), and later to several other problem areas, such as scheduling (Brueggemann and Hurink 2007, 2011) or generalized assignment (Mitrović-Minić and Punnen 2008, 2009).

Application-oriented works can be found for the ready-mixed concrete delivery problems (Schmid et al. 2010), and for the founder sequence reconstruction problem (Roli et al. 2012).

2.3 Kernel search

Kernel search (KS) was introduced in Angelelli et al. (2007), and then extended in Angelelli et al. (2010), as a heuristic method leveraging on LP-relaxation, duality and reduced costs.

KS is a matheuristic approach, which makes use of MILP solvers to obtain heuristic, possibly optimal, solutions of instances encoded as (mixed) integer linear programming problems. KS was in fact first presented as a method to solve MILP problems defined on binary variables that modeled items selection. The binary variables were possibly mixed with other integer or continuous variables related to the selected items. Later contributions extended the method to the possibility to effectively deal with other problems that do not involve a selection stage.

The central idea of KS is the use of some method, typically an LP relaxation, to identify a subset (a *kernel*) of promising decision variables and then to partition the remaining ones into *buckets*. The buckets are concatenated one at a time to the kernel to check whether improving solutions can be found, which include some of the bucket variables. To this end, KS goes through two phases: initialization and expansion.

In the *initialization phase*, the MILP formulation $\mathcal{F}$ of the instance to solve is first used to identify a promising subset of variables, which could enter the optimal solution. These variables are selected as the initial kernel. One way to identify them could be

by means of the LP-relaxation of $\mathcal{F}$ and the corresponding reduced costs, but other methods could be used as well. The variables not in the kernel are partitioned into subsets, the buckets. The best-found solution is initialized by calling a MILP solver on the kernel subset, possibly allowing it restricted computational resources (time, memory, or whatever), in the hope of finding for this easier instance a feasible primal solution.

The *expansion phase* follows, where a sequence of MILP subproblems is solved. Each i-th subproblem is restricted to a subset of the problem variables, which includes the current kernel and a successive bucket. The subproblems are further constrained to include in the solution at least one bucket variable and to provide a solution of cost better than that of the so far best found solution. In case such a solution is found, it becomes the new best found one and its non zero variables are included in the kernel. The procedure is iterated and terminates when no kernel expansions can be identified.

Applications of the method include portfolio selection (Angelelli et al. 2007, 2012), multidimensional knapsack (Angelelli et al. 2010), but also financial index tracking (Guastaroba and Speranza 2012), time-dependent rural postman optimization (Zanotti et al. 2019), and supply chain optimization (Zhang et al. 2019).

3 Branch and bound

There is surely no need to remind 4OR readers what a branch and bound approach is, however, we include this introduction in order to define the notation that will be used in the sections presenting relevant matheuristics.

A typical *branch and bound* implementation solves at the root node of the tree search a relaxation, for example, the LP-relaxation, of the original problem LP_0 obtaining a bound and, if its optimal solution is infeasible, for example being *fractional* (i.e., in the solution at least one integer variable is fractional), it generates a number of *branches* LP_i (i.e., child nodes of the current node), each one exploring a subset of the search-space and excluding the current infeasible solution. According to the specific *search strategy* the branch and bound selects one of the *unexplored* (i.e., not solved yet) problem LP_i and if the solution is still infeasible it generates further branches. During this process, all nodes having a lower bound (for a minimization problem) greater or equal to the current best upper bound (i.e., the value of the best feasible solution found) can be eliminated, reducing the search space.

3.1 Beam search

A matheuristic approach specifically targeting branch and bound is Beam search (BS), whose central idea can be traced back to Lowerre (1976). BS is a variant of standard tree search that limits the number of offspring that are expanded at each iteration. BS core ideas were originally introduced in artificial intelligence contexts, and only later transposed to optimization. The first problems for which BS was used were scheduling problems, but BS has since proved successful also on many other different combinatorial optimization problems.

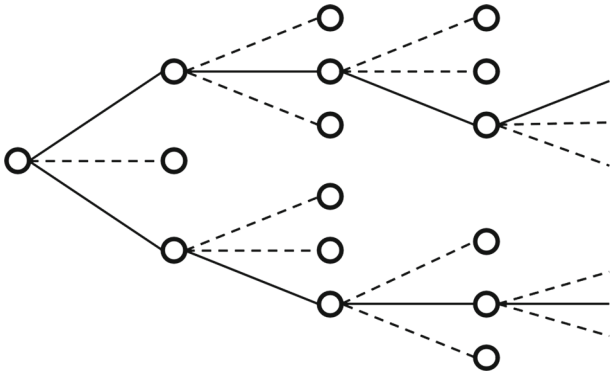

Fig. 2 Beam search, $\delta = 2$

BS does not complete the search that would normally be carried out by branch and bound algorithms, therefore it is an approximate method and a matheuristic of its own. BS has, in fact, been proposed as an effective heuristic methodology, and as such it has been enhanced and hybridized with other heuristics, for example, with ant colony optimization (see Sect. 7.2).

Moreover, other matheuristics closely related to BS have been proposed. One is the pilot method (Duin and Voß 1999), which consists of a partial enumeration strategy, where the possible expansions of each partial solution are evaluated by means of a pilot heuristic. Another one is the Filter&Fan method (Greistorfer and Rego 2006), which starts with a feasible solution and builds a search tree, where branches correspond to submoves in the neighborhood of the solution and where each node corresponds to a solution obtained as a result of the sequence of submoves associated with the root-node path. In this algorithm, the initial candidate list of moves is filtered at each tree level by evaluating each move in the list with respect to all the solutions at that level. The best moves at each level are included in the candidate list of the next level and the corresponding solutions are the nodes of the successive level.

The characterizing idea of BS is to allow the extension of partial solutions into a limited number of offsprings. This is similar to other approaches reported in this review, such as VLSNS (Sect. 2.2), Diving (Sect. 4.1) or the Corridor Method (Sect. 4.2), but in the case of BS the focus is on the result, the number of offsprings, and not on the method to limit their number. At each BS iteration, the algorithm extends a partial solution from a set $\mathbf{T}$, the *beam*, generating a possibly limited number of offsprings. Each offspring is either a complete solution, or it is inserted into the set $\mathbf{T}$ itself, in case it is a partial solution worth further analysis.

At the end of the expansions, BS selects from $\mathbf{T}$ up to δ (a parameter called the *beam width*) solutions. The selection is based on some criterion for ranking the expected usefulness of an expansion, for example, based on bounds to the cost of the completions. Fig. 2 shows a part of a possible beam search expansion tree. At each level, each active node generates all of its offsprings (3 for each node, in the figure), then only δ of them are selected and allowed to expand to the next search level. The parameter δ is the beam width (2 in the figure).

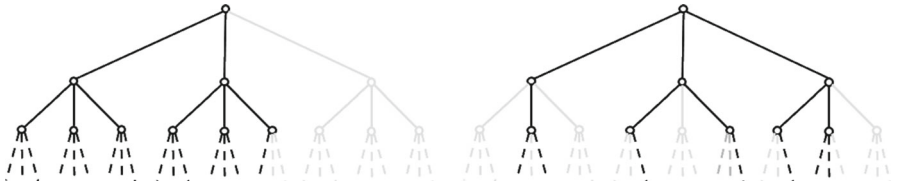

Fig. 3 Complete (left) and approximate (right) tree search

BS was applied for example to scheduling problems, both in a straightforward way (Ow and Morton 1988) and extended with some additional features, applied to the two-machine total completion time flow shop scheduling problem in Della Croce et al. (2004) and to open shop scheduling and to simple assembly line balancing in Blum (2005) and Blum (2008), respectively.

3.2 ANTS

Another matheuristic based on branch and bound is ANTS (Maniezzo 1999). It could be framed in Sect. 7 as it is a variant of Ant Colony Optimization, a well-known metaheuristic, but since it leverages on the analogy of a general ACO approach with branch and bound, it fits best here.

ANTS is in fact an acronym for *Approximate Nondeterministic Tree Search*, and it actually represents one of the first algorithms proposed in the literature that included MP elements in a metaheuristic structure. The general approach is here still a constructive one, where solutions are stepwise constructed, at each step computing a bound to the cost of the best feasible solution that can be obtained upon completing the incumbent partial solution. Backtracking occurs when a complete solution is reached or, possibly, when no feasible improving solution can be identified.

A few major differences exist with respect to standard branch and bound; the main one being that search is not allowed to backtrack to partial solutions but it always restarts from scratch, thus no stack data structure is needed and search can be run in parallel. However, a global data structure (the *trail matrix*) is updated after each solution completion, quantifying the correlation between variable assignments and quality of the solution eventually obtained with those assignments. Node expansion is then determined in probability on the basis both of the bound and of this trail data.

It derives that tree search is not exact but approximated and that it is non deterministic, given the random choice of the offspring, thus the name ANTS. Figure 3 depicts the two search strategies.

A few further elements related to MP were included in ANTS. One is the suggestion to initialize the trail matrix not randomly or using an ad-hoc user parameter, as in other ACO codes, but by the primal values of the decision variables, as appearing in the optimal bound solution (if the bound is computed by means of linear programming). Another is the possibility of pruning expansion branches, again if the bound is computed by means of linear programming, using the reduced costs of the decision variables, which in turn permit to a priori eliminate some variables in the node

expansion. This results in a reduction of the number of possible moves, therefore in a reduction of the search space.

Other details are more ACO-specific, thus of limited interest for this review. They include dynamic fitness scaling to reduce the risk of search stagnation and to promote fine-tuning in the late search stages and a simplified probability distribution function to be used in node expansion.

As a final remark, we note that ANTS was proposed with two alternative branching strategies. The first one is the depth-first, where the node expanded at each level is the offspring of the incumbent one having the least cost lower bound. The second strategy is Beam Search alike, where a number of nodes are expanded at the same level before stepping deeper into the search tree. This second strategy has been thoroughly investigated in another ACO variant named Beam-ACO (Blum 2005), which we already cited in the section dedicated to beam search.

4 Branch and cut

Branch and bound is not usually implemented in its simplest form, but it is integrated by enhancing elements. Two well-known variants of the branch and bound are the *branch-and-cut* and the *branch-and-price* (Wolsey 2020). The branch-and-cut tries to add at each node LP_i some *valid inequalities*, which are inequalities redundant in the original MILP model but violated by some fractional solutions, possibly increasing the value of the lower bound. The branch-and-price considers only a subset of the original variables and at each node of the tree search it adds some new variables that have the potential to be in the optimal solution. These two approaches can be combined into a *branch-and-price-and-cut* (Wolsey 2020). All these possibilities have given rise to matheuristics.

