MetaBoosting: Enhancing Integer Programming Techniques by Metaheuristics

Jakob Puchinger[1], Günther R. Raidl[2], and Sandro Pirkwieser[2]

[1] arsenal research
Vienna, Austria
jakob.puchinger@arsenal.ac.at
[2] Institute of Computer Graphics and Algorithms,
Vienna University of Technology, Vienna, Austria
{raidl|pirkwieser}@ads.tuwien.ac.at

Summary. This chapter reviews approaches where metaheuristics are used to boost the performance of exact integer linear programming (IP) techniques. Most exact optimization methods for solving hard combinatorial problems rely at some point on tree search. Applying more effective metaheuristics for obtaining better heuristic solutions and thus tighter bounds in order to prune the search tree in stronger ways is the most obvious possibility. Besides this, we consider several approaches where metaheuristics are integrated more tightly with IP techniques. Among them are collaborative approaches where various information is exchanged for providing mutual guidance, metaheuristics for cutting plane separation, and metaheuristics for column generation. Two case studies are finally considered in more detail: (i) a Lagrangian decomposition approach that is combined with an evolutionary algorithm for obtaining (almost always) proven optimal solutions to the knapsack constrained maximum spanning tree problem and (ii) a column generation approach for the periodic vehicle routing problem with time windows in which the pricing problem is solved by local search based metaheuristics.

1 Introduction

When considering optimization approaches that combine aspects from metaheuristics with mathematical programming techniques, the resulting hybrid system may either be of *exact* or *heuristic* nature. Exact approaches are guaranteed to yield proven optimal solutions when they are given enough computation time. In contrast, heuristics only aim at finding reasonably good approximate solutions usually in a more restricted time; performance guarantees are typically not provided. Most of the existing hybrid approaches are of heuristic nature, and mathematical programming techniques are used to boost the performance of a metaheuristic. Exploiting solutions to exactly solvable relaxations of the original problem, or searching large neighborhoods by means of mathematical programming techniques are examples for such approaches; see also Chapter **??**. On the other hand, there are also several

highly successful ways to exploit metaheuristic strategies for enhancing the performance of mathematical programming techniques, and often these methods retain their exactness. We refer to such improvement techniques as *MetaBoosting* and study them in detail in the present chapter.

Most exact approaches for solving hard *combinatorial optimization problems* (COPs) are based on a *tree search*, where the search space is recursively partitioned in a divide-and-conquer manner into mutually disjoint subspaces by fixing certain variables or imposing additional constraints. In a naive enumeration tree each subspace is further divided as long as it contains more than one feasible solution. Obviously, the size of such a naive search tree increases rapidly with the problem size, and naive enumeration is therefore inefficient. The key to successfully approach larger problem instances is to have some mechanism for substantially pruning the search tree. This is usually done by identifying subspaces that need not to be further pursued, as they cannot contain a feasible solution that is better than a solution already found before. The scalability of a tree search thus depends essentially on the efficiency of this pruning mechanism.

In *branch-and-bound* (B&B), upper and lower bounds are determined for the objective values of solutions, and subspaces for which the lower bounds exceed the upper bounds are discarded. Considering a minimization problem, any feasible solution provides a (global) upper bound. Thus, any (meta-)heuristic that is able to determine good heuristic solutions in reasonable time may be an essential help in B&B for pruning the search tree, even when the heuristic itself does not provide any performance guarantee.

Applying an effective metaheuristic to obtain better upper bounds for B&B is the most obvious way how one can boost the performance of an exact optimization technique by means of a metaheuristic. When considering established *integer (linear) programming* (IP) techniques including cutting plane methods, column generation, and diverse variants of relaxation based approaches in more detail, we can observe several further possibilities for exploiting the strengths of metaheuristics.

The next section will introduce our basic notations and briefly review important IP techniques. In Sections 3 to 5 we describe various successful MetaBoosting strategies. Two exemplary case studies are presented together with some practical results in more detail in Sections 6 and 7: First, we consider a Lagrangian decomposition/evolutionary algorithm hybrid for the knapsack constrained maximum spanning tree problem, and second, a column generation approach that uses metaheuristics for solving the pricing problem is discussed for the periodic vehicle routing problem with time windows. Conclusions are drawn in Section 8.

2 Integer Programming Techniques

This section introduces some basic notations and gives a short introduction into prominent IP techniques. For an in-depth coverage of the subject we refer to the books on linear optimization by Bertsimas and Tsitsiklis [6] and on combinatorial and integer optimization by Nemhauser and Wolsey [37] and Wolsey [53].

We consider IP problems of the form

$$z_{\text{IP}} = \min\{cx \mid Ax \geq b, x \geq 0, x \in \mathbb{Z}^n\}, \tag{1}$$

where x is an n-dimensional integer variable vector in column form and $c \in \mathbb{Q}^n$ an n-dimensional row vector. Their dot-product cx is the *objective function* that should be minimized. Matrix $A \in \mathbb{Q}^{m \times n}$ and the m-dimensional column vector $b \in \mathbb{Q}^m$ together define m inequality constraints. A *mixed integer program* (MIP) would involve a combination of integer and real-valued variables.

Maximization problems can be transformed into minimization problems by simply changing the sign of c. Less-than constraints are similarly brought into greater-than-or-equal form by changing the sign of the corresponding coefficients, and equalities can be translated to pairs of inequalities. Thus, we can handle all kinds of linear constraints by appropriate transformations. Without loss of generality, we may therefore restrict our following considerations to minimization problems of this standard form.

2.1 Relaxations and Duality

One of the most important concepts in integer programming are *relaxations*, where some or all constraints of a problem are loosened or omitted. Relaxations are mostly used to obtain related, simpler problems that can be solved efficiently yielding bounds and approximate (not necessarily feasible) solutions for the original problem. Embedded within a B&B framework, these techniques may lead to effective exact solution techniques.

The *linear programming* (LP) relaxation of the IP (1) is obtained by relaxing the integrality constraints, yielding

$$z_{\mathrm{LP}} = \min\{cx \mid Ax \geq b, x \geq 0, x \in \mathbb{R}^n\}. \tag{2}$$

Large instances of such LPs can be efficiently solved using simplex-based or interior-point algorithms. The solution to the LP relaxation provides a lower bound for the original minimization problem, i.e. $z_{\mathrm{IP}} \geq z_{\mathrm{LP}}$, since the search space of the IP is contained within the one of the LP and the objective function remains the same.

We can further associate a *dual problem* to an LP (2), which is defined by

$$w_{\mathrm{LP}} = \max\{ub \mid uA \leq c, u \geq 0, u \in \mathbb{R}^m\} \tag{3}$$

with u being the m-dimensional dual variable row vector. The dual of the dual LP is the original (*primal*) LP again. Important relations between the primal problem and its dual are known as weak and strong duality theorems, respectively:

Weak duality theorem: The value of every finite feasible solution to the dual problem is a lower bound for the primal problem, and each value of a finite feasible solution to the primal problem is an upper bound for the dual problem. As a consequence, if the dual is unbounded, the primal is infeasible and vice versa.

Strong duality theorem: If the primal has a finite optimal solution with value z^*_{LP}, than its dual has the same optimal solution value $w^*_{\mathrm{LP}} = z^*_{\mathrm{LP}}$ and vice versa.

In case of an IP we have to distinguish between weak and strong duals: A *weak dual* of an IP (1) is any maximization problem $w = \max\{w(u) \mid u \in S_{\mathrm{D}}\}$ such that $w(u) \leq cx$ for all $x \in \{Ax \geq b, x \geq 0, x \in \mathbb{Z}^n\}$. An obvious weak dual of (1) is the dual (3) of its LP relaxation (2). A *strong dual* is a weak dual that further has an optimal solution u^* such that $w(u^*) = cx^*$ for an optimal solution x^* of (1). For

solving IPs, weak duals which are iteratively strengthened during the course of the optimization process are often utilized.

Another commonly used relaxation of IPs, which often yields significantly tighter bounds than the LP relaxation, is *Lagrangian relaxation* [20, 21]. Consider the IP

$$z_{\text{IP}} = \min\{cx \mid Ax \geq b, Dx \geq d, x \geq 0, x \in \mathbb{Z}^n\}, \tag{4}$$

where constraints $Ax \geq b$ are "easy" in the sense that the problem can be efficiently solved when the m' "complicating" constraints $Dx \geq b$ are dropped. Simply removing these constraints yields a relaxation, but the resulting bound will usually be weak because of this complete ignorance. In Lagrangian relaxation, constraints $Dx \geq d$ are replaced by corresponding penalty terms in the objective function:

$$z_{\text{LR}}(\lambda) = \min\{cx + \lambda(d - Dx) \mid Ax \geq b, x \geq 0, x \in \mathbb{Z}^n\}. \tag{5}$$

Vector $\lambda \in \mathbb{R}^{m'}$ is the vector of Lagrangian multipliers, and for any $\lambda \geq 0$, $z_{\text{LR}}(\lambda) \leq z_{\text{IP}}$, i.e. we have a valid relaxation of the IP. We are now interested in finding a specific vector λ yielding the best—i.e. largest—possible lower bound, which leads to the *Lagrangian dual problem*

$$z^*_{\text{LR}} = \max_{\lambda \geq 0}\{z_{\text{LR}}(\lambda)\}. \tag{6}$$

This Lagrangian dual is a piecewise linear, convex function which can usually be well solved by iterative procedures like a subgradient method. A more elaborate algorithm that has been reported to converge faster on several problems is the volume algorithm [4], whose name is inspired by the fact that primal solutions are also considered, whose values come from approximating the volumes below active faces of the dual problem.

