Hybrid metaheuristics in combinatorial optimization: A survey

Christian Blum, Jakob Puchinger, Günther R. Raidl, Andrea Roli

Hybrid Metaheuristics in Combinatorial Optimization: A Survey☆

Christian Blum*,[a], Jakob Puchinger[b], Günther R. Raidl[c], Andrea Roli[d]

[a] *ALBCOM Research Group, Universitat Politècnica de Catalunya, Barcelona, Spain*
[b] *Mobility Department, Austrian Institute of Technology, Vienna, Austria*
[c] *Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria*
[d] *Dipartimento di Elettronica, Informatica e Sistemistica (DEIS),* Alma Mater Studiorum *Università di Bologna, Campus of Cesena, Italy*

Abstract

Research in metaheuristics for combinatorial optimization problems has lately experienced a noteworthy shift towards the hybridization of metaheuristics with other techniques for optimization. At the same time, the focus of research has changed from being rather algorithm-oriented to being more problem-oriented. Nowadays the focus is on solving the problem at hand in the best way possible, rather than promoting a certain metaheuristic. This has led to an enormously fruitful cross-fertilization of different areas of optimization. This cross-fertilization is documented by a multitude of powerful hybrid algorithms that were obtained by combining components from several different optimization techniques. Hereby, hybridization is not restricted to the combination of different metaheuristics but includes, for example, the combination of exact algorithms and metaheuristics. In this work we provide a survey of some of the most important lines of hybridization. The literature review is accompanied by the presentation of illustrative examples.

Key words:
hybrid metaheuristics, optimization

1. Introduction

The origins of *metaheuristics* are to be found in the Artificial Intelligence and Operations Research communities [1, 2, 3]. The term metaheuristic gener-

☆This work was supported by grant TIN2007-66523 (FORMALISM) of the Spanish government, and by the Austrian Science Fund (FWF) under contract number P20342-N13. Moreover, Christian Blum acknowledges support from the *Ramón y Cajal* program of the Spanish Ministry of Science and Innovation.

*Corresponding author
Email addresses: cblum@lsi.upc.edu (Christian Blum), jakob.puchinger@ait.ac.at (Jakob Puchinger), raidl@ads.tuwien.ac.at (Günther R. Raidl), andrea.roli@unibo.it (Andrea Roli)

ally refers to approximate algorithms for optimization that are not specifically expressed for a particular problem. Ant colony optimization, genetic and evolutionary algorithms, iterated local search, simulated annealing and tabu search (in alphabetical order) are typical representatives of the class of metaheuristic algorithms. Each of these metaheuristics has its own historical background. Some metaheuristics are inspired by natural processes such as evolution, others are extensions of less sophisticated algorithms such as greedy heuristics and local search [4].

During the first two decades of research on metaheuristics, different research communities working on metaheuristic techniques co-existed without much interaction, neither among themselves nor with the Operations Research community. This was surely justified by the fact that initially pure metaheuristics had a considerable success: for many problems they quickly became state-of-the-art algorithms. However, the attempt of being different to traditional Operations Research has led to a pernicious disregard of valuable optimization expertise collected over the years. Only when it became clear that pure metaheuristics had reached their limits, researchers turned towards the combination of different algorithms.

Over the last years, quite an impressive number of algorithms were reported that do not purely follow the paradigm of a single traditional metaheuristic. On the contrary, they combine various algorithmic components, often originating from algorithms of other research areas on optimization. These approaches are commonly referred to as *hybrid metaheuristics*. The lack of a precise definition of this term has sometimes been subject to criticism. Note, however, that the relatively open nature of this expression can be helpful, as strict borderlines between related fields of research are often a hindrance for creative thinking and the exploration of new research directions.