4.1 Diving heuristics

A family of heuristics is known by the name of Diving Heuristics (Bixby et al. 2000). These are methods that progressively complete a partial solution up to its possible feasibility; they can be seen as diving into a solution without the possibility of backtracking. This working is common to all constructive heuristics, what is distinctive of diving heuristics is that they are characterized by working on the mathematical formulation of the problem to solve, typically adding cuts in order to converge to feasibility. Some of these heuristics proved to be remarkably effective, and are included as standard components of general MIP solvers. Two well-known diving heuristics are relaxation induced neighborhood search (RINS) (Danna et al. 2005) and local branching (Fischetti and Lodi 2003).

Both methods are self-sufficient, complete heuristics, but they came to be used as elective algorithms to be applied at some branch and bound nodes in order to improve the incumbent feasible partial solution. They can be applied to a generic MIP problem of the form

$$z_{\text{MIP}} = \min \sum_{j \in J} c_j x_j \tag{11}$$

$$\text{s.t.} \sum_{j \in J} a_{ij} x_j \leq b_i, \qquad i \in I \tag{12}$$

$$x_j \in \{0, 1\}, \qquad j \in B, B \neq \emptyset \tag{13}$$

$$x_j \geq 0, \text{ integer}, \qquad j \in G \tag{14}$$

where the index set of the decision variables, $J = B \cup G$, is composed of B, the index set of binary decision variables, and of G, the possibly empty index set of general integer nonnegative variables.

During search, while expanding a branch tree, two solutions are compared: the *incumbent* feasible solution (if one was found) $\mathbf{x}^h$ and the *bound* solution at the current node $\tilde{\mathbf{x}}$. The incumbent solution is a heuristic feasible solution for the MIP, that is not guaranteed to be optimal. The bound solution is usually the solution of the continuous relaxation of the MIP. Typically, the two solutions will have some variables that take the same values, while other variables differ. RINS is a technique that tries to force the two solutions to agree on all variables, while Local Branching is a local search heuristic exploring a neighborhood of $\mathbf{x}^h$.

RINS relies on the assumption that the values taken by variables in a good or optimal solution are often in common with the values taken by the same variables in good lower bound solutions. It follows that selecting an appropriate set of values for the bound variables and completing that assignment appears to be a promising approach. The selection is made by fixing all variables that have the same values in the bound and in the incumbent solutions, and letting the solver try to solve optimally the remaining MIP problem, that is called *sub-MIP*, within a given node limit or with an objective cutoff.

One advantage of RINS, when applied within a branch-and-cut procedure, is that the continuous relaxation changes at every node of the tree, and this directly implements a diversification of the starting points for the completions. However, since RINS could be computationally demanding and since bounds of related nodes typically do not change by much, it is convenient to apply this procedure only periodically, after a given number of new nodes are explored.

The sub-MIP can be quite large if too few variables were fixed, so its solution could take a time comparable with that of the original problem. The issue of the complexity of the heuristic procedure is faced by setting a limit on the computational resources available for optimization, usually in the form of a limit to the number of nodes that can be expanded during the search. If a solution is found, it may become the new incumbent feasible solution, otherwise, nothing happens.

Local Branching is similar to RINS, in that it starts with an incumbent feasible solution $\mathbf{x}^h$ and defines neighborhoods specifying which variables to fix in further exploration, but it does so directly addressing the issue of how many variables are to be fixed by explicitly dictating their number at each iteration.

To understand how the local branching works, we consider two feasible solution vectors $\mathbf{x}^h$ and $\mathbf{x}$ for problem MIP, where the variable subset only contains binary variables (but this request can be partially lifted), i.e., $J = B$ and $G = \emptyset$. Solution $\mathbf{x}^h$

is the incumbent feasible solution and $\mathbf{x}$ will be a neighboring feasible solution. Their Hamming distance is $\Delta(\mathbf{x}^h, \mathbf{x}) = |\{j \in B : |x_j^h - x_j| = 1\}|$, and the *binary support* of $\mathbf{x}^h$, i.e., the subset of binary variables which take the value of 1 in the reference solution is denoted by $S^h = \{j \in B : x_j^h = 1\}$.

Local Branching defines a limited neighborhood of $\mathbf{x}^h$ that will be explored by the sub-MIP, consisting only of the solutions satisfying the additional constraint $\Delta(\mathbf{x}^h, \mathbf{x}) \leq k$, where k is a radius parameter. Analogously to RINS, the local branching sub-MIP includes all cutting planes and variable bounds deriving from valid inequalities found during the exploration of the global branch-and-cut tree and ignores variable bounds imposed by branching, which are valid only on a subtree.

The limit on the size of the neighborhood is enforced by adding to the formulation so-called *local branching constraints*. Given the value of the parameter k, the k-opt neighborhood $N(\mathbf{x}^h, k)$ of $\mathbf{x}^h$ of an incumbent solution is defined as the set of feasible solutions of the original MIP satisfying the additional local branching constraint:

$$\Delta(\mathbf{x}^h, \mathbf{x}) = \sum_{j \in S^h} (1 - x_j) + \sum_{j \in B \setminus S^h} x_j \leq k \tag{15}$$

where the two sums count the number of variables changing their value from 1 to 0 and from 0 to 1, with respect to $\mathbf{x}^h$.

4.2 Corridor method

The Corridor method (CM) is a general search method originally proposed by Sniedovich and Voß (2006) as a dynamic programming (DP) heuristic overlay, and later extended beyond DP to other exact approaches, such as branch and bound.

In its general form, CM tries to solve a possibly NP-hard optimization problem, for which we know an exact method (branch and bound, branch and cut, dynamic programming or else) that could effectively solve it on relatively small instances. However, instances of interest are too big to ensure the possibility of getting an optimal solution within an acceptable time, and the direct application of the exact method becomes impractical. CM, therefore, tries to use the exact method over successive restricted portions of the solution space of the given problem. The restriction is obtained by applying exogenous constraints, for example in the form of cuts, which define local neighborhoods around points of interest.

The constraints often result in neighborhoods that are exponentially large, but that are structured in such a way that the chosen exact method can efficiently solve the restricted sub-instances. The name "corridor" for the method comes from its first application, which made use of DP as the exact module. In this setting, the constraints were used to control the state trajectory followed by DP using search. The trajectory was forced not to change too much from its past path, thus it was constrained in its progression as when walking along a corridor. This initial reference to DP was then lifted, permitting the use of other exact techniques at the core, for example MIP, where the corridor is defined around incumbent solutions, and the solver is forced to move along a trajectory connecting successive sub-MIP solutions. In this case, the CM

represents a further variation of the idea of solving to optimality a possibly exponential neighborhood of the incumbent solution.

The execution depends on a control parameter, δ_{max}, which specifies the maximum "width of the corridor", i.e., the maximum size of the subproblems passed on to the exact method.

A further commonly accepted feature is the use of a dynamic corridor width (Caserta et al. 2010; Caserta and Voß 2014). It is in fact possible to adapt the width of the corridor following the presence or not of improving solutions in the current neighborhood. If an improving solution is found in a small neighborhood, the incumbent solution is updated and a new corridor is defined around this new solution. Otherwise, the width of the corridor is widened, in the hope of helping to find feasible solutions.

The first application where CM was tested was the block relocation problem (Caserta and Voß 2009a, b; Caserta et al. 2011), and specifically on applications requesting to stack container terminals in a yard. In this problem, an initial collection of stacks of blocks is given, for example, stacks of containers in a port terminal. Moreover, a pickup list of the next containers to collect is known. The containers (blocks) have to be picked up following the given sequence. For each block to be picked, if there are other blocks above it, the pickup operation requires to relocate the overlapping blocks into other stacks. The same problem arises in the management of block stacking warehouses, where items—usually pallets—are simply stacked on top of one another, with no supporting infrastructure. Here stacks are divided into successive substacks where only the topmost block of each first substack can be accessed. Each floor strip identifies a stack. The blocks relocation problem requires to find the relocation sequence for each pickup operation so that the number of future relocation moves for accessing blocks of interest is minimized.

A problem closely related to block relocation is warehouse pre-marshalling, where we have a block-stacking warehouse, but we are asked to sort the initial configuration using a minimum number of relocations so that as few as possible new relocations will be needed when blocks will have to be picked. This is in contrast to the block relocation problem, where the objective was to retrieve the blocks according to the picking list and using a minimum number of relocations. The approach was later extended, including a statistical estimator that can account for uncertainties in the picking lists that will be received after pre-marshalling (Maniezzo et al. 2020).

Another successful application of the CM was about DNA sequencing (Caserta and Voß 2014). The problem asked to find the order in which sequences of nucleotides appear in an unknown fragment of the DNA. The problem has some similarities with the TSP; following this, the authors propose to model it as an Orienteering Problem and proceed to solve it by CM. A similar approach was used also for solving the capacitated lot-sizing problem (Caserta et al. 2010).

5 Dynamic programming

Dynamic Programming (DP) is a well-known method for solving integer linear problems that is based on the Bellman's *Principle of Optimality*, which states that "*an optimal policy has the property that whatever the initial state and initial decisions*

are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision" (Bellman 1957).

Following this principle, DP splits the process of constructing the solution into an ordered set of *stages* and at each stage it enumerates all the possible *states*. A DP algorithm considers a stage at a time in the given order and for each state of the current stage it is able to evaluate the best move from a state of the immediately previous stage, applying one of the possible decisions available at the current stage.

For example, consider the classical 0–1 knapsack problem, where we must choose among n items, having weight w_i and value p_i, the ones that maximize the sum of their values and have the sum of their weights smaller or equal to the knapsack capacity W, i.e. $z_{KP} = \max_{\mathbf{x}\in\{0,1\}^n}\{\sum_{i=1}^{n} p_i x_i : \sum_{i=1}^{n} w_i x_i \leq W\}$.

The problem solved at a stage j and state w considers only the first j items and a knapsack of capacity w, that is $z_{KP}(j, w) = \max_{\mathbf{x}\in\{0,1\}^j}\{\sum_{i=1}^{j} p_i x_i : \sum_{i=1}^{j} w_i x_i \leq w\}$, and the corresponding recursion is:

$$z_{KP}(j, w) = \max\{z_{KP}(j-1, w), z_{KP}(j-1, w-w_j) + p_j\} \tag{16}$$

Notice that, for each state w of the current stage j, we consider the only two possible decisions: do not include or include the item j. It is straightforward that $z_{KP}(0, w) = 0$, for every $0 \leq w \leq W$, and that we cannot include the item j if $w < w_j$, i.e., $z_{KP}(j, w) = z_{KP}(j-1, w)$.

DP can be applied to more complex problems and in these cases each state can have a more complex structure (e.g., set, sequence, etc.) and also the possible decisions can be more complex. For example, in a DP for the Traveling Salesman Problem (TSP) the stage j could correspond to the length of the Hamiltonian path and the states are defined by the possible subsets of clients and the last client visited. The decision is concerned with the next client to include in the path.

DP algorithms can be also used to generate columns for a model in exact methods, but also in heuristic algorithms. Moreover, DP algorithms can be used to compute bounds on the value of the optimal solution and in these cases some approximation can be also applied as, for example, the *state-space relaxation* (Christofides et al. 1981).