Given a solution λ to the Lagrangian dual problem (6) and a corresponding optimal solution x^* to the Lagrangian relaxation (5) that is also feasible to the original problem (4), i.e. $Dx^* \geq d$, the following complementary slackness condition holds: x^* is an optimal solution to the original problem (4) if and only if

$$\lambda(d - Dx^*) = 0. \tag{7}$$

Provided the Lagrangian dual problem is solved to optimality, it can be shown that the Lagrangian relaxation always yields a bound that is at least as good as the one of the corresponding linear relaxation.

A third general-purpose relaxation technique for IPs is *surrogate relaxation* [25]. Here, some or all constraints are scaled by surrogate multipliers and cumulated into a single inequality by adding the coefficients. Similarly as in Lagrangian relaxation, the ultimate goal is to find surrogate multipliers yielding the overall best bound. Unfortunately, this surrogate dual problem usually has not such nice properties as the Lagrangian dual problem and solving it is often difficult. However, if one is able to determine optimal surrogate multipliers, the bound obtained for the IP is always at least as good as (and often better than) those obtained from the corresponding linear and Lagrangian relaxations.

2.2 LP-Based Branch-and-Bound

By solving the LP relaxation of an IP we obtain a lower bound on the optimal IP solution value and the solution will in general contain fractional variable values. (If all variable values would be integer, we already would have solved the IP.) The standard way to continue towards an optimal integer solution is the already mentioned B&B. Branching usually takes place over some variable x_i with a fractional LP-value x_i^*, defining as first subproblem the IP with the additional inequality $x_i \leq \lfloor x_i^* \rfloor$ and as second subproblem the IP with inequality $x_i \geq \lceil x_i^* \rceil$. For these subproblems with the additional branching constraints, the LP relaxations are resolved leading to increased lower bounds and eventually solutions where all integer variables have integral values. As mentioned in the introduction, primal heuristics are usually also applied to each subproblem in order to find improved feasible solutions and corresponding global upper bounds, enabling a stronger pruning of the search tree.

2.3 Cutting Plane Algorithm and Branch-and-Cut

When modeling COPs as IPs an important goal is to find a *strong* formulation, for which the solution value of the LP relaxation in general provides a *tight* bound. For many COPs it is possible to strengthen an existing IP formulation significantly by including further inequalities, which would actually be redundant w.r.t. the integer optimum. In general it is even possible to strengthen a model such that the LP relaxation already yields an integer optimum; however, the number of required constraints often grows exponentially with the problem size. Naively solving such an LP by standard techniques might quickly become too costly in practice.

Dantzig et al. [10] proposed the *cutting plane algorithm* for this purpose, which usually only considers a fraction of all constraints explicitly but is nevertheless able to determine an optimal solution to the whole LP.

The cutting plane approach starts by solving a reduced LP consisting of a small subset of initial inequalities only. It then tries to find inequalities that are violated by the obtained solution but are valid for the original problem (i.e. contained in the full LP). These valid inequalities are called *cuts* or *cutting planes*, and they are added to the current reduced LP, which is then resolved. The whole process is iterated until no further cutting planes can be determined. If the algorithm computing the cuts provides a proof that no further violated inequality exists, the final solution is optimal for the original full LP. The subproblem of identifying cuts is called *separation problem*. In practice it is crucial to have an efficient method for separating cuts as usually a significant number of valid inequalities must be derived until the cutting plane algorithm terminates.

From a theoretical point of view it is possible to solve any IP using a pure cutting plane approach with appropriate classes of cuts. There exist generic types of cuts, such as the Chvatal-Gomory cuts [53], which guarantee such a result. In practice, however, it may take a too long time for such a cutting plane approach to converge to the optimum, partly because it is often a hard subproblem to separate effective cuts and partly because of the large number of needed cuts.

The combination of B&B with cutting plane methods yields the highly effective class of *branch-and-cut* algorithms which are widely used. Specialized branch-and-cut approaches have been described for many applications and are known for their effectiveness. Cut separation is usually applied at each node of the B&B tree to

tighten the bounds of the LP relaxation and to exclude infeasible solutions as far as possible.

For cutting plane separation effective heuristic methods come into play once again: For strengthening the LP relaxations it is often sufficient to generate cuts heuristically since the correctness of the final solution does not depend on the generated cuts as long as they are valid. Almost all modern mixed integer programming (MIP) solvers include sophisticated generic cut separation heuristics, and they play a major role in the success of these solvers.

2.4 Column Generation and Branch-and-Price

Often it is possible to model COPs via strong formulations involving a huge number of variables. Dantzig-Wolfe decomposition [11] is a technique for obtaining such models from compact formulations in a systematic way. It replaces the original problem variables by linear combinations of the extreme points and extreme rays of the original search space, yielding a potentially exponential number of new variables. The obtained models can result in much stronger relaxations than their compact counterparts.

Despite the many variables, the LP relaxations of such formulations can often be efficiently calculated. The *column generation* approach starts with only a small subset of all variables (corresponding to columns in the matrix notation of the IP) and solves the corresponding restricted LP relaxation. It is then tried to identify one or more so far ignored variables whose inclusion may lead to an improved solution. This subproblem is called *pricing problem*. For a minimization problem a variable can eventually improve the current LP solution if it has negative reduced costs. After adding such a new variable to the restricted LP, it is resolved and the process iterated until no further variables with negative reduced costs exist. The final solution is an optimal solution for the complete LP.

Column generation can be seen as dual to the cutting plane approach, since inequalities correspond to variables in the dual LP. For a recent review on column generation see [35]. The cutting stock problem is an early example for the successful application of column generation based methods [24]. Every possible cutting pattern is represented by a variable and the pricing problem corresponds to the classical 0–1 knapsack problem, which can be solved efficiently in pseudo-polynomial time.

As the column generation algorithm only solves the LP relaxation, it must in general also be combined with B&B in order to obtain optimal integer solutions. When column generation is performed for each node of the B&B tree, the approach is called *branch-and-price*. One of the main difficulties in the implementation of such methods lies in the development of appropriate branching rules. Furthermore, the individual LPs may sometimes be degenerated, or newly added columns may only improve the solutions marginally leading to many iterations until convergence. In the latter cases, stabilization techniques as discussed in [13] often improve the situation.

Similarly as cutting plane separation may be performed by effective heuristics, one can also heuristically solve the pricing problem in column generation. Care must be taken that in the final iteration it is necessary to prove that no further columns with negative reduced costs exist so that the obtained solution value is guaranteed to be a lower bound for the original IP.

Finally, it occasionally makes sense to combine a cutting plane approach with column generation and embed both in B&B. Such methods, called *branch-and-cut-and-price*, are sometimes extremely successful but are typically also rather complex and highly specialized.

3 Metaheuristics for Finding Primal Bounds

Branch-and-bound based approaches rely on tight primal bounds that are most commonly obtained from feasible solutions. Obviously, heuristics and metaheuristics can be applied to the original problem before starting the B&B process, providing initial solutions. The search space of the exact method is immediately reduced, usually improving overall computation times. Such an approach has the practical advantage of also providing feasible solutions at an early stage of the optimization process.

Furthermore (meta-)heuristics can be repeatedly applied throughout the whole tree search, providing possibly improved solutions. Again, this can speed up the overall optimization essentially by further pruning the search tree. Even the optimal solution might be discovered by one of those heuristics. On the other hand, when heuristics are applied too often and have rather long run-times, they might slow down the overall process. Thus, an appropriate balance is required.

3.1 Initial Solutions

Generic MIP based heuristics for computing initial solutions are widely used. They range from early heuristics such as described in [2, 30] over pivot and complement [3] to the recent feasibility pump [17, 5], which is also discussed in Chapter **??** of this book. The major commercial generic MIP solvers such as CPLEX[3] or XPRESS MP[4] have very strong heuristics for finding initial feasible solutions, often outperforming simple problem-specific heuristics in terms of solution quality and speed. Unfortunately, not much is publicly known about these heuristics.

An interesting approach specifically tailored to the *multidimensional knapsack problem* (MKP) involving metaheuristics is presented in Vimont et al. [52]. The MKP can be defined by the following IP:

$$\text{(MKP)} \qquad \text{maximize} \quad z = \sum_{j=1}^{n} p_j x_j \tag{8}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_{ij} x_j \leq c_i, \quad i = 1, \ldots, m, \tag{9}$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n. \tag{10}$$

A set of n items with profits $p_j > 0$ and m resources with capacities $c_i > 0$ are given. Each item j consumes an amount $w_{ij} \geq 0$ from each resource i. Variables x_j

[3] http://www.ilog.com
[4] http://www.dashoptimization.com

indicate which items are selected. The objective is to choose a subset of items with maximum total profit that does not violate any of the capacity constraints (9).

An exact algorithm based on implicit enumeration and reduced cost propagation is applied. The enumeration algorithm tries to first handle the unpromising parts of the search space, with the goal of reducing it substantially. After computing an initial solution yielding a lower bound, the problem is first partitioned by fixing the number of selected items to certain values [50]. Each of the resulting subproblems is then explored by B&B with a special branching strategy based on the solution to the LP relaxation and reduced costs at each search tree node.

The search space is further reduced by fixing some variables using a propagation mechanism. It is based on the reduced cost constraint originally described in [38]. After solving the LP relaxation yielding a solution ($\overline{x}$), the following reduced cost inequality can be devised:

$$\sum_{j:\overline{x}_j=0} |\overline{c}_j| x_j + \sum_{j:\overline{x}_j=1} |\overline{c}_j|(1-x_j) \leq \mathrm{UB} - \mathrm{LB}, \tag{11}$$

where $\overline{c}$ is the reduced cost vector corresponding to $\overline{x}$ and LB is a primal lower bound, typically the objective value of a feasible solution.

This approach relies heavily on tight primal bounds, since constraint (11) becomes tighter with increasing values of LB. These bounds come from a sophisticated tabu search based hybrid algorithm described in [50]. The search space is partitioned via additional constraints fixing the total number of items to be packed. Lower and upper bounds for the number of items are calculated by solving modified LP relaxations of the original MKP. For each remaining partition of the search space, tabu search is independently applied, starting with a solution derived from the LP relaxation of the partial problem. The whole tabu search approach has further been improved in [51] by additional variable fixing.