The main motivation behind the hybridization of different algorithms is to exploit the complementary character of different optimization strategies, that is, hybrids are believed to benefit from *synergy*. In fact, choosing an adequate combination of complementary algorithmic concepts can be the key for achieving top performance in solving many hard optimization problems. Unfortunately, developing an effective hybrid approach is in general a difficult task which requires expertise from different areas of optimization. Moreover, the literature shows that it is nontrivial to generalize, that is, a certain hybrid might work well for specific problems, but it might perform poorly for others. Nevertheless, there are hybridization types that have shown to be successful for many applications. They may serve as a guidance for new developments.

Conferences and workshops such as *CPAIOR* [5, 6, 7], *Hybrid Metaheuristics* [8, 9], and *Matheuristics* [10, 11, 12, 13] document the growing popularity of hybridization. Moreover, the first book specifically devoted to hybrid metaheuristics has been published in 2008 [14]. In this paper, we provide an overview of hybrid metaheuristics for combinatorial optimization problems by illustrating prominent and paradigmatic examples, which range from the integration of metaheuristic techniques among themselves, to the hybridization of metaheuristics with constraint and mathematical programming. The interested reader can

find other reviews on hybrid metaheuristics in [15, 16, 17, 14, 18, 13]. Note that this article is an extension and actualization of the work that has appeared in [19, 20].

Finally, we would like to emphasize that this survey covers the area of hybrid metaheuristics for single-objective combinatorial optimization problems. Readers interested in recent developments concerning hybrid metaheuristics for multi-objective optimization are referred to a survey specifically devoted to this topic [21]. Concerning the very active field of (hybrid) metaheuristics for continuous—i.e., real parameter—optimization, readers may find a good starting point in recent papers published in a dedicated journal special issue [22]. Especially in the fields of evolutionary algorithms and swarm intelligence, the use of pure as well as hybrid metaheuristics for continuous optimization has already a quite long tradition [23, 24, 25]. For an overview on parallel hybrid metaheuristics we recommend [26, 27]. Finally, it is also important to mention the availability of several currently available software frameworks that support the implementation of hybrid metaheuristics for single and multi-objective optimization, both in a sequential and in a parallel way. One of the most powerful ones is ParadisEO [28].

Organization. The following five sections are devoted to the presentation of examples and short literature overviews concerning five important categories of hybrid metaheuristics. More specifically we focus on the hybridization of metaheuristics with (meta-)heuristics, constraint programming, tree search methods, problem relaxation, and dynamic programming. For each topic, first, two representative examples are outlined, and then, a short literature overview is provided.

2. Hybridizing Metaheuristics with (Meta-)Heuristics

When researchers first considered the hybridization of their preferred metaheuristic with another technique for optimization, most started to look at possible combinations with heuristics or other metaheuristics. In fact, nowadays the hybridization of different metaheuristics is very widespread, especially for what concerns the use of local search methods within population-based methods. In fact, both evolutionary computation and ant colony optimization often make use of local search procedures for refining the solutions that are generated during the search process. This can be attributed to the fact that these nature-inspired methods are good concerning the exploration of the search space and the identification of areas with high quality solutions. At initialization they generally try to capture a global picture of the search space. Then, during the search process they successively focus the search on more promising regions of the search space. However, they are usually not so effective concerning the exploitation of the accumulated search experience, that is, finding the best solutions in these high quality areas. On the other side, the strength of local search is the capability of quickly finding better solutions in the vicinity of given starting solutions. In summary, population-based methods are good in identifying promising areas of

the search space in which local search methods can then quickly determine the best solutions. This is why this type of hybridization is usually very successful. Evolutionary algorithms making use of local search methods are sometimes labelled as *memetic algorithms* [29, 30].

In contrast to the standard way of hybridization that was mentioned above, the two examples that are presented in more detail in the following represent rather unconventional ways of hybridization. First, *population-based iterated local search* is outlined. The second example is devoted to *multilevel techniques*.