As it was the case for generic MIP, also DP has been used for getting heuristic solutions. Here we outline two DP-based matheuristics, Dynasearch and Fore-and-Back.

5.1 Dynasearch

Dynasearch (Congram et al. 2002) makes use of DP as a large neighborhood exploration enabler, thus it could have been equally well listed under VLSNS. The idea is to improve a current solution by combining simple search steps, thereby defining complex neighborhoods. The basic simple steps, in order to be viable for dynasearch, need to be mutually independent, which means that they must not have any combined impact, neither on the cost function nor on the feasibility of candidate solutions. This

independence guarantees that the overall effect of a combined step can be computed as the sum of the effects of the single composing steps.

Given the combined steps, dynasearch makes use of DP to organize search. The general dynasearch recursion is usually presented for permutation problems, where the independence of the search steps is guaranteed by the constraint that the moves operate on disjunct subsets of the permutation indices.

The recursion equation can be presented with reference to a permutation problem trying to find the permutation $\Pi = (\pi(j), j = 1, \ldots, n)$ that minimizes a given cost function. The recursion equation is defined assuming that the maximum cost reduction, considering moves modifying the solution up to position j, can be obtained by selecting the maximum value computed either by keeping the assignment at position j or by changing it with a combination of the best assignment up to a position $i < j$ with the best move sequence from position i up to position j, for each $j \in J$.

Dynasearch has been applied to different optimization problems, mostly scheduling problems that involve search in a space of permutations, for example the earliness-tardiness scheduling problem (Sourd 2006), the single machine total weighted tardiness problem (Congram et al. 2002), or the dynamic berth allocation of container ships (Nishi et al. 2020). However, also non scheduling problems have been attacked using dynasearch, including vehicle routing problems (Ergun et al. 2006) and generalized knapsack problems (Cunha and Ahuja 2005).

5.2 Fore and back

Fore-and-Back, also referred to as *Forward and Backward* or *F&B* (Bartolini et al. 2008; Bartolini and Mingozzi 2009), can be seen as an extension of Beam Search (BS, see Sect. 3.1) that can boost its effectiveness when the problem is adequately structured. Similarly to BS, when Fore-and-Back is run with no limits on computational resources, it becomes an exact solution method. However, by design, it is mainly concerned with heuristic solving, trying to quickly get high quality solutions.

Fore-and-back is a primal only method, but it is able to compute bounds to the cost of completing the partial solutions that are iteratively constructed, and it is therefore able to discard partial solutions from expansion thereby pruning the search trees. It can do this, because it alternates BS-like searches in opposite expansion directions, each time storing into memory partial results and their costs. These can be used as a lookahead to complete partial solutions when search is performed in the opposite direction. The algorithm works therefore best when the problem suggests a natural direction of partial solution expansions, which can also be reversed.

It has been observed that there is a significant subset of combinatorial optimization problems that can be optimized by fore-and-back. These are problems that exhibit a regular substructure that can be decomposed into n subproblems that are linked together by a set of coupling constraints. For example, in the case of the TSP, the subproblems can refer to the node to visit in the k-th position.

Fore-and-Back exploits this structure, by means of an iterative heuristic algorithm, that adopts a memory-based look-ahead strategy that exploits the knowledge gained in its past search history. In detail, it iterates a partial exploration of the solution space

by generating a sequence of beam search-like trees of two types, called *forward* and *backward* trees. Each node at level h of a tree represents a partial solution, made of h components. At each iteration t, the algorithm generates a forward tree $\mathbf{F}^t$ if t is odd, or a backward tree $\mathbf{B}^t$ if t is even. In generating the tree, the partial solution is extended to a feasible solution using the partial solutions generated at the previous iteration in the complementary tree, and the cost of the resulting solution is used to bound the quality of the best complete solution that can be obtained.

This is, for example, the case for the generalized assignment problem, where subproblems could refer to the assignments of single clients and the capacity constraints act as linking constraints, or vice-versa (subproblems defined on capacities and linking constraints on assignments, see Maniezzo et al. 2021).

6 Decomposition methods

Decomposition methods allow us to break up a difficult problem into smaller and easier subproblems, that can be solved separately, and to get the overall solution "recomposing" the individual solutions of each subproblem. Decompositions have a long history in optimization, and they come in many different flavors, ranging from constraint programming to logical decomposition, from dynamic programming to linear decompositions, among many others.

In this section, we overview how three well-known and intertwined decomposition methods, namely Lagrangian, Dantzig–Wolfe, and Benders decompositions, have been used as seeds for classes of heuristic algorithms, plainly to be included among matheuristics.

A good introduction to the topic of decompositions in mathematical programming is in Bazaraa et al. (1990), while the close relationship among decomposition methods is outlined in Boschetti and Maniezzo (2009a). Reviews of literature presenting decomposition-based matheuristics can also be found in Boschetti et al. (2009a) and Raidl (2015).

In the literature, decomposition methods are mainly applied to continuous, mixed-integer, and pure integer linear programming problems with suitable specific structures. In particular, decompositions of a mathematical formulation can be trivially obtained when the constraint matrix is block-separable. More interesting cases arise when blocks can be identified in the constraint matrix, but they are linked by some constraints (*linking constraints*) or by some variables (*linking variables*). We will concentrate only on some specific cases, as the interested reader can anyway refer to the abundant literature on general decomposition in optimization.

A simple example of a structure of a linear problem having linking variables suitable for effective decomposition can be the following problem P:

$$z_P = \min \ \mathbf{c}_1\mathbf{x}_1 + \mathbf{c}_2\mathbf{x}_2 + \mathbf{c}_3\mathbf{y} \tag{17}$$

$$s.t. \ \ \mathbf{A}_1\mathbf{x}_1 + \mathbf{B}_1\mathbf{y} \geq \mathbf{b}_1 \tag{18}$$

$$\mathbf{A}_2\mathbf{x}_2 + \mathbf{B}_2\mathbf{y} \geq \mathbf{b}_2 \tag{19}$$

$$\mathbf{x}_1, \quad \mathbf{x}_2, \quad \mathbf{y} \geq \mathbf{0} \tag{20}$$

In this case, if we fix the variables $\mathbf{y}$ to some values $\bar{\mathbf{y}}$, problem P becomes block separable in the variables $\mathbf{x}_1$ and $\mathbf{x}_2$ and could be split into two subproblems. Hence, we can solve the problem $z_{MP} = \min\{z_{MP}(\mathbf{y}) : \mathbf{y} \geq \mathbf{0}\}$, where for a given $\bar{\mathbf{y}}$ we evaluate $z_{MP}(\bar{\mathbf{y}}) = z_{SP1}(\bar{\mathbf{y}}) + z_{SP2}(\bar{\mathbf{y}}) + \mathbf{c}_3\bar{\mathbf{y}}$ by solving the following two subproblems independently:

$$\begin{aligned} z_{SP1}(\bar{\mathbf{y}}) = \min\ & \mathbf{c}_1\mathbf{x}_1 \\ \text{s.t. } & \mathbf{A}_1\mathbf{x}_1 \geq \mathbf{b}_1 - \mathbf{B}_1\bar{\mathbf{y}} \\ & \mathbf{x}_1 \geq \mathbf{0} \end{aligned} \qquad \begin{aligned} z_{SP2}(\bar{\mathbf{y}}) = \min\ & \mathbf{c}_2\mathbf{x}_2 \\ \text{s.t. } & \mathbf{A}_2\mathbf{x}_2 \geq \mathbf{b}_2 - \mathbf{B}_2\bar{\mathbf{y}} \\ & \mathbf{x}_2 \geq \mathbf{0} \end{aligned}$$

Notice that this structure and solution approach can be generalized for a case where the problem could be split into k subproblems.

The problem of fixing the $\mathbf{y}$ variables to eventually achieve global optimality is called the *master problem*. The variables of the master problem are therefore the linking (complicating) variables of the original problem and its objective derives from the sum of the optimal values of the subproblems.

This basic decomposition method is called *primal decomposition* because the master problem and the subproblems are defined only on the primal variables. A generic primal decomposition method solves problem P by iteratively solving the master problem. Each iteration fixes the linking variables and proceeds solving the subproblems obtaining information on how to update the linking variables at the next master iteration. Such a decomposition method is effective when there are few complicating variables, and there are efficient algorithms for solving the subproblems.

An example of a structure of a linear problem having linking constraints suitable for effective decomposition can be the following problem P:

$$\begin{aligned} z_P = \min\ & \mathbf{c}_1\mathbf{x}_1 + \mathbf{c}_2\mathbf{x}_2 && (21)\\ s.t.\ & \mathbf{A}_1\mathbf{x}_1 \geq \mathbf{b}_1 && (22)\\ & \mathbf{A}_2\mathbf{x}_2 \geq \mathbf{b}_2 && (23)\\ & \mathbf{B}_1\mathbf{x}_1 + \mathbf{B}_2\mathbf{x}_2 \geq \mathbf{b}_3 && (24)\\ & \mathbf{x}_1, \quad \mathbf{x}_2 \geq \mathbf{0} && (25) \end{aligned}$$

Dualizing the linking constraints (24) in the objective function by a non-negative Lagragian *penalty* (or *multiplier*) vector $\boldsymbol{\lambda}$, we obtain the following *Lagrangian Relaxation* LR (Beasley 1993b; Guignard and Kim 1987):

$$\begin{aligned} z_{LR}(\boldsymbol{\lambda}) = \min\ & \mathbf{c}_1\mathbf{x}_1 + \mathbf{c}_2\mathbf{x}_2 - \boldsymbol{\lambda}(\mathbf{B}_1\mathbf{x}_1 + \mathbf{B}_2\mathbf{x}_2 - \mathbf{b}_3) && (26)\\ s.t.\ & \mathbf{A}_1\mathbf{x}_1 \geq \mathbf{b}_1 && (27)\\ & \mathbf{A}_2\mathbf{x}_2 \geq \mathbf{b}_2 && (28)\\ & \mathbf{x}_1, \quad \mathbf{x}_2 \geq \mathbf{0} && (29) \end{aligned}$$

that can be rewritten as:

$$z_{LR}(\boldsymbol{\lambda}) = \min \ \mathbf{c}_1'\mathbf{x}_1 + \mathbf{c}_2'\mathbf{x}_2 + \boldsymbol{\lambda}\mathbf{b}_3 \tag{30}$$

$$s.t. \ \ \mathbf{A}_1\mathbf{x}_1 \geq \mathbf{b}_1 \tag{31}$$

$$\mathbf{A}_2\mathbf{x}_2 \geq \mathbf{b}_2 \tag{32}$$

$$\mathbf{x}_1, \quad \mathbf{x}_2 \geq \mathbf{0} \tag{33}$$

where $\mathbf{c}_1' = \mathbf{c}_1 - \boldsymbol{\lambda}\mathbf{B}_1$ and $\mathbf{c}_2' = \mathbf{c}_2 - \boldsymbol{\lambda}\mathbf{B}_2$ are the *penalized costs*.