This example demonstrates that a combination of highly developed specialized methods for computing bounds with the aid of a metaheuristic, generating dependent cuts, and guiding the search is sometimes able to achieve exceedingly good results.

3.2 B&B Acting as Local Search Based Metaheuristic

Fischetti and Lodi proposed *local branching* as an extension for generic branch-and-cut based MIP solvers with the aim of producing good heuristic solutions early during the exact tree search [18]. Local branching introduces the spirit of classical k-opt local search in B&B by modifying the branching rule and the strategy for choosing the next tree node to process. Let us consider MIPs with 0–1 variables; let $x = (x_1, \ldots, x_n)$ be the variable vector and $\mathcal{B} \subseteq \{1, \ldots, n\}$ be the index set of the 0–1 variables. A k-opt neighborhood around a given incumbent solution $\overline{x} = (\overline{x}_1, \ldots, \overline{x}_n)$ can be defined by the *local branching constraint*

$$\Delta(x, \overline{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \overline{S}} x_j \leq k, \tag{12}$$

where $\overline{S}$ corresponds to the index set of the 0–1 variables that are set to one in the incumbent solution, i.e. $\overline{S} = \{j \in \mathcal{B} \mid \overline{x}_j = 1\}$. $\Delta(x, \overline{x})$ resembles the classical Hamming distance between x and $\overline{x}$ for integer values.

Starting from an initial solution, the search space is partitioned into the k-opt neighborhood of this incumbent and the remaining part of the search space by applying the local branching constraint and its inverse $\Delta(x, \overline{x}) \geq k + 1$, respectively. The MIP solver is then forced to find the best solution in the k-opt neighborhood first. If an improved solution x' has been found, a new subproblem $\Delta(x, \overline{x}')$ corresponding to the search of the k-opt neighborhood of this new incumbent is split off the remaining search space and solved in the same way; otherwise a larger k may be tried. The process is repeated until no further improvement can be achieved. Finally, the remaining problem corresponding to all yet unconsidered parts of the search space is processed in a standard way.

This basic mechanism is extended by introducing time limits, automatically modifying the neighborhood size k, and adding diversification strategies to improve performance. An extension of the branching constraint for general integer variables is also described. Results on various MIP benchmark instances using CPLEX as MIP solver indicate the advantages of the approach in terms of an earlier identification of high-quality solutions.

Hansen et al. [27] suggest a variant of local branching which follows more closely the classical variable neighborhood search metaheuristic for choosing the next k-opt neighborhood to process. Improved results are reported. Fischetti et al. [19] describe another variant of the original local branching where they consider problems in which the set of variables naturally partitions into two levels and fixing the first-level variables to some values yields substantially easier subproblems.

Danna et al. [9] suggest a different approach called *relaxation induced neighborhood search* (RINS) for exploring the neighborhoods of incumbent solutions more intensively. The central idea is to occasionally devise a sub-MIP at a node of the B&B tree that corresponds to a special neighborhood of an incumbent solution: Variables having the same values in the incumbent and in the current solution of the LP relaxation are fixed, and an objective cutoff is set based on the objective value of the incumbent. A sub-MIP is solved on the remaining variables with a given time limit. If a better solution can be found it is passed to the global MIP-search, which is resumed after the sub-MIP's termination. In the authors' experiments, CPLEX is used as MIP solver, and RINS is compared to standard CPLEX, local branching, combinations of RINS and local branching, and guided dives. Results indicate that RINS often performs best. CPLEX includes RINS as a standard strategy for quickly obtaining good heuristic solutions since version 10. Local branching constraints are said to be often less effective as they are dense inequalities involving all integer variables. In particular, adding the inverse local branching constraints of already searched k-opt neighborhoods to the remaining problem is found to be disadvantageous as the reduced node processing throughput caused by the series of these dense constraints outweighs the benefit of avoiding redundant exploration of parts of the search space.

Recently Ghosh [23] proposed a *distance induced neighborhood search* (DINS). It is conjectured that better MIP solutions are more likely to be close to the solution of the LP relaxation than farther away. Hence, an appropriate distance metric is utilized. DINS combines soft fixing of variables as in local branching as well as hard fixing of variables as in RINS, plus an additional rebounding procedure, which adapts the lower and upper bounds of selected variables. Experimental results indicate that DINS outperforms both local branching and RINS; DINS is also integrated now in CPLEX.

3.3 Solution Merging

In *solution merging* new, possibly better solutions are created from attributes appearing in two or more promising heuristic solutions. Such an approach is based on the assumption that high-quality solutions often share many attributes.

Recombination, the primary variation operator in genetic algorithms, can be seen as a classical solution merging approach. Usually, two parent solutions are selected and an offspring is derived by simple random inheritance of parental attributes. Classical recombination operations do not try to optimize this offspring, which therefore often is worse than its parents. However, these operations are computationally cheap and can be repeated many times in order to achieve improvements.

Alternatively, one can put more effort into the derivation of such offspring. A sometimes effective technique is *path relinking* [26], which traces a path in the search space from one parent to a second by repeatedly exchanging a single attribute only (or more generally by performing a series of moves in a simple neighborhood structure). An overall best solution found on this path is finally taken as offspring.

This idea can further be extended by considering not just solutions on a single path between two parents, but the whole subspace of solutions induced by the joined attributes appearing in a set of two or more input solutions. An *optimal merging* operation returns a best solution from this subspace, i.e. it identifies a best possible combination of the parents' attributes. Depending on the underlying problem, identifying such an optimal offspring is often a hard optimization problem on its own, but due to the usually quite limited number of different attributes appearing in the parents, it can often be solved in reasonable time in practice.

For mixed integer programming, Rothberg [47] suggests a tight integration of an evolutionary algorithm (EA) including optimal merging in a branch-and-cut based MIP solver. In regular intervals the evolutionary algorithm is applied as B&B tree node heuristic. The population of the EA consists of the best non-identical solutions found so far, which have either been discovered by the MIP tree search or by previous iterations of the EA.

Mutation selects one parent, fixes a randomly chosen subset of variables, and calls the MIP solver for determining optimal values for the remaining problem. Since the number of variables to be fixed is a critical parameter, an adaptive scheme is applied to control it. In contrast to classical EAs, mutation is performed before recombination on a fixed number of randomly chosen solutions, since at the beginning of the optimization only one or very few solutions will be in the population.

Recombination is performed by first fixing all variables that have the same values in two selected parental solutions and applying the MIP solver to this reduced subproblem. The exploration of this subproblem is eventually truncated when a given node-limit is exceeded. New high-quality solutions discovered during this search are added to the population. This recombination is further generalized to more than two parents by fixing variable values that are identical in all of them.

The applied selection strategy simply chooses the first parent from the population at random, and the second is then chosen randomly amongst the solutions with a better objective value than the first one. This guarantees a certain bias towards better solutions. For mutation the same mechanism is used, but only the second solution is used.

Experimental results indicate that this hybrid often is able to find significantly better solutions than other heuristic methods for several very difficult MIPs. The method is integrated in the commercial MIP solver CPLEX since version 10.

3.4 Metaheuristics and Lagrangian Relaxation

As mentioned in Section 2.1, Lagrangian relaxations may sometimes yield substantially tighter lower bounds than simpler LP relaxations. Furthermore, heuristic solutions and, thus, upper bounds are often either automatically obtained as intermediate by-products from the subgradient procedure or by applying typically rather simple Lagrangian heuristics such as rounding or repairing procedures. When embedded in a B&B framework, such Lagrangian relaxation based methods are frequently turned into highly successful exact optimization approaches.

To further improve performance by obtaining better upper bounds, more sophisticated metaheuristics may be applied in combination with Lagrangian relaxation. For example, a well-working hybrid of a Lagrangian relaxation approach and variable neighborhood descent has recently been described for a real-world fiber optic network design problem in Leitner and Raidl [33].

An interesting additional aspect of such combinations is that also the metaheuristic may benefit by exploiting diverse intermediate results from the subgradient search. A successful example for this is the hybrid Lagrangian genetic algorithm (GA) for the prize collecting Steiner tree problem proposed by Haouari and Siala [28]. They apply a Lagrangian relaxation on a minimum spanning tree formulation of the prize collecting Steiner tree problem and use the volume algorithm for solving the Lagrangian dual. After termination, the GA is started on a reduced problem, consisting only of the edges appearing in all the intermediate trees derived by the volume algorithm. Furthermore, some of the GA's initial solutions are derived from the volume algorithm's intermediate reduced edge costs by applying a greedy Lagrangian heuristic. Last but not least, the GA uses a modified objective function: Instead of the original costs, the reduced costs that are finally obtained by the volume algorithm are used; in this way, the metaheuristic search is guided into regions of the search space deemed promising by the Lagrangian relaxation.

The authors of the present chapter describe a similar approach for the knapsack constrained maximum spanning tree problem in [40]. Section 6 summarizes this work as an exemplary case study.

4 Collaborative Hybrids

In collaborative combinations of different types of optimization techniques, the algorithms exchange information but are not part of each other; i.e. there is no clear master containing the other method(s) as subprocedures [42]. The individual algorithms may be executed sequentially, intertwined, or in a parallel way and exchange information for guidance. In principle, any metaheuristic that provides incumbent solutions to a B&B-based approach might already be considered to fall into this class of approaches. The above mentioned hybrid Lagrangian relaxation approach from Haouari and Siala can, for example, also be regarded a sequential collaborative combination, where the Lagrangian relaxation provides guidance for the GA.