2.1. Example 1: Population-Based Iterated Local Search

In contrast to the use of local search methods within population-based methods, recent years have witnessed the appearance of some algorithms that result from the enhancement of metaheuristics based on local search with concepts from population-based approaches. An example is *population-based iterated local search* [31] where iterated local search is extended from working on a single solution to working on a population which is managed in the style of evolution strategies. The resulting algorithm, which we will shortly outline in the following, is very competitive for the quadratic assignment problem (QAP).

Iterated local search (ILS) [32, 33] is a metaheuristic based on a simple idea. Instead of repeatedly applying local search to randomly generated starting solutions, an ILS algorithm produces the starting solution for the next iteration by perturbing an incumbent solution. This is done in the expectation that the perturbation mechanism provides a solution located in the basin of attraction of a local minimum that is better than the incumbent solution, but close in distance. The pseudo-code of ILS is shown in Algorithm 1. It works as follows. First, an initial solution is generated in function GenerateInitialSolution(). This solution is subsequently improved by the application of local search in function LocalSearch(s). The construction of initial solutions should be fast (computationally not expensive), and—if possible—initial solutions should be a good starting point for local search. At each iteration, the incumbent solution $\hat{s}$ is perturbed in function Perturbation($\hat{s}$,$history$), resulting in a perturbed solution s'. The perturbation is usually non-deterministic in order to avoid cycling. The importance of the perturbation mechanism is obvious: On the one side, a perturbation that is not strong enough might not enable the algorithm to escape from the basin of attraction of the current solution. On the other side, a perturbation that is too strong would make the algorithm similar to a random restart local search. After the application of local search to the perturbed solution, the resulting solution $\hat{s'}$ may either be accepted as new current solution, or not. This is decided in function ApplyAcceptanceCriterion($\hat{s'}, \hat{s}, history$). Two extreme examples are (1) accepting the new local minimum only in case of an improvement and (2) always accepting the new solution. In between, there are several possibilities. For example, an acceptance criterion that is similar to the one of simulated annealing can be adopted.

The extension to population-based ILS is quite simple. Instead of working on a single incumbent solution, population-based ILS maintains at all times a population P of n solutions. The usual ILS-steps, that is, perturbation and

Algorithm 1 Iterated Local Search

```
1: s ← GenerateInitialSolution()
2: ŝ ← LocalSearch(s)
3: while termination conditions not met do
4:    s' ← Perturbation(ŝ, history)
5:    ŝ' ← LocalSearch(s')
6:    ŝ ← ApplyAcceptanceCriterion(ŝ', ŝ, history)
7: end while
```

Algorithm 2 Population-Based Iterated Local Search

```
1: P ← GenerateInitialPopulation(n)
2: Apply LocalSearch() to all s ∈ P
3: while termination conditions not met do
4:    P' ← P
5:    for all s ∈ P do
6:       s' ← Perturbation(s, history)
7:       ŝ' ← LocalSearch(s')
8:       P' ← P' ∪ {ŝ'}
9:    end for
10:   P ← Best n solutions from P'
11: end while
```

the subsequent application of local search are, at each iteration, applied to each solution $s \in P$. Adding all solutions generated in this way to P results in an augmented population P' of $2n$ solutions. The population for the next generation is then obtained by removing the worst n solutions from P'. The pseudo-code of this procedure is shown in Algorithm 2.

When to use this technique? As a simple extension of ILS, population-based ILS can be applied to any problem for which a neighborhood structure for local search can be designed. We expect population-based ILS to improve over pure ILS especially in those cases in which high-quality local optima are scattered all over the search space. This is the case for the QAP, as opposed to other problems with a high fitness-distance correlation.

2.2. Example 2: Multilevel Techniques

Multilevel techniques [34, 35] are heuristic frameworks that were introduced in particular for dealing with large-scale problem instances. Their origins are so-called multigrid methods [36]. Walshaw et al. were among the first ones to apply multilevel techniques to combinatorial optimization problems. Among the earliest applications were the ones to mesh partitioning [37], the traveling salesman problem [38], and graph coloring [34].