Problem LR is block separable in the variables $\mathbf{x}_1$ and $\mathbf{x}_2$ and could be split into two subproblems. For a given $\boldsymbol{\lambda}$, we evaluate $z_{LR}(\boldsymbol{\lambda}) = z_{LR1}(\boldsymbol{\lambda}) + z_{LR2}(\boldsymbol{\lambda}) + \boldsymbol{\lambda}\mathbf{b}_3$ by solving the following two subproblems independently:

$$z_{LR1}(\boldsymbol{\lambda}) = \min \ \mathbf{c}_1'\mathbf{x}_1 \quad \text{s.t.} \ \ \mathbf{A}_1\mathbf{x}_1\mathbf{b}_1 \quad \mathbf{x}_1 \geq \mathbf{0}$$

$$z_{LR2}(\boldsymbol{\lambda}) = \min \ \mathbf{c}_2'\mathbf{x}_2 \quad \text{s.t.} \ \ \mathbf{A}_2\mathbf{x}_2 \geq \mathbf{b}_2 \quad \mathbf{x}_2 \geq \mathbf{0}$$

Notice that also this structure and solution approach can be generalized for a case where the problem could be split into k subproblems.

Duality and decomposition methods provide some very useful data that can feed heuristic algorithms, in particular, dual variables and Lagrangian penalties have the power to gather information on the overall structure of the problem and of the specific instance to be solved.

As the reduced and penalized costs drive the exact methods to the optimal solution, they can be of help also for the heuristic approaches. In the following we show how duality can be used for developing constructive heuristics, and we further extend this opportunity to the dual and penalized costs and the primal solutions generated during the execution of the decomposition approaches.

6.1 Lagrangian heuristics

Lagrangian decomposition has been the seed of different heuristic algorithms, mostly based on one of the best known approaches for solving the Lagrangian dual, the *subgradient algorithm* originally proposed by Shor et al. (1985). This is an iterative procedure that, at each iteration k, computes a new approximation $\boldsymbol{\lambda}^{k+1}$ of the Lagrangian multipliers in such a way that, for $k \to +\infty$, $\boldsymbol{\lambda}^k$ is an optimal or near optimal solution of the corresponding Lagrangian dual.

The subgradient algorithm generates at each iteration a new (possibly unfeasible) primal solution $\mathbf{x}^k$, a new Lagrangian penalty vector $\boldsymbol{\lambda}^k$ and, therefore, new penalized costs $\mathbf{c}' = \mathbf{c} - \boldsymbol{\lambda}^k\mathbf{A}$. These can be useful parameters for heuristic algorithms, typically constructive heuristics, using the penalized costs for choosing moves or applying some repair procedures to the unfeasible primal solutions.

A note is worth making for the penalty update. Considering for example relaxed inequality constraints, the Lagrangian multipliers would be updated by a simple local search as follows:

$$\lambda_i^{k+1} = max\left\{0, \lambda_i^k + \alpha^k g_i^k\right\}, \quad i \in V \tag{34}$$

where g_i^k is the i-th component of the subgradient (i.e., the amount of infeasibility on the corresponding relaxed constraint) and α^k is the length of the step along the search direction given by the subgradient itself. The literature proposes several rules to update the step size α^k. The standard update rule proposed by Polyak (1969) is:

$$\alpha^k = \beta^k \frac{\bar{z} - z_{LR}(\boldsymbol{\lambda}^k)}{\left\| \boldsymbol{g}^k \right\|^2} \tag{35}$$

where $\bar{z}$ is an overestimate of the optimal Lagrangian dual solution z_{LR}. Polyak proved the convergence of the method for $\epsilon \leq \beta^k \leq 2$. Unfortunately, according to this approach, the optimization process is based on global parameters, the elements of the subgradient $\boldsymbol{g}$, which restricts the potential for distributed or parallel implementation of this type of decomposition. However, there exist alternative update rules such as the *quasi-constant* step size update that addresses this issue. Applications exploiting this possibility and devising a matheuristic for a peer to peer network design problem have been presented in Boschetti et al. (2011, 2019).

Alternative approaches for solving the Lagrangian dual are the multiplier adjustment (Fisher et al. 1986), the volume algorithm (Barahona and Anbil 2000), or the bundle methods (Hiriart-Urruty and Lemarechal 1993). All these methods generate new primal solutions and penalties at each iteration, and provide the same opportunities for developing heuristics as those given by the subgradient method.

In the literature, a wide range of heuristic algorithms that make use of the Lagrangian penalties has been proposed. These algorithms are known as *Lagrangian heuristics*, and there is a huge variety of them, literally hundreds of contributions. Focusing for example just on location problems, contribution include works on general location problems (Beasley 1993a), on capacitated plant location problems (Agar and Salhi 1998; Barcelo and Casanova 1984; Sridharan 1991), or on facility location (Holmberg and Ling 1997), among others.

Lagrangian heuristics are typically primal heuristics trying to fix the subproblem solution so to have something feasible and hopefully of good quality. Interestingly, Lagrangian relaxation permits also the exploitation of dual information contained in the Lagrangian penalties. There is in fact a strong relation between the Lagrangian penalties and the dual variables associated with the same constraints, that can for example be used in dual ascents procedures. The literature shows that for some models it is possible to generate, for each set of Lagrangian penalties, a dual solution having the same value of the Lagrangian bound. This has many advantages, it is in fact possible to use dual variables, penalties and primal solutions to feed some heuristic algorithm for generating new improved feasible solutions, but also for generating new variables in a column generation fashion (see Sect. 6.2.1).

This possibility was implemented in Boschetti and Maniezzo (2015), where the authors describe an application to a real-world city logistics problem for a mid-sized town, whose core is modeled as a multitrip vehicle routing problem with time windows, pickup and deliveries, and heterogeneous fleet. The proposed matheuristic is based on the dual ascent procedure applied to an extended set covering model (SC) where

columns are generated based on the dual information derived from the Lagrangian penalty vector.

Similarly, in Boschetti et al. (2020), dual ascent is used for the problem of the generation of pivot tables, that are one of the most popular tools for data visualization in both business and research applications, however, their intelligibility becomes progressively lower when the quantity of data to be visualized increases, causing the so-called *information flooding* problem. To cope with the information flooding problem, a so-called *shrink operation* enables users to balance the quantity of data to present with their approximation. The authors propose a model for optimizing the implementation of the shrink operation as set partitioning problems with a side constraint, that is solved by a matheuristic that combines a dual ascent procedure, a Lagrangian pricing approach, and a Lagrangian heuristic.

6.2 Dantzig–Wolfe heuristics

Dantzig–Wolfe decomposition (Dantzig and Wolfe 1960) is a procedure best applied to problems that can be formulated on a constraint matrix where some constraints can be grouped in a block-diagonal structure, while the remaining ones are left as coupling constraints.

The master problem includes all coupling constraints, it is initialized on columns containing no or just one of the blocks, then the columns corresponding to the successive blocks are added defining successive subproblems. The master problem contains all currently active columns and checks whether each subproblem can add ("*generate*") some of its columns to the current basis, thereby improving the objective function.

This abstract structure has been generalized into column generation methods, that proved highly effective on a wide range of combinatorial optimization problems. Matheuristics have been obtained both turning column generation into heuristics and heuristically applying Dantzig–Wolfe to general MILP formulations.

6.2.1 Column generation heuristics

Column generation approaches, rooted in Dantzig–Wolfe decomposition, are very effective when the number of variables is huge, as it happens for many real-world problem models.

One of the classical examples is the Bin Packing Problem (BPP) that consists in minimizing the number of bins of capacity W required for loading the set I of m items of size w_i. Among the possible models, one of the most effective is $z_{BPP} = \min_{\mathbf{x}\in\{0,1\}^{|S|}}\{\sum_{j\in S} x_j : \sum_{j\in S_i} x_j = 1, i \in I\}$, where S is the index set of all feasible *configurations* (i.e., subset of items having a total size smaller than or equal to W) and S_i are the configurations containing item i. Instead of enumerating all the feasible configurations (i.e., columns/variables), we only generate the configurations having the potential to be in the optimal solution. For doing that, we generate a small initial set of columns (e.g., a feasible heuristic solution), then we solve the LP-relaxation of BPP and using the current dual variables u_i, $i \in I$, we generate a new column solving the knapsack problem $z_{KP} = \max_{\mathbf{y}\in\{0,1\}^m}\{\sum_{i\in I} u_i y_i : \sum_{i\in I} w_i y_i \leq W\}$. If

its solution $\mathbf{y}^*$ is such that $\sum_{i \in I} u_i y_i^* \leq 1$, we have reached the optimal solution of the LP-relaxation of BPP, otherwise we add the new column having the coefficients defined by the values of solution $\mathbf{y}^*$.

In many applications, the generation of the columns requires the solution of difficult problems, therefore we may consider the possibility to heuristically generate the columns. The result can be a procedure that applies exact column generation when the heuristic procedure does not find any candidate columns or that only uses the heuristically generated columns, thus the overall algorithm definitely is a heuristic (see, for example, Boschetti and Maniezzo 2015).

Another possibility is to generate more columns at a time. For example, we can generate the k least reduced cost columns using the current dual solution (see Boschetti et al. 2004; Mingozzi et al. 1999). Similarly, we can also select (*price*) the k least reduced cost columns from the complete set of columns already available (see Boschetti et al. 2008, 2020). In these cases, if we solve the resulting reduced problem, its optimal solution (if there exists) is certainly a heuristic solution for the original problem, but we are able to prove its possible optimality or estimate its maximum distance from the optimal solution. This feature has a great potential, because it is not usually possible to estimate the maximum gap between heuristic and optimal solutions. Obviously, the literature shows that the better the quality of the dual solution, the better the quality of the heuristic solution.

6.3 Benders heuristics

Benders decomposition (Benders 1962) is another decomposition technique best applied to linear problems whose formulation has a block diagonal structure. In Benders' context, it is worth mentioning that this block structure has often been pointed out and utilized in stochastic programming applications, where blocks derive from scenarios.

In Benders decomposition, the constraints of the problem are divided into two subsets, where one (possibly initially empty) subset pertains to the master problem. If the solution is infeasible for the subproblem, then cuts are generated [the "Benders cuts", generalized into "combinatorial Benders cuts" when the original problem is not an LP but a mixed integer linear problem (Codato and Fischetti 2004)] and added to the master problem, with the objective to drive it to feasibility. The updated master problem is then solved again, and the procedure goes on until no cuts can be generated.

Observing that Benders decomposition adds new cuts, thus constraints, at each iteration, the approach is called "row generation", as opposed to the column generation deriving from Dantzig–Wolfe decomposition.