Intertwined and parallel combinations allow for *mutual* guidance, i.e., all participating methods may exploit information from each other. Talukdar et al. [49] describe a very general agent-based model for such systems, called *asynchronous teams* (A-Teams). This problem solving architecture consists of a collection of agents and memories connected in a strongly cyclic directed way, and each optimization agent works on the target problem, a relaxation, or a subclass of the original problem. Denzinger and Offerman [12] describe a similar framework called TECHS (TEams for Cooperative Heterogeneous Search). It consists of teams of one or more agents using the same search paradigm. Communication between the agents is controlled by so-called send- and receive-referees.

A specific example for a successful intertwined collaboration of an EA and the branch-and-cut based MIP solver XPRESS MP is the hybrid algorithm from French et al. [22] for solving general IPs. It starts with a branch-and-cut phase, in which information from the B&B tree nodes is collected in order to derive candidate solutions that are added to the originally randomly initialized EA-population. When a certain criterion is satisfied, the EA takes over for some time using the augmented initial population. After termination of the EA, its best solutions are passed back and grafted onto the B&B tree. Full control is given back to branch-and-cut after the newly added nodes had been examined to a certain degree. Reported results on instances of the maximum satisfiability problem show that this hybrid yields better solutions than XPRESS MP or the EA alone.

Another cooperative approach involving a memetic algorithm and branch-and-cut has been described by Puchinger et al. [44] for the MKP. Both methods are performed in parallel and exchange information in a bidirectional asynchronous way. In addition to promising primal solutions, the memetic algorithm also receives dual variable values of certain LP relaxations and uses them for improving its repair and local improvement functions by updating the items' pseudo-utility ratios. Results that are often better than those from [50] and partly competitive to those from [51] have been obtained.

5 Metaheuristics for Cut and Column Generation

As already pointed out in Section 2, in cut and column generation based IP methods the dynamic separation of cutting planes and the pricing of columns can be done by means of (meta-)heuristics in order to speed up the optimization process. Such approaches are reviewed in more detail in the following two sections.

5.1 Cut Separation

In branch-and-cut algorithms inequalities that are satisfied by feasible integer solutions but are violated by the current solution to the LP relaxation have to be derived quickly. Of course, the cuts one wants to find should be *strong* in the sense that they cut away "large" portions of the search space, leading to a significant increase of the LP solution value and thus to relatively few iterations until convergence of the cutting plane algorithm. As many classes of strong cuts are difficult to separate, heuristic separation procedures are commonly applied. More sophisticated metaheuristics, however, have so far only rarely been used for this purpose.

A reason might be the usually large number of cuts that must be generated, and hence the strong requirements w.r.t. speed. Nevertheless, there exist some examples of successful metaheuristic cut separation approaches.

Augerat et al. [1] consider a capacitated vehicle routing problem and describe a branch-and-cut algorithm in which a sequence of methods consisting of a simple construction heuristic, a randomized greedy method, and a tabu search is used for separating capacity constraints. The approach starts with the fastest simple heuristic and switches to the next, more complex strategy as long as no valid cutting plane could be found.

Another example is the branch-and-cut algorithm by Gruber and Raidl for the bounded diameter minimum spanning tree problem described in detail in Chapter **??** of this book. The diameter bound is ensured via an exponentially large number of so-called jump inequalities. Again, a sequence of methods is used for their separation, starting from a greedy construction technique over a local search procedure to a tabu search algorithm. On several benchmark instances, this algorithm outperforms other state-of-the-art IP approaches for this problem, and some larger instances than before could be solved to proven optimality.

Rei et al. [46] describe the acceleration of Benders decomposition by local branching. The basic principle of Benders decomposition is to project a MIP into the space of complicating integer variables only; continuous variables and the constraints involving them are replaced by corresponding constraints on the integer variables. These constraints, however, are not directly available but need to be dynamically created. According to the classical method, an optimal solution to the relaxed master problem (including only the already separated cuts) is needed and an LP involving this solution must be solved in order to separate a single new cut. Rei et al. improved this method by introducing phases of local branching on the original problem in order to obtain multiple feasible heuristic solutions. These solutions provide improved upper bounds and further allow to derive multiple additional cuts before the relaxed master problem needs to be resolved.

5.2 Column Generation

In column generation based algorithms the pricing problem often is difficult by itself, and applying fast (meta-)heuristics can be a meaningful option. It can be beneficial for the overall performance if most of the columns are heuristically derived.

Filho and Lorena [16] apply a heuristic column generation approach to graph coloring. A GA is used to generate initial columns and to solve the pricing problem at every iteration. Column generation is performed as long as the GA finds columns with negative reduced costs. The master problem is solved using CPLEX.

Puchinger and Raidl [41, 43] describe an exact branch-and-price algorithm for the three-stage two-dimensional bin packing problem. Rectangular items have to be orthogonally packed into the least number of larger rectangles of fixed size, and only non-overlapping three-stage guillotine packing patterns are allowed. The pricing problem occurring in this application is a three-stage two-dimensional knapsack packing problem. Fast column generation is performed by applying a sequence of four methods: (i) a greedy heuristic, (ii) an evolutionary algorithm, (iii) solving a restricted, simpler IP-model of the pricing problem using CPLEX within a certain time-limit, and finally (iv) solving a complete IP-model by CPLEX. The algorithms

coming later in this sequence are only executed if the previous ones did not find columns with negative reduced costs. The greedy heuristic is based on the classical finite first fit heuristic but is adapted to consider additional constraints introduced by the branching decisions during the search process of the branch-and-price algorithm. The EA uses a direct set-based representation for solutions making it possible to ignore the order of the items to be packed and therefore avoiding redundancies introduced by many symmetries. Specific recombination and mutation operators were developed for this problem. The presented computational experiments show that each pricing algorithm contributes essentially to the whole column generation process. Applied to large problem instances with limited run-time, better solutions are often obtained by the sequential pricing compared to using just one strategy. It is conjectured that also in other applications such combinations of multiple (meta-)heuristic and exact pricing algorithms may be beneficial.

6 Case Study: A Lagrangian Decomposition/EA Hybrid

This first case study demonstrates a combination of a Lagrangian decomposition approach with an evolutionary algorithm (EA) for the knapsack constrained maximum spanning tree problem. The EA exploits information of the Lagrangian decomposition and improves previously obtained primal solutions. Proven optimal solutions are obtained in most cases, especially also on large problem instances. More details on this work can be found in [40].

6.1 The Knapsack Constrained Maximum Spanning Tree Problem

The *knapsack constrained maximum spanning tree* (KCMST) problem arises in practical situations where the aim is to design a most profitable communication network under a strict limit on total costs, e.g. for cable laying or similar resource constraints. The problem is also referred to as budget or side constrained minimum spanning tree problem and is NP-hard [54].

It is defined on an undirected connected graph $G = (V, E)$ with node set V and edge set $E \subseteq V \times V$ representing all possible connections. Each edge $e \in E$ has associated a weight $w_e \in \mathbb{Z}_+$ (corresponding to costs) and a profit $p_e \in \mathbb{Z}_+$. In addition, a weight limit (capacity) $c > 0$ is specified. We seek a spanning tree $G_T = (V, T)$, $T \subseteq E$, on G that maximizes the total profit $\sum_{e \in T} p_e$ and where weight $\sum_{e \in T} w_e$ does not exceed c. By introducing binary variables x_e, $\forall e \in E$, indicating which edges are part of the solution, i.e. $x_e = 1 \leftrightarrow e \in T$ and $x_e = 0$ otherwise, the problem can be stated as:

$$\text{(KCMST)} \quad \max \quad p(x) = \sum_{e \in E} p_e x_e \tag{13}$$

$$\text{s.t.} \quad x \text{ represents a spanning tree on } G, \tag{14}$$

$$\sum_{e \in E} w_e x_e \leq c, \tag{15}$$

$$x_e \in \{0, 1\}, \qquad \forall e \in E. \tag{16}$$

Obviously, the problem represents a combination of the classical minimum spanning tree problem (with changed sign in the objective function) and the 0–1 knapsack problem due to constraint (15). Yamada et al. [54] proposed a straight-forward Lagrangian relaxation where the knapsack constraint is relaxed and primal solutions are improved by local search. We enhance this approach in the following.

6.2 Lagrangian Decomposition of the KCMST Problem

The aforementioned natural combination lends itself to obtain tighter upper bounds via *Lagrangian decomposition* (LD), which is a special variant of Lagrangian relaxation that can be meaningful when there is evidence of two or possibly more intertwined subproblems, and each of them can be efficiently solved on its own by specialized algorithms.

For this purpose, we duplicate variables x_e, $\forall e \in E$, by introducing new, corresponding variables y_e and including linking constraints, leading to the following reformulation:

$$\max \quad p(x) = \sum_{e \in E} p_e x_e \tag{17}$$

$$\text{s.t.} \quad x \text{ represents a spanning tree on } G, \tag{18}$$

$$\sum_{e \in E} w_e y_e \leq c, \tag{19}$$

$$x_e = y_e, \qquad \forall e \in E, \tag{20}$$

$$x_e, y_e \in \{0,1\}, \qquad \forall e \in E. \tag{21}$$

Now we relax the linking constraints (20) in a Lagrangian fashion using Lagrangian multipliers $\lambda_e \in \mathbb{R}$, $\forall e \in E$, hence obtaining the Lagrangian decomposition of the original problem, denoted by KCMST-LD(λ):

$$\max \quad p(x) = \sum_{e \in E} p_e x_e - \sum_{e \in E} \lambda_e (x_e - y_e) \tag{22}$$

$$\text{s.t.} \quad x \text{ represents a spanning tree on } G, \tag{23}$$

$$\sum_{e \in E} w_e y_e \leq c, \tag{24}$$

$$x_e, y_e \in \{0,1\}, \qquad \forall e \in E. \tag{25}$$

Stating KCMST-LD(λ) in a more compact way and emphasizing the now independent subproblems yields

$$\text{(MST)} \quad \max \ \{(p-\lambda)^T x \mid x \mathrel{\hat{=}} \text{a spanning tree on } G,\ x \in \{0,1\}^E\} \ + \tag{26}$$

$$\text{(KP)} \quad \max \ \{\lambda^T y \mid w^T y \leq c,\ y \in \{0,1\}^E\}. \tag{27}$$

For a given λ, the maximum spanning tree (MST) subproblem (26) can be efficiently solved by standard algorithms. The 0–1 knapsack subproblem (27) is known to be weakly NP-hard and we apply the COMBO dynamic programming algorithm [36] for efficiently solving it.