The working of multilevel techniques is based on the following simple idea. Taking as input an original problem instance, smaller and smaller instances are

generated by successive coarsening until some stopping criteria are met. This process provides a hierarchy of problem instances. Hereby, the problem instance corresponding to a certain level is always smaller than (or of equal size as) the problem instance corresponding to the next higher level. After this coarsening process, an optimization technique (such as, for example, a metaheuristic) is used to generate a solution to the smallest problem instance. This solution is successively transformed into a solution to the problem instance corresponding to the next level until a solution to the original problem instance is obtained. Note that these solutions may be subject to a refinement process at each level. Any improvement technique may be used for this refinement process, but most often a metaheuristic is applied. The working of multilevel techniques is graphically illustrated in Fig. 1. More sophisticated multilevel approaches may even use the best solution produced by the above-mentioned method for guiding a successive re-coarsening, which is again followed by the refinement process. Both phases are iterated to obtain even better solutions [35].

When to use this technique? In general, the application of a multilevel framework is only recommended if an efficient and not excessively complicated way of coarsening problem instances can be found. This is the case, for example, when graph-based problems are considered. In these cases, the coarsening of a problem instance may be done by means of edge or node contractions. However, it is not only important to be able to identify a way of coarsening a problem instance, it is equally important that this way of coarsening a problem instance approximates the corresponding backbone. Roughly, the backbone of an optimization problem consists of the set of solution components that are present in high-quality solutions. A recent survey over existing applications is given in [35]. Applications that are not covered by this survey include [39, 40].

2.3. Literature Overview

In addition to the two examples that have been outlined above, the literature offers a multitude of different hybridizations between metaheuristics and other (meta-)heuristic methods. Other examples of rather unusual hybrids—similar to population-based iterated local search—are proposed in the paper by Lozano and García-Martínez [41], where the authors use an evolutionary algorithm as a perturbation technique for iterated local search. Moreover, Resende et al. [42] devise several versions of a hybrid algorithm based on greedy randomized adaptive search procedures (GRASP) and path relinking methodologies for the max-min diversity problem. One example is *evolutionary path relinking* where the pool of elite solutions is evolved in order to be both diverse and of high quality.

Variable fixing strategies are to some extent related to multilevel techniques. Hereby, variables of the original problem are fixed to certain values (according to some criterion) and optimization is performed over the resulting restricted search space. Examples of effective variable fixing strategies are the *core concepts* for knapsack problems [43, 44]. Another example where variable fixing is essential is the variable neighborhood decomposition approach proposed in [45]. *Problem*

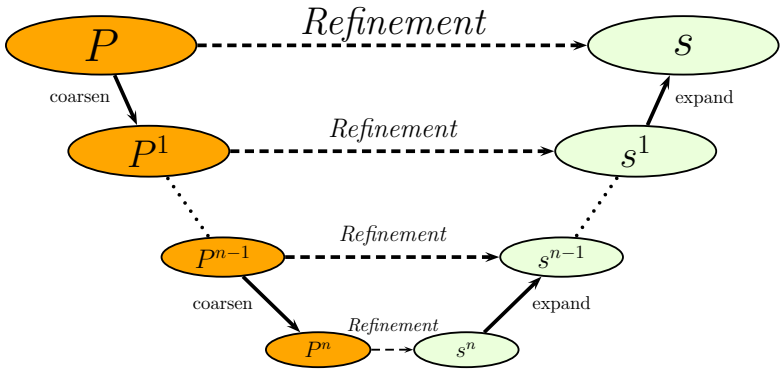

Figure 1: The principle of a multilevel technique. The original problem instance is labelled P. This problem instance is iteratively simplified until a stopping criterion is satisfied at some level n corresponding to instance P^n. Then, a metaheuristic may be applied for the generation of a solution s^n to instance P^n, which is subsequently expanded into a solution $s^{n-1'}$ to instance P^{n-1} of level $n-1$. The refinement of solution $s^{n-1'}$ may be done by a metaheuristic again. This process is stopped once a solution to the original problem instance has been obtained.