Benders decomposition has been the least utilized for designing matheuristics. There have been a number of implementations of Benders' based heuristics, but they are usually very problem-dependent, not easy to abstract and generalize into a matheuristics, that is into a framework that can be readily adapted to different problems. One such effort was presented in Boschetti and Maniezzo (2009b), where a "*bendHeuristic*" pseudocode suggested to use the standard Benders' decomposition and cutting planes insertion framework, applied to the LP relaxation of the problem of

interest in case it is a combinatorial one, essentially suggesting to solve heuristically the master problem. This bars the possibility to obtain a provably optimal solution, but produces an approach that was successfully tested on different combinatorial optimization problems, including the single capacitated facility location, the multi-mode project scheduling and the membership overlay problem.

This structure is closely related to *combinatorial Benders' cuts* (Codato and Fischetti 2004), a decomposition scheme that is defined over a master Integer Linear Problem with no continuous variables, but containing combinatorial information on the feasible integer variable combinations, and a slave Linear Program, which possibly returns combinatorial inequalities to be added to the current master. The inequalities are associated with infeasible subsystems of the relevant linear system, and must be separated efficiently in case the master solution is integer. The overall solution mechanism is closely akin to Benders' decomposition, but the cuts we produce are purely combinatorial, and quite similar to the *bendHeuristic* outlined above, the difference being in the focus on heuristic solution of the latter and on tightening bounds of the former.

6.4 Surrogate relaxation heuristics

Surrogate relaxation was first proposed for integer programming by Glover (1965, 1968). It is less common than other techniques discussed in this Section, but its principle is simple, as it suggests to replace a set of constraints with their linear combination.

Given the problem $z_P = \min\{\mathbf{cx} : \mathbf{Ax} \geq \mathbf{b}, \mathbf{x} \in X\}$, if we surrogate the constraints $\mathbf{Ax} \geq \mathbf{b}$ by the non-negative *surrogate multipliers* $\boldsymbol{\mu}$, we have the surrogate problem $z_{SR}(\boldsymbol{\mu}) = \min\{\mathbf{cx} : \boldsymbol{\mu}\mathbf{Ax} \geq \boldsymbol{\mu}\mathbf{b}, \mathbf{x} \in X\}$.

A simple example can be derived for the set partitioning problem (SPP). Given a collection S of subsets of the items I, the SPP minimizes the sum of the values of the subsets of S that are in the solution covering each item of I exactly once, i.e., $z_{SPP} = \min_{\mathbf{x}\in\{0,1\}^{|S|}}\{\sum_{j\in S} p_j x_j : \sum_{j\in S_i} x_j = 1, i \in I\}$, where S_i is the collection of subsets containing the item i. A surrogate relaxation can be obtained replacing the set partitioning constraints as follows:

$$\sum_{j\in S_i} x_j = 1,\ i \in I \quad \Longrightarrow \quad \sum_{i\in I}\sum_{j\in S_i} \mu_i x_j = \sum_{i\in I} \mu_i \tag{36}$$

where each non-negative surrogate multiplier μ_i is associated to each constraint $i \in I$. The resulting problem is a 0–1 knapsack problem with equality constraint, i.e., $z_{SR}(\boldsymbol{\mu}) = \min_{\mathbf{x}\in\{0,1\}^{|S|}}\{\sum_{j\in S} p_j x_j : \sum_{j\in S} w_j x_j = W\}$, where $W = \sum_{i\in I} \mu_i$, $w_j = \sum_{i\in I_j} \mu_i$, and $I_j = \{i \in I : j \in S_i\}$.

Solving the *surrogate dual* problem $z_{SR} = \max\{z_{SR}(\boldsymbol{\mu}) : \boldsymbol{\mu} \geq \mathbf{0}\}$ we define the *surrogate multipliers* $\boldsymbol{\mu}$, that maximize the lower bound $z_{SR}(\boldsymbol{\mu})$. An important theoretical result is that surrogate duality gaps are at least as small as Lagrangian duality gaps, possibly smaller [see the theoretical analyses of surrogate duality proposed by Greenberg and Pierskalla (1970), Glover (1975) and Dokka et al. (2021a)].

The surrogate dual can also be solved using iterative approaches, similar to the subgradient method for the Lagrangian relaxation (Sect. 6.1). At each iteration, we

have a possibly unfeasible primal solution that can be repaired by some heuristic procedures. For example, when we apply the surrogate relaxation to the SPP as shown in (36) the primal solution obtained can be unfeasible because some "*row*" is not covered exactly once. In this case, a procedure can eliminate or add some columns from the solution in order to recover the feasibility; the difficulty is how to implement this procedure to have feasible solutions of good quality.

A seminal paper about surrogate heuristics is Glover (1977) where the authors define a general framework and gives some insights for implementing effective heuristic algorithms. Other interesting surrogate heuristics are proposed by Lorena and Belo Lopes (1994) for the set covering problem and by Boyer et al. (2009) and Dokka et al. (2021b) for the 0–1 multidimensional knapsack problem. The algorithm proposed by Lorena and Belo Lopes (1994) is based on a continuous surrogate relaxation and subgradient optimization, while Boyer et al. (2009) defines a surrogate relaxation and solves the relaxed problem by a modified dynamic-programming algorithm. The algorithm proposed by Dokka et al. (2021b) uses the information generated during the solution of the surrogate dual to drive a primal heuristic, whereas Narciso and Lorena (1999) and Senne and Lorena (2000) consider a combined application of Lagrangian and surrogate relaxation.

7 Metaheuristic hybrids

Metaheuristics are problem agnostic approaches, that can be turned into solution algorithms for specific problems with little added details. They date back from the 70s and have traditionally striven for simplicity, including little if any mathematical component in their functioning. Metaheuristics have often proven their undoubted effectiveness, however, along with the awareness of their eventual limits, also the awareness has risen of the possible contributions that MP components can grant them. Most well-accepted metaheuristic frameworks have enjoyed MP components, the resulting algorithm is often presented in the literature as a metaheuristic hybrid (along with combinations of metaheuristics among themselves, such as, for example, of genetic algorithms with simulated annealing).

7.1 Single solution heuristics

Metaheuristic approaches can be classified according to different criteria, one being the number of solutions that are evolved at each stage of the algorithm: one single solution or more than one. This section deals with metaheuristic algorithms that evolve one single solution, they are all essentially enhancements of a basic local search procedure. Many different approaches have been presented, which could be included here. We choose four of them, namely Simulated Annealing, Tabu Search, Iterated Local Search, and Variable Neighborhood Search, as representatives of the class.

Matheuristic components have been used to complement each of them, along with most of the relevant unreported ones. The techniques used to include mathematical

components in the basic structure of the considered metaheuristic tend to be general and independent of the specific metaheuristic of interest.

7.1.1 Iterated local search

Iterated local search (ILS, Lourenço et al. 2002) is an extension of plain local search, where a simple mechanism is added to avoid to get stuck in local optima. It has been generalized into *Stochastic Local Search* (Hoos and Stützle 2004) to make evident the idea of implementing a sampling of the search space based on the local optima identified by whichever specific local search is implemented. The sampling can be easily achieved by repeating a random generation of a starting solution and applying the local optimizer to it, but there is experimental evidence that a tighter control on the starting solutions can lead to better results.

ILS guides a local search heuristic at its core by generating a reasoned sequence of starting solutions. This usually leads to better results than if one were to use repeated random trials of that same core heuristic. The algorithm can be very simple, as it only prescribes to start from a solution, find its local optimum with reference to a specific quality measure, perturb the incumbent solution, optimize this new one, and so on. Any local search procedure can be included, from simple ones such as, for example in the case of the TSP, *2-opt* and *3-opt* (i.e., complete explorations of variations of 2 or 3 solution components), to very involved ones such as Lin Kernighan or local branching (see Sect. 4.1).

ILS builds upon a number of internal procedures. A feasible solution is generated by any constructive approach, and the results often turn out to be significantly independent of the starting solution. More relevant for the quality of the results is the perturbation step, that requires to modify the current solution into a new one, which is not too close to the original one, otherwise it would fall into the same local optimum basin of attraction, nor too far from it, otherwise the whole algorithm becomes a random restart. It is helpful to this end to be able to control the amount of perturbation to be imposed on the instance to solve.

ILS can also make use of different mathematical elements, as surveyed in Lourenço et al. (2010). Lopes et al. (2015) describe how ILS can be used for approximately solving a linear integer programming (IP) formulation of a real-life machine reassignment problem proposed in a Google ROADEF/EURO Challenge. A combination with a heuristic for solving a nonlinear formulation of a subproblem arising in a cutting stock problem application is presented in Umetani et al. (2003).

7.1.2 Variable neighborhood search

Variable neighborhood search (VNS, Mladenovic and Hansen 1997) proposes a variation of the ILS idea by implementing a sequence of neighborhoods at each main search iteration, instead of a single one. This phase is called a Variable Neighborhood Descent (VND) and it is followed by a perturbation of the incumbent solution as in ILS. In the VND context, perturbation is often implemented as the generation of a random solution in one further, larger neighborhood.

This procedure is motivated by the observation that a local minimum with respect to one neighborhood is not necessarily a local minimum for other neighborhoods. Only a global optimum is guaranteed to be a local optimum for any neighborhood function.

The core idea of varying the neighborhood used during search has been implemented in many different ways, giving rise to many variants, such as variable neighborhood descent, basic variable neighborhood search, reduced variable neighborhood search, and variable neighborhood decomposition search. Here we present a version where VNS contains at its heart a basic neighborhood sequencing procedure, the VND method.

VND is usually simple to implement and effective, thus it was chosen as a primary method for solving very different problems, and also its matheuristics extensions reflect this flexibility. An inclusion of a local branching neighborhood is described in Hu and Raidl (2006), a combination with integer linear programming for the generalized minimum spanning tree problem in Hu et al. (2008), but several applications involve VNS including a MIP based local search (Fonseca et al. 2016; Pirkwieser and Raidl 2010; Prandtstetter and Raidl 2008).

A further interesting extension, named relaxation guided variable neighborhood search (Puchinger and Raidl 2008), makes use of the general multiple-neighborhood VNS scheme, but the order in which the different neighborhoods are sequenced is not hardcoded but it is determined dynamically by solving relaxations of them. The objective values of these relaxations are used as indicators for the potential gains of searching the corresponding neighborhoods.

7.1.3 Simulated annealing

Simulated annealing (SA) was introduced in Kirkpatrick et al. (1983) based on the Monte Carlo model of Metropolis et al. (1953). The general structure is therefore similar to the local search algorithm, where an incumbent solution $\mathbf{x}$ is iteratively updated, possibly moving it to another solution $\mathbf{x}'$ in its neighborhood $\mathcal{N}(\mathbf{x})$, except that in the case of SA the moves can be made also toward worsening solutions.

Worse solutions are accepted in probability, where the probability of acceptance decreases with the decrease in quality of the solution. The move acceptance formula for worsening solutions mimics the Metropolis formula, derived in turn from the numerator of the Boltzmann equation, where the energy values are replaced by the objective function values (i.e., high energy corresponds to high cost). Since the anneal permits an effective decrease of the free energy, the simulated anneal hopefully provides an effective means to decrease the solution costs.