To obtain the tightest (smallest) upper bound, we have to solve the Lagrangian dual problem:

$$\min_{\lambda \in \mathbb{R}^E} v(\text{KCMST-LD}(\lambda)), \tag{28}$$

where $v(\text{KCMST-LD}(\lambda))$ denotes the optimal solution value to KCMST-LD(λ). This is achieved by applying the volume algorithm [4].

6.3 Lagrangian Heuristic

We employ several methods to also derive heuristic solutions and corresponding lower bounds. An obvious Lagrangian heuristic is the following: Whenever the spanning tree created in an iteration of the volume algorithm satisfies the capacity limit, we already have a feasible KCMST. In order to further improve such solutions we consecutively apply a local search based on an edge exchange neighborhood. Thereby we select an edge (u, v) not present in the solution and identify the least profitable edge—choosing an edge with highest weight in case of ties—of the path that connects nodes u and v in the current tree and that may be replaced by (u, v) without violating the capacity constraint. We then exchange these two edges in case the profit increases or it stays the same but the overall weight decreases. The edge to be included, (u, v), is either chosen (i) at random from $E \setminus T$, or (ii) at the beginning of the local search, all edges are sorted according to decreasing $p'_e = p_e - \lambda_e$ (the reduced profits used to solve the MST subproblem) and in every iteration of the local search the next less profitable edge not active in the current solution is chosen. The latter selection scheme results in a greedy search where every edge is considered at most once. Since Lagrangian multipliers are supposed to be of better quality in later phases of the optimization process, local search is only applied when the ratio of the incumbent lower and upper bounds is larger than a certain threshold τ. Local search stops after 100 consecutive non-improving iterations have been performed.

6.4 Evolutionary Algorithm for the KCMST

The EA for heuristically solving the KCMST is based on a direct edge-set representation as described in [45]. This encoding and its corresponding variation operators are known to provide strong locality and heritability, and all operations can efficiently be performed in time that depends (almost) only linearly on the number of nodes.

The general framework is steady-state, i.e. in each iteration one feasible offspring solution is created by means of recombination, mutation, and eventually local improvement, and it replaces the worst solution in the population. Duplicates are not allowed in the population; they are always immediately discarded. The EA's operators work as follows.

Initialization: A diversified initial population is obtained via a random spanning tree construction based on Kruskal's algorithm with a bias towards selecting edges with high profits. In case a generated solution is infeasible with respect to the knapsack constraint, it is stochastically repaired by iteratively selecting a not yet included edge at random, adding it to the tree, and removing an edge with highest weight from the induced cycle.

Recombination: An offspring is derived from two selected parental solutions in such a way that it always exclusively consists of inherited edges: In a first step all edges contained in both parents are immediately adopted. The remaining ones are merged into a single candidate list. From this list, we iteratively select edges

by binary tournaments with replacement favoring high-profit edges again. Selected edges are included in the solution if they do not introduce a cycle; otherwise, they are discarded. The process is repeated until a complete spanning tree is obtained. If it exceeds the capacity limit, the solution is repaired in the same way as during initialization, but only considering parental edges for inclusion.

Mutation: Mutation is performed by inserting a new randomly selected edge and removing another edge from the introduced cycle. The choice of the edge to be included is again biased towards high-profit edges by utilizing a normally-distributed rank-based selection, see [45]. The edge to be removed from the induced cycle is chosen at random among those edges whose removal retains a feasible solution.

Local Search: With a certain probability, a newly derived candidate solution is further improved by the previously described local search procedure.

6.5 LD/EA Hybrid

For the LD/EA hybrid we apply similar ideas as described in [28] for the prize collecting Steiner tree problem, where the EA is used successfully for finding better final solutions after performing LD. Here, the EA is adapted to exploit a variety of (intermediate) results from LD. Of course, the EA is only applied if the best feasible solution obtained by LD does not correspond to the determined upper bound; otherwise a proven optimal solution is already found. These steps are performed after LD has terminated and before the EA is executed:

1. For the selection of edges during initialization, recombination, and mutation, original edge profits p_e are replaced by reduced profits $p'_e = p_e - \lambda_e$. In this way, Lagrangian dual variables are exploited, and the heuristic search emphasizes the inclusion of edges that turned out to be beneficial in LD.
2. The edge set to be considered by the EA is reduced from E to a subset E' containing only those edges that appeared in any of the feasible solutions encountered by LD. For this purpose, LD is extended to mark those edges.
3. The best feasible solution obtained by LD is directly included in the EA's initial population.
4. Finally, the upper bound obtained by LD is exploited by the EA as an additional stopping criterion: When a solution with a corresponding total profit is found, it is optimal, and the EA terminates.

6.6 Experimental Results

The ability of the LD to yield extremely tight upper bounds that are significantly better than those resulting from the simple Lagrangian relaxation [54] is documented in [40]. Here we concentrate on the ability of the involved heuristics for improving the primal solutions. Therefore we show and compare results for the pure Lagrangian decomposition (LD), LD with local search (LD+LS), and the LD/EA hybrid (LD+LS+EA). Due to the absence of publicly available test instances we generated maximal planar graphs ($\mathrm{P}_{|V|,\gamma}$), and random ($\mathrm{R}_{|V|,|E|,\gamma,\delta}$) as well as complete graphs ($\mathrm{K}_{|V|,\gamma,\delta}$) as detailed in [29]. The instances differ in

1. size: number of nodes $|V|$ and edges $|E|$,

2. profit/weight correlation γ: being uncorrelated, weakly or strongly correlated for maximal planar graphs and of type outliers, weakly or strongly correlated for the other graph types,
3. and capacity limit δ: low, medium, or high limit.

A detailed treatment of these instances is given in [40]. For the optional local search, greedy edge selection is used for random and complete graphs with an application threshold set to $\tau = 0.99$ and random edge selection with $\tau = 0.995$ for the maximal planar graphs. The EA operates with a population size of 100 individuals, binary tournament selection is used. Local search is applied with a probability of 20% on each new candidate solution. The maximum number of EA iterations is 10000 for maximal planar graphs and 30000 for random and complete graphs. The edge set reduction was applied only in case of maximal planar graphs, as it turned out to be sometimes too restricting in the other cases.

All experiments were performed on a 2.2 GHz AMD Athlon 64 PC with 2 GB RAM. The results are given in Table 1; 10 runs per instance were performed for the stochastic algorithms. We state the CPU-time in seconds $t[s]$, the number of iterations *iter*, the average lower bounds LB, i.e. the objective values of the best feasible solutions. Upper bounds UB are expressed in terms of the relative gap to these lower bounds: $gap = (UB - LB)/LB$; corresponding standard deviations are listed in columns σ_{gap}. Columns *%-Opt* show the percentages of instances for which the gaps are zero and, thus, optimality has been achieved. For LD+LS+EA, the table additionally lists the average numbers of EA iterations $iter_{\text{EA}}$, the relative amounts of edges discarded after performing LD $red = (|E| - |E'|)/|E| \cdot 100\%$, stating (*red*) in case no reduction was applied, and the percentages of optimal solutions $\%\text{-}Opt_{\text{EA}}$, among *%-Opt*, found by the EA.

As can be seen, the solutions obtained by LD are already quite good and gaps are small in general. Applying the local search (LD+LS) always improves the average lower bound and in some cases helps to find more proven optimal solutions, which in turn reduces the number of iterations of the volume algorithm. The hybrid approach (LD+LS+EA) further boosts the average solution quality in almost all cases and substantially increases the number of solutions for which optimality could be proven, the increase in running time one has to pay is mostly only moderate. Of course, in order to solve the very few remaining instances to proven optimality as well, one could embed LD+LS+EA within a B&B.

7 Case Study: Metaheuristic Column Generation

In this section we discuss as a second case study a successful application of metaheuristics for solving the pricing subproblem within a column generation approach. The presented results are part of a currently ongoing project of the authors.

7.1 The Periodic Vehicle Routing Problem with Time Windows

Periodic vehicle routing problems (PVRPs) are generalized variants of the classical vehicle routing problem (VRP) where customers must be served several times within a given planning period. They occur in real-world applications as in courier services, grocery distribution or waste collection. The PVRP considered here is the

Table 1. Results of Lagrangian decomposition and hybrid algorithms on maximal planar, random, and complete graphs.

Instance	**LD**						**LD+LS**						**LD+LS+EA**							
	$t[s]$	$iter$	LB	gap $[\cdot 10^{-5}]$	σ_{gap} $[\cdot 10^{-5}]$	$\%$-Opt	$t[s]$	$iter$	LB	gap $[\cdot 10^{-5}]$	σ_{gap} $[\cdot 10^{-5}]$	$\%$-Opt	$t[s]$	red	$iter_{\mathrm{EA}}$	LB	gap $[\cdot 10^{-5}]$	σ_{gap} $[\cdot 10^{-5}]$	$\%$-Opt	$\%$-Opt_{EA}
$P_{2000,u}$	1.48	791	147799.50	0.0683	0.2049	90	2.28	782	147799.55	0.0342	0.1489	95	2.90	41.21	150	147799.60	0	0	100	5
$P_{2000,w}$	1.52	853	85570.50	0.3519	0.7513	80	2.38	844	85570.63	0.1994	0.5261	86	4.26	42.61	457	85570.78	0.0235	0.1643	98	12
$P_{2000,s}$	2.12	1030	82521.70	1.9389	2.3118	40	2.66	868	82523.30	0	0	100	2.66	21.99	0	82523.30	0	0	100	0
$P_{4000,u}$	3.35	859	294872.00	0.0340	0.1019	90	5.59	841	294872.03	0.0238	0.0866	93	8.64	40.17	316	294872.10	0	0	100	7
$P_{4000,w}$	4.19	1053	170956.70	0.8195	0.9155	40	6.15	978	170957.79	0.1813	0.306	72	14.66	43.82	842	170958.06	0.0234	0.1147	96	24
$P_{4000,s}$	4.71	1066	165049.80	1.0300	0.8590	30	5.99	915	165051.44	0.0364	0.1439	94	9.95	19.92	410	165051.48	0.0121	0.0848	98	4
$P_{6000,u}$	5.66	912	441977.80	0.0680	0.1038	70	9.33	886	441977.96	0.0317	0.0786	86	15.41	40.25	339	441978.10	0	0	100	14
$P_{6000,w}$	6.55	1022	256317.40	0.3904	0.4621	50	9.25	964	256318.09	0.1210	0.2452	76	24.47	45.14	909	256318.36	0.0156	0.0764	96	20
$P_{6000,s}$	8.14	1157	247587.90	1.7368	1.3032	20	10.44	996	247592.04	0.0646	0.1481	84	33.73	19.94	1401	247592.09	0.0444	0.1264	89	5
$P_{8000,u}$	8.32	960	589446.50	0.1017	0.1357	60	13.81	918	589446.89	0.0356	0.077	81	28.44	39.98	595	589447.09	0.0017	0.0168	99	18
$P_{8000,w}$	9.78	1107	341902.50	0.5555	0.5139	30	14.18	1037	341903.85	0.1609	0.2124	58	48.40	44.82	1384	341904.37	0.0088	0.0499	97	39
$P_{8000,s}$	10.88	1125	330117.10	1.5147	1.3065	20	14.20	990	330121.86	0.0727	0.1294	76	57.00	17.99	1727	330121.96	0.0424	0.1051	86	10
$R_{300,11213,o,l}$	9.53	1737	542839.40	1.7477	1.8326	10	11.72	1737	542840.60	1.5271	1.5937	10	29.99	(92.93)	27000	542843.63	0.9706	0.6928	10	0
$R_{300,11213,o,m}$	7.10	1536	580716.50	0.2583	0.2464	30	8.89	1506	580716.60	0.2411	0.2576	40	21.43	(91.63)	18000	580716.64	0.2342	0.2477	40	0
$R_{300,11213,o,h}$	3.57	1260	591409.00	0.1690	0.2507	50	5.11	1259	591409.30	0.1183	0.1320	50	13.73	(91.02)	12285	591409.54	0.0778	0.1132	64	14
$R_{300,11213,s_2,l}$	24.58	1563	77466.60	8.5209	5.6046	20	24.45	1409	77473.00	0.2581	0.5161	80	24.69	(80.64)	336	77473.20	0	0	100	20
$R_{300,11213,s_2,m}$	15.37	1351	155244.80	5.4064	5.1165	0	14.77	1051	155253.20	0	0	100	14.73	(81.54)	0	155253.20	0	0	100	0
$R_{300,11213,s_2,h}$	16.52	1332	232877.70	6.5305	5.2668	10	16.74	1238	232892.50	0.1718	0.2847	70	18.34	(85.28)	2222	232892.89	0.0043	0.0428	99	29
$R_{300,22425,o,l}$	26.39	3324	568771.90	6.8383	6.1475	10	32.10	3324	568788.80	3.8714	4.3327	10	52.08	(95.24)	26700	568796.00	2.6042	3.3654	11	1
$R_{300,22425,o,m}$	14.70	1943	588410.30	0.2210	0.2020	30	18.83	1943	588410.50	0.1870	0.1605	30	33.05	(95.46)	18078	588410.80	0.1360	0.1272	40	10
$R_{300,22425,o,h}$	7.28	1358	594373.50	0.0168	0.0505	90	10.10	1358	594373.50	0.0168	0.0505	90	12.40	(94.54)	3000	594373.50	0.0168	0.0505	90	0
$R_{300,22425,s_2,l}$	44.08	2059	77445.70	12.2628	9.0170	0	42.58	1793	77455.20	0	0	100	42.58	(86.26)	0	77455.20	0	0	100	0
$R_{300,22425,s_2,m}$	29.69	1687	154940.30	7.8185	8.9007	10	28.81	1392	154952.40	0	0	100	28.81	(93.71)	0	154952.40	0	0	100	0
$R_{300,22425,s_2,h}$	34.63	1964	232424.80	16.2741	12.5659	10	36.55	1885	232461.90	0.3013	0.3874	50	44.59	(89.39)	10682	232462.37	0.0990	0.1811	77	27
$K_{300,o,l}$	247.29	19163	582646.00	4.0334	7.1749	10	316.33	19163	582660.30	1.5789	1.4435	10	333.98	(97.50)	27000	582663.46	1.0366	0.8511	10	0
$K_{300,o,m}$	40.44	2909	592797.70	0.1856	0.1401	30	45.96	2864	592797.90	0.1518	0.1401	40	55.19	(97.70)	10212	592798.50	0.0506	0.0773	70	30
$K_{300,o,h}$	30.13	2373	596076.40	0.0503	0.1074	80	35.49	2371	596076.50	0.0336	0.0671	80	36.13	(96.94)	1239	596076.70	0	0	100	20
$K_{300,s_2,l}$	63.20	2495	77225.70	28.6269	20.8442	0	60.80	2195	77247.80	0	0	100	60.80	(93.07)	0	77247.80	0	0	100	0
$K_{300,s_2,m}$	62.25	2704	154445.00	12.4958	8.3394	0	59.11	2404	154464.30	0	0	100	59.11	(94.48)	0	154464.30	0	0	100	0
$K_{300,s_2,h}$	76.60	3396	231665.00	15.9285	18.7408	10	78.10	3142	231701.90	0	0	100	78.10	(92.77)	0	231701.90	0	0	100	0

Periodic Vehicle Routing Problem with Time Windows (PVRPTW). It is defined on a complete directed graph $G = (V, A)$, where $V = \{v_0, v_1, \dots v_n\}$ is the vertex set and $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is the arc set. The planning horizon shall be t days, also referred to as $T = \{1, \dots, t\}$. Vertex v_0 represents the depot with time window $[e_0, l_0]$ at which we have a fleet of m homogeneous vehicles with capacity Q and maximal daily working time D. Each vertex $i \in V_C$, with $V_C = V \setminus \{v_0\}$, corresponds to a customer and has an associated demand $q_i \geq 0$, a service duration $d_i \geq 0$, a time window $[e_i, l_i]$, a service frequency f_i and a set C_i of allowable combinations of visit days. For each arc $(v_i, v_j) \in A$ there are given travel times (costs) $c_{ij} \geq 0$. The aim is (i) to select a single visit combination per customer and (ii) to find at most m vehicle routes on each of the t days on G, such that

(1) each route starts and ends at the depot,
(2) each customer i belongs to f_i routes over the planning horizon,
(3) the total demand of the route for each vehicle does not exceed the capacity limit Q,
and its duration does not exceed the maximal working time D,
(4) the service at each customer i begins in the interval $[e_i, l_i]$ and every vehicle leaves the depot and returns to it in the interval $[e_0, l_0]$, and
(5) the total travel costs of all vehicles are minimized.

We further assume so-called hard time windows, i.e. arriving before e_i at customer i incurs a waiting time at no additional costs, whereas arriving later than l_i is not allowed. The PVRPTW has been first mentioned in Cordeau et al. [8], where a tabu search metaheuristic is described for it.

7.2 Set Covering Formulation for the PVRPTW

Among the most successful solution approaches for VRPs in general are algorithms based on column generation. Therefore we focus on an IP formulation suitable for such an approach and formulate the integer master problem (IMP) for the PVRPTW as a set-covering model:

$$\min \sum_{\tau \in T} \sum_{\omega \in \Omega} \gamma_\omega \, \upsilon_{\omega\tau} \tag{29}$$

$$\text{s.t.} \quad \sum_{r \in C_i} y_{ir} \geq 1, \qquad \forall i \in V_C, \tag{30}$$

$$\sum_{\omega \in \Omega} \upsilon_{\omega\tau} \leq m, \qquad \forall \tau \in T, \tag{31}$$

$$\sum_{\omega \in \Omega} \alpha_{i\omega} \, \upsilon_{\omega\tau} - \sum_{r \in C_i} \beta_{ir\tau} \, y_{ir} \geq 0, \qquad \forall i \in V_C, \forall \tau \in T, \tag{32}$$

$$y_{ir} \in \{0, 1\}, \qquad \forall i \in V_C, \forall r \in C_i, \tag{33}$$

$$\upsilon_{\omega\tau} \in \{0, 1\}, \qquad \forall \omega \in \Omega, \forall \tau \in T. \tag{34}$$

The set of all feasible individual routes is denoted by Ω, and with each route $\omega \in \Omega$ we have associated costs γ_ω and variables $\upsilon_{\omega\tau}$, $\forall \tau \in T$, representing the number of times route ω is selected on day τ. For each customer $i \in V_C$, variable y_{ir} indicates whether or not visit combination $r \in C_i$ is chosen. The objective is to minimize

the total costs of all routes (29). Covering constraints (30) guarantee that at least one visit day combination is selected per customer, fleet constraints (31) restrict the number of daily routes to not exceed the number of available vehicles m, and visit constraints (32) link the routes and the visit combinations, whereas $\alpha_{i\omega}$ and $\beta_{ir\tau}$ are binary constants indicating if route ω visits customer i and if day τ belongs to visit combination $r \in C_i$ of customer i, respectively.