SA is another very simple algorithm, and simple to combine with mathematical modules. An example applied to a school timetable problem can be found in Avella et al. (2007), where SA is superimposed to a VLSNS, whose neighborhood is explored by solving an Integer Programming problem. Another timetable application is described in Gunawan et al. (2012), where the authors design a simple matheuristic where an initial solution, obtained by solving a Lagrangian relaxation of the problem, is later improved by a SA.

7.1.4 Tabu search

Tabu search (TS, Glover 1989, 1990) is another iterative procedure, which extends a core local search to help it escape from local optima. This is achieved by making use of an additional memory structure whose objective is to prevent the algorithm from repeatedly visit the same solutions. The memory of past search is stored in the *tabu list*, and used to limit the successive moves. Since this is the only limitation, TS grants the possibility to accept worsening moves when they are anyway neighborhood best.

The tabu list thus acts as a short-term memory structure, and it is possibly combined with other structures implementing a long-term memory. Moreover, usually it is not solutions that are stored in the tabu list, but moves, that correspond to the local search moves that modified a current solution leading it to the next explored one. The tabu list prevents reversing the stored moves for a number of iterations specified by a parameter called *tabu tenure*. This permits escaping from local optima and supports therefore search diversification. The long term memory, when present, collects information about the explored regions of the search space and is used to direct search toward unexplored regions, thereby providing a strategic diversification guidance.

There are multiple possibilities for integrating this basic working with MP elements. For example, Gendron et al. (2016), working on the multicommodity fixed-charge network design problem, propose to solve an LP relaxation of the problem and possibly fix it to get a heuristic solution. Then, in order to avoid returning to already explored solutions a cut is added to the LP formulation, taking the place of the tabu list. Similarly, Yaghini et al. (2013) use added cuts combining a cutting-plane neighborhood structure and a tabu search metaheuristic for the capacitated p-median problem. The neighborhood structure they propose consists first in closing an open median, then, generating a LP model by relaxing binary constraints and adding new constraints. The LP solution is the new neighbor. The neighborhood structure is then combined with a standard tabu search. Ngueveu et al. (2009) defined the neighborhood based on a b-matching (or b-directed flow) problem for a m-peripathetic VRP, a VRP variant defined on a periodic horizon and asking to use each arc at most once per period. The b-matching suggests the arcs to test in neighborhood exploration.

7.2 Population heuristics

A specialized thread of metaheuristic research, bordering and often overlapping with Artificial Intelligence, studied heuristics that evolved whole sets of candidate solutions, often named "*populations*" of solutions [a widely shared strong criticism of the excessive use of metaphors in optimization has been published by Sörensen (2015)]. Genetic algorithms were among the first results, and following their success it became common to get inspiration from some natural phenomenon to design the heuristics. This chapter considers three representative population-evolving metaheuristics, namely genetic algorithms, ant colony optimization and scatter search (with path relinking), and shows how they have been complemented with mathematical programming modules to achieve better performance.

7.2.1 Evolutionary algorithms

Evolutionary algorithms (EAs), also referred to as Evolutionary computation (EC), have been designed loosely following a population evolution inspiration. They are a collection of different methods sharing a few properties; the most important being that they are iterative methods updating a set of solutions, where a solution at an iteration's set is selected in probability to be included in the next iteration, with a probability depending on its quality, as measured by the function to optimize. This in EC terms becomes "selective pressure drives the species living in a given environment to structures that ensure a better probability of surviving and of reproducing". The literature proposes several EC methods, including Genetic Algorithms, Evolution Strategies, Evolutionary Programming and Genetic Programming.

Genetic algorithms (GAs, Goldberg 1989; Holland 1975) are thus iterative search algorithms where a set of solutions is updated at each iteration. The update is made applying to each solution a minimal local search step (named *mutation*), just a random change of the value of one or of a few solution variables, a recombination operator applied to pairs of solutions (named *crossover*), swapping between them the values of randomly chosen variables, and usually a montecarlo sampling with repetition (named *selection*) to update the solution set. Initially, GAs were proposed for working on solutions codified as equal length strings of boolean variables, then they have been generalized to different representations. The denotation Genetic algorithms is in the plural, because it does not signify an algorithm, rather a class of algorithms, which share the general structure outlined above, but can implement the three essential components in widely different ways.

Evolution strategies (ES, Beyer and Schwefel 2002) are algorithms similar to GAs, but they have been designed for continuous variables and undergo a more prescriptive description. The general structure of the ES is again one main loop on a set of candidate solutions, which are modified by three main operators, named selection, reproduction, and mutation. Peculiar to ES is the encoding within the real valued array that represents a solution of both the data and of the control parameters of the operators that will act upon it.

The inclusion of MP modules has been primarily focused on optimizing the crossover operator, the primary search motor of ES. Two main lines have been investigated: fixing the solution parts common to both parents and optimizing the rest (Yagiura and Ibaraki 1996) or making the union of the parent solution components and optimizing within that set (Aggarwal et al. 1997). The most successful among these two lines has been the first one, as it proposes a solution to a common problem arising in variable fixing (see also Sect. 4.1): how many variables to fix. If the number is too small, the optimization will likely produce one of the two parents, while if it is too high, there is little advantage with respect to solving the whole problem.

Optimized crossover has been applied to supply management problems (Borisovsky et al. 2009) and to the problem of balancing transfer lines with multi-spindle machines (Dolgui et al. 2009). A different contribution leveraged on the possibility of optimizing a two-level formulation of a relay placement problem for wireless sensor networks (Flushing and Di Caro 2012). The top level problem was solved by the GA, while a MILP solver completed the solution by solving the subproblem.

7.2.2 Ant colony optimization

Ant colony optimization (ACO, Dorigo and Stützle 2004) is the name given to a class of algorithms, which expand and generalize ideas originally presented in a parallel constructive method called *Ant System* (Colorni et al. 1991; Dorigo et al. 1996).

A characterizing feature of all ACO algorithms is the way solutions get constructed. Again, the algorithm has a main loop, where at each iteration a set of solutions is constructed. Each solution is built from scratch in probability, based on a driving heuristic and on a shared memory structure that accounts for the expected quality of solutions including each particular component. The structure is updated each time a new solution gets completed, thus its quality and components can be assessed, and represents the grounding of an indirect communication among the successive constructive threads.

Two contributions framing MP within ACO algorithms were already presented above, namely ANTS (Sect. 3.2) and Beam-ACO (Sect. 3.1), but other examples exist. A case was proposed by Reimann (2007), where the author proposed an ACO method for solving a symmetric traveling salesman problem where the attractiveness among pairs of customers is defined using information derived by the calculation of a Minimum Spanning Tree Problem. Another is due to D'Andreagiovanni (2014), working on cooperative wireless networks. In this case, the quality of the feasible solutions found through the ant-construction phase is refined by a modified Relaxation Induced Neighborhood Search (RINS, see Sect. 4.1).

7.2.3 Particle swarm optimization

Particle swarm optimization (PSO, Kennedy and Eberhart 1995) is a derivative free, continuous optimization method, making reference to swarms in its metaphorical language. As in previous cases in this section, the algorithm is based on a main loop, where a set of solutions is updated. The tentative solutions are expected to explore the search space by moving inside it. Each solution applies a velocity vector to its current position, obtaining a new position. The velocity vector is specific to each solution, it has some inertia but it is also influenced by the best solutions found in the solution history or by the best solution found by the whole solution set.

PSO was conceived for solving continuous optimization problems, but it has been bent also toward combinatorial problems. One work presenting a PSO-based matheuristic applied to the binary cutting stock problem, in a comparison context, was presented by Sanchez et al. (2018). The authors used a column generation framework (see Sect. 6.2.1), where the master problem was solved by a MIP solver, and the subproblem was solved using different metaheuristic algorithms, among which PSO.

Dewan et al. (2014) couples PSO with a MILP solver first letting PSO run until the stopping criterion is met, then feeding its results to a MILP solver, thus letting it start from a high quality incumbent solution. They successfully applied this technique to the thermal unit commitment problem in power generation planning, a problem arising in energy production defined over a quadratic cost function.

8 Conclusions and future research directions

In this survey on Matheuristics we mainly consider the heuristic approaches that make use of linear programming, however the developments in nonlinear programming in recent years give new opportunities. In the literature we can already find several heuristic approaches using nonlinear models and the related math. Many mathematical tools for nonlinear programming are in fact not very effective for the exact solution of real-world problems, but they could be very useful for developing heuristic algorithms. For example, mipping or decomposing nonlinear models can be interesting options for exploring portions of the feasible region having properties that make the solution process easier and effective.

Decomposition methods offer further opportunities for developing fully-decentralized heuristic algorithms or for exploiting parallel computing (e.g., GPU and many-core computing). In Sect. 6.1 we cite some examples of applications where the *global problem* can be decomposed by Lagrangian relaxation in many *local components* which solve their local Lagrangian problem, update their local Lagrangian multipliers, and compute their local heuristic solution. In this case it is possible to design heuristic algorithms where the local optimization procedures are able to produce overall near-optimal solutions for the global problem by exchanging among the local components only the Lagrangian multipliers which encapsulate the global information about the problem to be solved.

This research direction is very promising, in particular in the case of complex, large-size problems arising in real-world systems, where many individual entities have to take some decisions that need to be globally optimal (e.g., communication and supply chain networks, urban traffic management, etc.). In these situations, we can use centralized solutions or more flexible and scalable fully-distributed approaches, where each individual entity optimizes its problem and exchanges information with a small subset of other entities (its neighbors), but allowing everyone to obtain a satisfactory solution and an overall near-optimal solution.

An interesting observation is that there exist approaches described in the mathematical literature that have not yet proved effective enough to be used for the development of exact methods and, for this reason, they are not very popular today, but they could prove effective when designing heuristics. An example is the surrogate relaxation discussed in Sect. 6.4, which has interesting properties that give a great potential even for developing exact methods, but which can also be used for developing heuristic algorithms, sometimes combined with other approaches such as Lagrangian relaxation. Probably, there exist some neglected mathematical methods that could be rediscovered to develop new matheuristics, as well as to be better used in exact methods.

A challenge for the next years will be to identify new mathematical approaches suitable for developing both exact methods and matheuristics, without forgetting their possible use in metaheuristics. We also believe that the convergence between exact and matheuristic algorithms could be an interesting research direction, aimed at devel-

oping exact methods that can be easily transformed into heuristics and able to quickly generate feasible, near-optimal solutions.