7.3 Column Generation for Solving the LP Relaxation

Here, our aim is to derive a lower bound for the IMP by exactly solving its LP relaxation. An extension of the approach towards an exact branch-and-price algorithm is part of our ongoing work. Conditions (33) and (34) are replaced by $y_{ir} \geq 0$ and $v_{\omega\tau} \geq 0$, yielding the (linear) master problem (MP). Due to the large number of variables (columns) corresponding to routes, this LP cannot be solved directly. Instead, we restrict ourselves to a small number of initial columns $\Omega' \subset \Omega$, yielding the corresponding restricted master problem (RMP). Additional columns (routes) that are able to improve the current LP solution are generated by iteratively solving the pricing subproblem, which resembles in our case a *shortest path problem with resource constraints* (SPPRC) [31] and is NP-hard. Regarding the quality of the theoretically obtainable lower bound it is beneficial to restrict the search to elementary paths, hence only considering the *elementary SPPRC* (ESPPRC). The following ESPPRC pricing subproblem holds for each day $\tau \in T$ and is solved on an auxiliary graph $G' = (V', A')$, with $V' = V \cup \{v_{n+1}\}$ and $A' = \{(v_0, i), (i, v_{n+1}) : i \in V_C\} \cup \{(i, j) : i, j \in V_C,\, i \neq j\}$, where v_{n+1} is a copy of the (starting) depot v_0 and acts as target node:

$$\min \sum_{i \in V'} \sum_{j \in V'} \hat{c}_{ij\tau}\, x_{ij} \tag{35}$$

$$\text{s.t.} \quad \sum_{j \in V_C} x_{0j} = 1 \tag{36}$$

$$\sum_{i \in V'} x_{ik} - \sum_{j \in V'} x_{kj} = 0 \qquad \forall k \in V_C \tag{37}$$

$$\sum_{i \in V_C} x_{i,n+1} = 1 \tag{38}$$

$$\sum_{i \in V_C} \sum_{j \in V'} q_i\, x_{ij} \leq Q \tag{39}$$

$$a_{n+1} - w_0 \leq D \tag{40}$$

$$a_i + w_i + d_i + c_{ij} - M_{ij}(1 - x_{ij}) \leq a_j \qquad \forall (i,j) \in A' \tag{41}$$

$$e_i \leq (a_i + w_i) \leq l_i \qquad \forall i \in V' \tag{42}$$

$$w_i \geq 0 \qquad \forall i \in V' \tag{43}$$

$$a_i \geq 0 \qquad \forall i \in V' \setminus \{v_0\} \tag{44}$$

$$a_0 = 0 \tag{45}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i,j) \in A' \tag{46}$$

Variables x_{ij}, $\forall (i,j) \in A'$, denote which arcs from A' are used, and $\hat{c}_{ij\tau}$ are the reduced costs of using arc (i,j) on day τ:

$$\hat{c}_{ij\tau} = \begin{cases} c_{ij} - \rho_\tau & \text{if } i = v_0,\ j \in V_C, \\ c_{ij} - \pi_{i\tau} & \text{if } i \in V_C,\ j \in V'. \end{cases} \tag{47}$$

with ρ_τ and $\pi_{i\tau}$ being the dual variable values of constraints (31) and (32), respectively. Equalities (36) to (38) are flow conservation constraints, and inequalities (39) and (40) guarantee to not exceed the capacity and duration limits, respectively. Finally, (41) and (42) are time constraints, with variable a_i denoting the arrival time at customer i and w_i the waiting time occurring after a_i.

7.4 Exact and Metaheuristic Pricing Procedures

We apply an exact algorithm as well as metaheuristics for solving the ESPPRC subproblem. The former is realized by a dynamic programming approach based on [7, 14]. We use a label correcting algorithm and expand the partial paths from the depot v_0 to the target node v_{n+1}, thereby retaining only non-dominated labels taking into account the cumulated costs, load, duration, and overall waiting time, as well as the arrival time and the set of unreachable nodes. To minimize route duration we adhere to the concept of *forward time slack* [48] and maximize the waiting time w_0 at the depot without introducing a time window violation. This is also considered when extending labels and checking the dominance relation. The algorithm can also be stopped after a certain number of negative cost paths have been found, i.e. applying a "forced early stop", cf. [32].

The first metaheuristic is an instance of iterated local search (ILS) [34]. It starts with the "empty" path (v_0, v_{n+1}) with zero costs and applies in each iteration a perturbation and a subsequent local improvement phase. Both phases make use of the following neighborhood structures: inserting, deleting, moving, replacing, and exchanging individual customers. The local search selects them in a random fashion and always accepts the first improving change. Perturbation applies ten random neighborhood moves in a sequence.

Our second, alternative metaheuristic approach can be regarded a greedy randomized adaptive search procedure (GRASP) [15]: In each iteration we start with the "empty" path (v_0, v_{n+1}) with zero costs and successively try to add arcs having negative costs, always selecting one at random in case there are more available; afterwards we also apply the perturbation and local search phase as described for the ILS algorithm.

Whenever an iteration of the metaheuristics results in a negative cost path it is stored and returned at the end of the procedure. Once one or more negative cost routes have been determined for one of the daily subproblems, corresponding variables are priced in for all days and the RMP is resolved. In the following iteration we start the column generation with the same daily subproblem before considering the others. The whole process is continued until a full iteration over all days yields no new negative cost routes.

7.5 Experimental Results

Benchmark instances were taken from [8]. They are divided in types 'a' and 'b' having narrow and wide time windows, respectively. We reduced some of them by selecting only a random subset of the customers and decreasing the number of vehicles in an

appropriate way; in this case we give a subscript denoting the index of the reduced instance. The initial set of columns is provided by taking the routes of feasible solutions of a variable neighborhood search described in [39]. All algorithms have been implemented in C++ using GCC 4.1 and were executed on a single core of a 2.2 GHz AMD Opteron 2214 PC with 4 GB RAM. CPLEX in version 11.1 was used as LP solver. The ESPPRC subproblem is solved in four alternative ways: (i) by dynamic programming (DP), (ii) by dynamic programming with a forced early stop after 1000 generated columns (DP^S), (iii) by ILS and a subsequent application of DP (ILS+DP), and finally (iv) by GRASP and running DP afterwards (GRASP+DP). The metaheuristics' iteration limit is originally set to 1000 and extended to 10000 if no new columns have been generated so far (on a per-run basis). DP is applied after the metaheuristic if less than 100 new columns have been generated. In all experiments, column generation was performed until the LP relaxation of the set covering formulation has been solved to optimality, i.e. it has been proven by DP that no further columns with negative reduced costs exist.

Table 2 shows the instances, the upper bounds (UB) initially provided by the variable neighborhood search, the (exact) lower bounds (LB) obtained from column generation, the percentage gaps between them, i.e. %-gap $= (\mathrm{UB} - \mathrm{LB})/\mathrm{LB} \cdot 100\%$, the CPU-times of settings DP and DP^S, as well as the minimal and average times of settings ILS+DP and GRASP+DP over 10 runs per instance. It can be observed that DP^S is faster than DP for instances with narrow time windows, whereas it is almost the opposite for instances with wide time windows. However, using one of the metaheuristic combinations ILS+DP or GRASP+DP is almost always fastest, especially for larger instances, when the speed of the heuristic column generation outweighs the probably higher quality columns of the DP algorithm. Among the two metaheuristic combinations, no obvious advantage is observable for either of them.

8 Conclusions

The present chapter reviewed important MetaBoosting literature and presented two exemplary case studies where an in-depth description of successful hybrid algorithms was given. Many different hybridization approaches exist, most of them are specialized methods for specific problems, but another significant part of the surveyed research considers generic problems such as mixed integer programming. It is often possible to accelerate exact methods by introducing (meta-)heuristic knowledge and, if fixed time-limits are given, the overall solution quality might also benefit from such ideas.

In most exact approaches tight bounds are crucial aspects of success. Thus, different ways of applying metaheuristics for finding primal bounds were examined. The obvious way of determining high quality initial solutions can be beneficial to the overall optimization process, as it has been described for the multidimensional knapsack problem. General methods for determining improved primal solutions throughout the search process for generic mixed integer programming, such as local branching and relaxation or distance induced neighborhood search, have been found so effective that some of them have been included into the commercial MIP solver CPLEX. Solution merging approaches based on evolutionary algorithms were also successfully included in this solver. Other more problem specific methods often yielding opti-

Table 2. Experimental results of column generation with different pricing strategies for the PVRPTW: CPU-times for exactly solving the LP relaxation of the set covering formulation.

Instance							DP	DP^S	ILS+DP		GRASP+DP	
No.	n	m	t	UB	LB	%-gap	t[s]	t[s]	min t[s]	avg. t[s]	min t[s]	avg. t[s]
1a	48	3	4	2909.02	2882.01	0.94	5.01	4.87	11.44	13.92	13.40	18.59
2a	96	6	4	5032.06	4993.48	0.77	72.27	53.91	48.35	55.83	45.31	57.09
3a	144	9	4	7138.65	6841.44	4.34	374.07	291.61	231.17	265.28	223.03	262.89
$4a_{r1}$	160	10	4	6929.84	6641.67	4.34	1240.96	1136.64	605.06	744.69	640.15	754.71
7a	72	5	6	6784.71	6641.39	2.16	24.78	17.10	16.98	20.77	19.58	23.82
$9a_{r1}$	96	7	6	8545.80	8035.09	6.36	146.88	134.55	88.89	104.17	96.60	103.75
$9a_{r2}$	120	8	6	8598.40	8140.15	5.63	898.90	693.44	446.49	545.09	475.98	521.66
8a	144	10	6	9721.25	9153.79	6.20	745.95	592.07	367.82	418.87	383.18	421.63
$2b_{r1}$	32	2	4	2709.15	2682.52	1.00	89.65	121.30	43.58	64.79	25.36	55.78
1b	48	3	4	2277.44	2258.85	0.82	156.77	158.66	76.66	109.17	99.59	123.35
$2b_{r2}$	64	4	4	2771.68	2733.55	1.40	277.76	254.38	131.72	192.18	168.75	188.61
$3b_{r1}$	72	4	4	3306.86	3241.90	2.00	726.37	749.11	426.10	533.68	417.05	488.95
$7b_{r1}$	24	2	6	3776.25	3677.21	2.70	0.54	0.55	1.92	2.29	1.72	2.25
$8b_{r1}$	36	2	6	3640.79	3476.43	4.73	10.01	10.15	8.50	13.26	10.24	12.55
$7b_{r2}$	48	3	6	3723.18	3599.72	3.43	48.04	31.78	30.95	43.13	34.03	43.20
$8b_{r2}$	60	3	6	4606.17	4324.87	6.50	1538.18	1196.51	826.08	1121.28	804.35	967.87

mal or close to optimal primal solutions based on the hybridization of Lagrangian relaxation and metaheuristics were also examined.

A multitude of collaborative approaches exist and some intertwined and parallel combinations were described in this chapter. Parallel combinations gain importance because of the current hardware developments and the broad availability of multi-core processors.

Mathematical programming techniques based on problem decomposition, such as cut and column generation approaches, play an essential role in the advances of exact methods. Applications where metaheuristic algorithms are used for such tasks were described. Especially sequential combinations of fast and simple construction heuristics and more sophisticated metaheuristic approaches are very promising in both cut and column generation.

Different aspects and difficulties in the development of hybrid methods were discussed in more detail in the two case studies. The first one describes a Lagrangian decomposition approach combined with an evolutionary algorithm for solving the knapsack constrained maximum spanning tree problem. The Lagrangian approach combined with an implicit construction heuristic and a subsequent local search is already a powerful procedure yielding tight gaps, but its combination with the EA allows to optimally solve substantially more of the large-scale instances. The second case study presents a column generation approach for solving the periodic vehicle routing problem with time windows. A greedy randomized adaptive search procedure, an iterated local search, and a dynamic programming algorithm are applied for solving the pricing subproblem. The inclusion of metaheuristic techniques led to a significant acceleration of the column generation process compared to using the dynamic programming subproblem solver alone.

Hybridizing exact algorithms and metaheuristics, and MetaBoosting in particular are promising research areas. Further exciting results can be expected since various possible synergies are still unexplored. Especially generating, exchanging, and translating information about the ongoing optimization process by exploiting advanced features of the different algorithms will possibly lead to further progress in the field.

Acknowledgements

This work is supported by the Austrian Science Fund (FWF) under contract number P20342-N13.

References

1. P. Augerat, J. Belenguer, E. Benavent, A. Corberan, and D. Naddef. Separating capacity constraints in the CVRP using tabu search. *European Journal of Operational Research*, 106(2):546–557, 1999.
2. E. Balas. An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, 13(4):517–549, 1965.
3. E. Balas and C. H. Martin. Pivot and complement – a heuristic for 0–1 programming. *Management Science*, 26(1):86–96, 1980.

4. F. Barahona and R. Anbil. The volume algorithm: Producing primal solutions with a subgradient method. *Mathematical Programming, Series A*, 87(3):385–399, 2000.
5. L. Bertaccoa, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4:63–76, 2007.
6. D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.
7. A. Chabrier. Vehicle routing problem with elementary shortest path based column generation. *Computers & Operations Research*, 33(10):2972–2990, 2006.
8. J.-F. Cordeau, G. Laporte, and A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52:928–936, 2001.
9. E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming, Series A*, 102:71–90, 2005.
10. G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a large scale traveling salesman problem. *Operations Research*, 2:393–410, 1954.
11. G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.
12. J. Denzinger and T. Offermann. On cooperation between evolutionary algorithms and other search paradigms. In W. Porto et al., editors, *Proceedings of the 1999 Congress on Evolutionary Computation*, volume 3, pages 2317–2324. IEEE Press, 1999.
13. O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194(1-3):229–237, 1999.
14. D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004.
15. T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.
16. G. R. Filho and L. A. N. Lorena. Constructive genetic algorithm and column generation: an application to graph coloring. In L. P. Chuen, editor, *Proceedings of the Fifth Conference of the Association of Asian-Pacific Operations Research Societies within IFORS*, 2000.
17. M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.
18. M. Fischetti and A. Lodi. Local Branching. *Mathematical Programming, Series B*, 98:23–47, 2003.
19. M. Fischetti, C. Polo, and M. Scantamburlo. Local branching heuristic for mixed-integer programs with 2-level variables, with an application to a telecommunication network design problem. *Networks*, 44(2):61–72, 2004.
20. M. L. Fisher. The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Management Science*, 27(1):1–18, 1981.
21. A. Frangioni. About Lagrangian methods in integer optimization. *Annals of Operations Research*, 139(1):163–193, 2005.
22. A. P. French, A. C. Robinson, and J. M. Wilson. Using a hybrid genetic algorithm/branch and bound approach to solve feasibility and optimization integer programming problems. *Journal of Heuristics*, 7:551–564, 2001.

23. S. Ghosh. DINS, a MIP improvement heuristic. In M. Fischetti and D. P. Williamson, editors, *Integer Programming and Combinatorial Optimization: 12th International IPCO Conference, Proceedings*, volume 4513 of *LNCS*, pages 310–323. Springer, 2007.
24. P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849–859, 1961.
25. F. Glover. Surrogate constraints. *Operations Research*, 16(4):741–749, 1968.
26. F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39(3):653–684, 2000.
27. P. Hansen, N. Mladenović, and D. Urosević. Variable neighborhood search and local branching. *Computers and Operations Research*, 33(10):3034–3045, 2006.
28. M. Haouari and J. C. Siala. A hybrid Lagrangian genetic algorithm for the prize collecting Steiner tree problem. *Computers & Operations Research*, 33(5):1274–1288, 2006.
29. S. T. Henn. Weight-constrained minimal spanning tree problem. Master's thesis, University of Kaiserslautern, Department of Mathematics, May 2007.
30. F. S. Hillier. Efficient heuristic procedures for integer linear programming with an interior. *Operations Research*, 17(4):600–637, 1969.
31. S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. In G. Desaulniers et al., editors, *Column Generation*, chapter 2, pages 33–65. Springer, 2005.
32. J. Larsen. *Parallelization of the Vehicle Routing Problem with Time Windows*. PhD thesis, Technical University of Denmark, 1999.
33. M. Leitner and G. R. Raidl. Lagrangian decomposition, metaheuristics, and hybrid approaches for the design of the last mile in fiber optic networks. In M. J. Blesa et al., editors, *Hybrid Metaheuristics 2008*, volume 5296 of *LNCS*, pages 158–174. Springer, 2008.
34. H. R. Lourenco, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, 2003.
35. M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.
36. S. Martello, D. Pisinger, and P. Toth. Dynamic programming and strong bounds for the 0–1 knapsack problem. *Management Science*, 45:414–424, 1999.
37. G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.
38. C. Oliva, P. Michelon, and C. Artigues. Constraint and linear programming: Using reduced costs for solving the zero/one multiple knapsack problem. In *International Conference on Constraint Programming, Proceedings of the workshop on Cooperative Solvers in Constraint Programming*, pages 87–98, Paphos, Greece, 2001.
39. S. Pirkwieser and G. R. Raidl. A variable neighborhood search for the periodic vehicle routing problem with time windows. In C. Prodhon et al., editors, *Proceedings of the 9th EU/MEeting on Metaheuristics for Logistics and Vehicle Routing*, Troyes, France, 2008.
40. S. Pirkwieser, G. R. Raidl, and J. Puchinger. A Lagrangian decomposition/evolutionary algorithm hybrid for the knapsack constrained maximum spanning tree problem. In C. Cotta and J. van Hemert, editors, *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, volume 153 of *Studies in Computational Intelligence*, pages 69–85. Springer, 2008.

41. J. Puchinger and G. R. Raidl. An evolutionary algorithm for column generation in integer programming: an effective approach for 2D bin packing. In X. Yao et al., editors, *Parallel Problem Solving from Nature – PPSN VIII*, volume 3242 of *LNCS*, pages 642–651. Springer, 2004.
42. J. Puchinger and G. R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation, Part II*, volume 3562 of *LNCS*, pages 41–53. Springer, 2005.
43. J. Puchinger and G. R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, 183:1304–1327, 2007.
44. J. Puchinger, G. R. Raidl, and M. Gruber. Cooperating memetic and branch-and-cut algorithms for solving the multidimensional knapsack problem. In *Proceedings of the 6th Metaheuristics International Conference*, pages 775–780, Vienna, Austria, 2005.
45. G. R. Raidl and B. A. Julstrom. Edge sets: an effective evolutionary coding of spanning trees. *IEEE Transactions on Evolutionary Computation*, 7(3):225–239, 2003.
46. W. Rei, J.-F. Cordeau, M. Gendreau, and P. Soriano. Accelerating Benders decomposition by local branching. *INFORMS Journal on Computing*, to appear.
47. E. Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.
48. M. W. P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4:146–154, 1992.
49. S. Talukdar, L. Baeretzen, A. Gove, and P. de Souza. Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, 4:295–321, 1998.
50. M. Vasquez and J.-K. Hao. A hybrid approach for the 0–1 multidimensional knapsack problem. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 328–333. Morgan Kaufman, 2001.
51. M. Vasquez and Y. Vimont. Improved results on the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, 165(1):70–81, 2005.
52. Y. Vimont, S. Boussier, and M. Vasquez. Reduced costs propagation in an efficient implicit enumeration for the 0–1 multidimensional knapsack problem. *Journal of Combinatorial Optimization*, 15(2):165–178, 2008.
53. L. A. Wolsey. *Integer Programming*. Wiley-Interscience, 1998.
54. T. Yamada, K. Watanabe, and S. Katakoa. Algorithms to solve the knapsack constrained maximum spanning tree problem. *International Journal of Computer Mathematics*, 82(1):23–34, 2005.