Funding Open access funding provided by Alma Mater Studiorum - Università di Bologna within the CRUI-CARE Agreement.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

Agar M, Salhi S (1998) Lagrangean heuristics applied to a variety of large capacitated plant location problems. J Oper Res Soc 49:1072–1084

Aggarwal C, Orlin J, Tai R (1997) An optimized crossover for the maximum independent set. Oper Res 45:226–234

Ahuja RK, Orlin JB, Sharma D (1999) New neighborhood search structures for the capacitated minimum spanning tree problem. Technical Report 99-2, Department of Industrial and Systems Engineering, University of Florida

Ahuja RK, Orlin JB, Sharma D (2000) Very large-scale neighborhood search. Int Trans Oper Res 7(4–5):301–317

Ahuja RK, Ergun O, Orlin JB, Punnen APA (2002) Survey of very large-scale neighborhood search techniques. Discrete Appl Math 123:75–102

Angelelli E, Mansini R, Speranza MG (2007) Kernel search: a heuristic framework for MILP problems with binary variables. Technical report, Department of Electronics for Automation, University of Brescia, R.T.2007-04-56

Angelelli E, Mansini R, Speranza MG (2010) Kernel search: a general heuristic for the multi-dimensional knapsack problem. Comput Oper Res 37(11):2017–2026

Angelelli E, Mansini R, Speranza MG (2012) Kernel search: a new heuristic framework for portfolio selection. Comput Optim Appl 51(1):345–361

Avella P, D'Auria B, Salerno S, Vasil'ev I (2007) A computational study of local search algorithms for Italian high-school timetabling. J Heuristics 13:543–556

Barahona F, Anbil R (2000) The volume algorithm: producing primal solutions with a subgradient method. Math Program 87:385–399

Barcelo J, Casanova J (1984) A heuristic Lagrangean algorithm for the capacitated plant location problem. Eur J Oper Res 15:212–226

Bartolini E, Mingozzi A (2009) Algorithms for the non-bifurcated network design problem. J Heuristics 15(3):259–281

Bartolini E, Maniezzo V, Mingozzi A (2008) An adaptive memory-based approach based on partial enumeration. In: Maniezzo V, Battiti R, Watson JP (eds) LION 2, LNCS 5313. Springer, Berlin, pp 12–24

Bazaraa MS, Jarvis J, Sherali HD (1990) Linear programming and network flows. Wiley, Hoboken

Beasley J (1993a) Lagrangian heuristics for location problems. Eur J Oper Res 65:383–399

Beasley JE (1993b) Lagrangian relaxation. In: Reeves CR (ed) Modern heuristic techniques for combinatorial problems. Wiley, New York, pp 243–303

Bellman R (1957) Dynamic programming and the numerical solution of variational problems. Oper Res 5(2):277–288
Benders JF (1962) Partitioning procedures for solving mixed-variables programming problems. Numer Math 4:280–322
Beyer HG, Schwefel HP (2002) Evolution strategies—a comprehensive introduction. Nat Comput 1(1):3–52
Bixby RE, Fenelon M, Gu Z, Rothberg E, Wunderling R (2000) MIP: theory and practice—closing the gap. Kluwer Academic Publishers, Amsterdam, pp 19–49
Blum C (2005) Beam-ACO—hybridizing ant colony optimization with beam search: an application to open shop scheduling. Comput Oper Res 32(6):1565–1591
Blum C (2008) Beam-ACO for simple assembly line balancing. INFORMS J Comput 20(4):618–627
Borisovsky P, Dolgui A, Eremeev A (2009) Genetic algorithms for a supply management problem: MIP-recombination vs greedy decoder. Eur J Oper Res 195(3):770–779
Boschetti M, Maniezzo V (2009a) Benders decomposition, Lagrangean relaxation and metaheuristic design. J Heuristics 15:283–312
Boschetti MA, Maniezzo V (2009b) Benders decomposition, Lagrangian relaxation and metaheuristic design. J Heuristics 15(3):283–312
Boschetti MA, Maniezzo V (2015) A set covering based matheuristic for a real-world city logistics problem. Int Trans Oper Res 22(1):169–195
Boschetti MA, Mingozzi A, Ricciardelli S (2004) An exact algorithm for the simplified multi depot crew scheduling problem. Ann Oper Res 127:177–201
Boschetti MA, Mingozzi A, Ricciardelli S (2008) A dual ascent procedure for the set partitioning problem. Discrete Optim 5(4):735–747
Boschetti M, Maniezzo V, Roffilli M (2009a) Decomposition techniques as metaheuristic frameworks. In: Maniezzo V, Stützle T, Voß S (eds) Matheuristics, vol 10. Annals of information systems. Springer, Boston
Boschetti M.A, Maniezzo V, Roffilli M, Bolufé Röhler A (2009b) Matheuristics: optimization, simulation and control. In: Blesa M, Blum C, Di Gaspero L, Roli A, Sampels M, Schaerf A (eds) Hybrid metaheuristics, vol 5818. HM 2009. Lecture notes in computer science. Springer, Berlin
Boschetti MA, Maniezzo V, Roffilli M (2011) Fully distributed Lagrangian solution for a peer-to-peer overlay network design problem. INFORMS J Comput 23(1):90–104
Boschetti MA, Maniezzo V, Strappaveccia F (2019) Membership overlay design optimization with resource constraints (accelerated on GPU). J Parallel Distrib Comput 133:286–296
Boschetti MA, Golfarelli M, Graziani S (2020) An exact method for shrinking pivot tables. Omega 93:10–44
Boyer V, Elkihel M, El Baz D (2009) Heuristics for the 0–1 multidimensional knapsack problem. Eur J Oper Res 199(3):658–664
Brueggemann T, Hurink JL (2007) Two exponential neighborhoods for single machine scheduling. OR Spectrum 29:513–533
Brueggemann T, Hurink JL (2011) Matching based very large-scale neighborhoods for parallel machine scheduling. J Heuristics 17(6):637–658
Caserta M, Voß S (2009a) A cooperative strategy for guiding the corridor method. In: Krasnogor N et al (eds) Nature inspired cooperative strategies for optimization (NICSO 2008), vol 236. Studies in computational intelligence. Springer, Berlin, Heidelberg
Caserta M, Voß S (2009b) Corridor selection and fine tuning for the corridor method. In: Stützle T (ed) Learning and intelligent optimization. LION 2009. Lecture notes in computer science, vol 5851. Springer, Berlin, Heidelberg
Caserta M, Voß S (2014) A hybrid algorithm for the DNA sequencing problem. Discrete Appl Math 163:87–99
Caserta M, Ramirez A, Voß S (2010) A math-heuristic for the multi-level capacitated lot sizing problem with carryover. In: Chio CD et al (eds) Applications of evolutionary computation, vol 6025. EvoApplications 2010. Lecture notes in computer science. Springer, Berlin, pp 462–471
Caserta M, Voß S, Sniedovich M (2011) Applying the corridor method to a blocks relocation problem. Oper Res Spektrum 33:915–929
Chiarandini M, Dumitrescu I, Stützle T (2008) Very large-scale neighborhood search: overview and case studies on coloring problems. In: Blum C, Blesa MJ, Roli A, Sampels M (eds) Hybrid metaheuristics, vol 114. Studies in computational intelligence. Springer, Berlin, pp 117–150
Christofides N, Mingozzi A, Toth P (1981) State-space relaxation procedures for the computation of bounds to routing problems. Networks 11(2):145–164

Codato G, Fischetti M (2004) Combinatorial benders' cuts. In: Bienstock D, Nemhauser G (eds) Integer programming and combinatorial optimization. Springer, Berlin Heidelberg, pp 178–195
Colorni A, Dorigo M, Maniezzo V (1991) Distributed optimization by ant colonies. In: Varela F, Bourgine P (eds) Proceedings of the European conference on artificial life, ECAL'91, Paris. Elsevier Publishing, Amsterdam, pp 134–142
Congram RK, Potts CN, van de Velde S (2002) An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. INFORMS J Comput 14(1):52–67
Cunha CB, Ahuja RK (2005) Very large scale neighborhood search for the k-constrained multiple knapsack problem. J Heuristics 11:465–481
D'Andreagiovanni FA (2014) Hybrid exact-ACO algorithm for the joint scheduling, power and cluster assignment in cooperative wireless networks. In: Di Caro G, Theraulaz G (eds) Bio-inspired models of network, information, and computing systems. Springer, Berlin, pp 3–17
Danna E, Rothberg E, Pape C (2005) Exploring relaxation induced neighborhoods to improve MIP solutions. Math Program 102(1):71–90
Dantzig GB, Wolfe P (1960) Decomposition principle for linear programs. Oper Res 8:101–111
De Franceschi R, Fischetti M, Toth P (2006) A new ILP-based refinement heuristic for vehicle routing problems. Math Program B 105(2–3):471–499
Della Croce F, Ghirardi M, Tadei R (2004) Recovering beam search: enhancing the beam search approach for combinatorial optimization problems. J Heuristics 10(1):89–104
Dewan FR, Viana A, Pedroso J (2014) Metaheuristic search based methods for unit commitment. J Int J Electr Power Energy Syst 59:14–22
Dokka T, Letchford A, Mansoor M (2021a) On the complexity of surrogate and group relaxation for integer linear programs. Oper Res Lett 49(4):530–534
Dokka T, Letchford A, Mansoor M (2021b) Revisiting surrogate relaxation for the multi-dimensional knapsack problem. Oper Res Lett (Submitted)
Dolgui A, Eremeev A, Guschinskaya O (2009) MIP-based GRASP and genetic algorithm for balancing transfer lines. In: Maniezzo V, Stützle T, Voß S (eds) Matheuristics. Annals of Information Systems, vol 10. Springer, Boston. https://link.springer.com/chapter/10.1007/978-1-4419-1306-7_7
Dorigo M, Stützle T (2004) Ant colony optimization. MIT Press, Cambridge
Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern Part B (Cybern) 26(1):29–41
Duin C, Voß S (1999) The pilot method: a strategy for heurisic repetition with application problem in graphs. Networks 34:181–191
Ergun O, Orlin JB, Steele-Feldman A (2006) Creating very large scale neighborhoods out of smaller ones by compounding moves. J Heuristics 12(1–2):115–140
Fischetti M, Fischetti M (2018) Matheuristics. In: Marti R, Pardalos PM, Resende MGC (eds) Handbook of heuristics. Springer, Cham. https://doi.org/10.1007/978-3-319-07124-4_14
Fischetti M, Lodi A (2003) Local branching. Math Program Ser B 98(1–3):23–47
Fischetti M, Lodi A, Salvagnin D (2009) Just mip it! In: Maniezzo V, Stützle T, Voss S (eds) Matheuristics, hybridizing metaheuristics and mathematical programming, vol 10. Annals of information systems. Springer, Boston
Fisher ML, Jaikumar R, Van Wassenhove LN (1986) A multiplier adjustment method for the generalized assignment problem. Manag Sci 32(9):1095–1103
Flushing EF, Di Caro GA (2012) Exploiting synergies between exact and heuristic methods in optimization: an application to the relay placement problem in wireless sensor networks. In: Di Caro G, Theraulaz G (eds) BIONETICS 2012, Lecture notes for computer sciences, social informatics and telecommunications engineering, vol 134, pp 250–265
Fonseca GH, Santos HG, Carrano EG (2016) Integrating matheuristics and metaheuristics for timetabling. Comput Oper Res 74:108–117
Gendreau M, Guertin F, Potvin JY, Seguin R (2006) Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. Transp Res Part C Emerg Technol 14:157–174
Gendron B, Hanafi S, Todosijević R (2016) An efficient matheuristic for the multicommodity fixed-charge network design problem. IFAC PapersOnLine 49(12):117–120
Glover F (1965) A multiphase-dual algorithm for the zero-one integer programming problem. Oper Res 13:879–919
Glover F (1968) Surrogate constraints. Oper Res 16:741–749
Glover F (1975) Surrogate constraint duality in mathematical programming. Oper Res 23:434–451

Glover F (1977) Heuristics for integer programming using surrogate constraints. Decis Sci 8(1):156–166
Glover F (1989) Tabu search—part I. ORSA J Comput 1(3):190–206
Glover F (1990) Tabu search—part II. ORSA J Comput 2(1):14–32
Goldberg D (1989) Genetic algorithms in search. Optimization and machine learning. Addison-Wesley Professional, Reading
Greenberg HJ, Pierskalla WP (1970) Surrogate mathematical programming. Oper Res 18:924–939
Greistorfer P, Rego C (2006) A simple filter-and-fan approach to the facility location problem. Comput Oper Res 33:2590–2601
Guastaroba G, Speranza MG (2012) Kernel search: an application to the index tracking problem. Eur J Oper Res 217(1):54–68
Guignard M, Kim S (1987) Lagrangean decomposition: a model yielding stronger Lagrangean bounds. Math Program 39:215–228
Gunawan A, Ming Ng K, Leng Poh K (2012) A hybridized Lagrangian relaxation and simulated annealing method for the course timetabling problem. Comput Oper Res 39(12):3074–3088
Hewitt M, Nemhauser GL, Savelsbergh MWP (2010) Combining exact and heuristic approaches for the capacitated fixed-charge network flow problem. INFORMS J Comput 22(2):314–325
Hiriart-Urruty JB, Lemarechal C (1993) Convex analysis and minimization algorithms II: advanced theory and bundle methods. A series of comprehensive studies in mathematics, 306. Springer, Berlin
Holland JH (1975) Adaptation in natural and artificial systems. MIT Press, Cambridge
Holmberg K, Ling J (1997) A Lagrangean heuristic for the facility location problem with staircase costs. Eur J Oper Res 97(1):63–74
Hoos H, Stützle T (2004) Stochastic local search-foundations and applications. Morgan Kaufmann, San Francisco
Hu B, Raidl GR (2006) Variable neighborhood descent with self-adaptive neighborhood ordering. In: Proceedings of the 7th EU/ME meeting on adaptive, self-adaptive and multi-level metaheuristics
Hu B, Leitner M, Raidl GR (2008) Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. J Heuristics 14(5):473–499
Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of ICNN'95—international conference on neural networks, vol 4, pp 1942–1948
Kirkpatrick S, Gelatt C, Vecchi M (1983) Optimization by simulated annealing. Science 220:671–680
Lopes R, Morais VW, Noronha TF, Souza V (2015) Heuristics and matheuristics for a real-life machine reassignment problem. Int Trans Oper Res 22:77–95
Lorena L, Belo Lopes F (1994) A surrogate heuristic for set covering problems. Eur J Oper Res 79(1):138–150
Lourenço HR, Martin O, Stützle T (2002) Iterated local search. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics. International series in operations research and management science. Kluwer Academic Publishers, New York, pp 321–353
Lourenço HR, Martin O, Stützle T (2010) Iterated local search: framework and applications. In: Gendreau M, Potvin J (eds) Handbook of metaheuristics, vol 146, 2nd edn. International series in operations research and management science. Springer, New York, pp 363–397. ISBN: 978-1-4419-1663-1
Lowerre B (1976) The HARPY speech recognition system. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA
Maniezzo V (1999) Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. INFORMS J Comput 11(4):358–369
Maniezzo V (2006) Matheuristics 2006 conference web portal. http://astarte.csr.unibo.it/Matheuristics2006/
Maniezzo V, Stützle T (2020) Special issue: matheuristics and metaheuristics. Int Trans Oper Res 27:1
Maniezzo V, Stützle T, Voß S (2009) Matheuristics: hybridizing metaheuristics and mathematical programming. Annals of information systems, 10. Speringer, Berlin
Maniezzo V, Boschetti M, Gutjahr W (2020) Stochastic premarshalling of block stacking warehouses. Omega. https://doi.org/10.1016/j.omega.2020.102336
Maniezzo V, Boschetti M, Stuezle T (2021) Matheuristics: algorithms and implementations. EURO advanced tutorials on operational research. Springer, Berlin
Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E (1953) Equation of state calculations by fast computing machines. J Chem Phys 21(6):1087–1092
Meyers C, Orlin JB (2006) Very large-scale neighborhood search techniques in timetabling problems. In: Burke EK, Rudová H (eds) Proceedings of the 6th international conference on practice and theory of automated timetabling VI (PATAT'06). Springer, Berlin, Heidelberg, pp 24–39

Mingozzi A, Boschetti MA, Ricciardelli S, Bianco LA (1999) Set partitioning approach to the crew scheduling problem. Oper Res 47:873–888

Mitrović-Minić S, Punnen AP (2008) Very large-scale variable neighborhood search for the generalized assignment problem. J Interdiscip Math 11(5):653–670

Mitrović-Minić S, Punnen AP (2009) Variable intensity local search. In: Maniezzo V, Stützle T, Voß S (eds) Matheuristics: hybridizing metaheuristics and mathematical programming, vol 10. Annals of information systems. Springer, Boston, pp 245–252

Mladenovic N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24(11):1097–1100

Narciso M, Lorena L (1999) Lagrangean/surrogate relaxation for generalized assignment problems. Eur J Oper Res 114(1):165–177

Ngueveu SU, Prins C, Wolfler R (2009) A hybrid tabu search for the m-peripatetic vehicle routing problem. In: Maniezzo V, Stützle T, Voß S (eds) Matheuristics, vol 10. Annals of information systems. Springer, Boston

Nishi T, Okura T, Lalla-Ruiz E, Voß S (2020) A dynamic programming-based matheuristic for the dynamic berth allocation problem. Ann Oper Res 286:391–410

Ow P, Morton T (1988) Filtered beam search in scheduling. Int J Prod Res 26:297–307

Pirkwieser S, Raidl GR (2010) Variable neighborhood search coupled with ILP-based very large neighborhood searches for the (periodic) location-routing problem. In: Blesa M, Blum C, Raidl G, Roli A, Sampels M (eds) Hybrid metaheuristics, vol 6373. HM 2010. Lecture notes in computer science. Springer, Berlin, pp 174–189

Pisinger D, Ropke S (2010) Large neighborhood search. In: Gendreau M, Potvin J (eds) Handbook of metaheuristics, vol 146. International series in operations research and management science. Springer, Boston, pp 399–419

Polyak B (1969) Minimization of unsmooth functionals. USSR Comput Math Math Phys 9(3):14–29

Prandtstetter M, Raidl GR (2008) An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem. Eur J Oper Res 191(3):1004–1022

Puchinger J, Raidl GR (2008) Bringing order into the neighborhoods: relaxation guided variable neighborhood search. J Heuristics 14(5):457–472

Raidl G (2015) Decomposition based hybrid metaheuristics. Eur J Oper Res 244:66–76

Reimann M (2007) Guiding ACO by problem relaxation: a case study on the symmetric TSP. In: Bartz-Beielstein T et al (eds) Hybrid metaheuristics, vol 4771. HM 2007, Lecture notes in computer science. Springer, Berlin, pp 45–56

Roli A, Benedettini S, Stützle T, Blum C (2012) Large neighbourhood search algorithms for the founder sequence reconstruction problem. Comput Oper Res 39:213–224

Ropke S, Pisinger D (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp Sci 40(4):455–472

Salari M, Toth P, Tramontani A (2010) An ILP improvement procedure for the open vehicle routing problem. Comput Oper Res 37(12):2106–2120

Sanchez I, Mora J, Santos C, Gonzalez-Mendoza M, Montiel Moctezuma C (2018) Solving binary cutting stock with matheuristics using particle swarm optimization and simulated annealing. Soft Comput 22(18):41–53

Sarvanov V.I, Doroshko NN (1981) Approximate solution of the traveling salesman problem by a local algorithm with scanning neighborhoods of factorial cardinality in cubic time. Softw Algorithms Programs Math Inst Beloruss Acad Sci Minsk 31:11–13

Schmid V, Doerner KF, Hartl RF, Salazar-González JJ (2010) Hybridization of very large neighborhood search for ready-mixed concrete delivery problems. Comput Oper Res 37(3):559–574

Senne ELF, Lorena LAN (2000) Lagrangean/surrogate heuristics for p-median problems. In: Laguna M, Gonzalez-Velarde JL (eds) Computing tools for modeling, optimization and simulation: interfaces in computer science and operations research. Kluwer Academic Publishers, New York, pp 115–130

Shor N, Kiwiel K, Ruszcaynski A (1985) Minimization methods for non-differentiable functions. Springer, New York

Sniedovich M, Voß S (2006) The corridor method. A dynamic programming inspired metaheuristic. Control Cybern 35(3):551–578

Sörensen K (2015) Metaheuristics—the metaphor exposed, international transactions in operational research. Special Issue Matheuristics Model-Based Metaheuristics 22(1):3–18

Sourd F (2006) Dynasearch neighborhood for the earliness-tardiness scheduling problem with release dates and setup constraints. Oper Res Lett 34(5):591–598

Sridharan R (1991) A Lagrangian heuristic for the capacitated plant location problem with single source constraints. Eur J Oper Res 66:305–312

Thompson PM, Psaraftis HN (1993) Cyclic transfer algorithms for multivehicle routing and scheduling problems. Oper Res 41:935–946

Umetani S, Yagiura M, Ibaraki T (2003) One-dimensional cutting stock problem to minimize the number of different patterns. Eur J Oper Res 146(2):388–402

Wolsey L (2020) Integer programming, vol 2. Wiley, Hoboken

Yaghini M, Karimi M, Rahbar MA (2013) Hybrid metaheuristic approach for the capacitated p-median problem. Appl Soft Comput 13(9):3922–3930

Yagiura M, Ibaraki T (1996) The use of dynamic programming in genetic algorithms for permutation problems. Eur J Oper Res 92:387–401

Zanotti R, Mansini R, Ghiani G, Guerriero E (2019) A Kernel search approach for the time-dependent rural postman problem. In: WARP3, 3rd International workshop on arc routing problems. Pizzo (Calabria, Italy)

Zhang Y, Chu F, Che A, Yu Y, Feng X (2019) Novel model and kernel search heuristic for multi-period closed-loop food supply chain planning with returnable transport items. Int J Prod Res 57(23):7439–7456