kernelization, which is a systematic approach based on tools from the field of parameterized complexity, is also related to multilevel strategies and variable fixing. The basic idea of this approach is to reduce a given problem instance in polynomial time to a so-called problem kernel such that an optimal solution to the problem kernel can be transformed in polynomial time to an optimal solution for the original problem instance. In [46], Gilmour and Dras proposed an ant colony optimization algorithm that makes use—in several different ways—of the above mentioned problem kernels.

One of the main arguments in favor of metaheuristics has always been their generality. In principle metaheuristics may be applied to any combinatorial optimization problem. However, over the years the focus of many metaheuristic applications has shifted toward performance, at the cost of loosing generality. Research on so-called *hyper-heuristics* [47] has started with the idea of developing general algorithms that can potentially be applied to many related problems without much effort of adaptation. The aim is to raise the level of generality at which optimization systems operate. Hyper-heuristics do not directly operate on the search space of the problem under consideration. Instead, they act on a search space defined by lower-level heuristics—or even metaheuristics—for the tackled problem. Hyper-heuristics are broadly concerned with selecting the right (meta-)heuristic at any situation.

Another heuristic framework for potentially improving metaheuristics is the so-called *proximate optimality principle* (POP), which was introduced by Glover and Laguna in the context of tabu search [48]. It is based on the intuition that good solutions are likely to have parts in common and can therefore be found close to each other in the search space. Fleurent and Glover made use of this principle in [49] in the context of partial solutions generated by GRASP. The idea was that bad decisions made during the construction process may

be undone by applying local search during (and not only at the end of) the GRASP construction phase. They proposed a practical application of POP within GRASP by applying local search at certain stages of the construction phase. Other examples can be found in [50, 51].

A different branch of hybridization is concerned with enhancing metaheuristics with additional techniques for decreasing run-time, improving on the results, or both. Montemanni and Smith [52] introduced an approach based on tabu search for solving the frequency assignment problem. Hereby, tabu search is enhanced by *heuristic manipulation*, a principle which is based on adding constraints to a problem having the effect of reducing the search space. This, in turn, may facilitate the solution of the problem under consideration. Another example is the paper by Chaves et al. [53] dealing with *clustering search*, which works roughly as follows: first, a metaheuristic is used to create a set of initial solutions. These solutions are then clustered. As final step, local search is used to find possibly better solutions within the cluster areas.

Finally, it is worthwhile to mention some rather unusual hybridizations between different metaheuristics. In [54], Chen et al. propose the combination of genetic algorithms with *extremal optimization* [55], which is inspired by self-organized critical models of co-evolution abstracted from the fundamentals of ecosystems. The search process of extremal optimization tries to eliminate those solution components that can be associated to bad performance. Another example concerns the work proposed in [56], where the authors propose the combination of genetic algorithms and particle swarm optimization. The last example concerns an algorithm for the arc routing problem proposed in [57], which is obtained by enhancing tabu search with scatter search principles. However, note that this is just a representative sample of many different ways of combining different metaheuristics.

3. Hybridizing Metaheuristics With Constraint Programming

Metaheuristics and constraint programming (CP) are two quite different problem solving techniques. Each one is usually applied with success on problem classes for which the other is not particularly effective. In CP, constrained optimization problems are modelled by means of variables, domains[1] and constraints, which can be mathematical (as for example in linear programming) or symbolic. Constraints encapsulate well-defined parts of the problem into sub-problems. Each constraint is associated to a *filtering* algorithm that deletes values from a variable domain which do not contribute to feasible solutions. The solution process of CP is characterized by an interleaving of a *propagation* phase in which values are removed from domains by means of the filtering algorithms, and a *labelling* phase in which an unassigned variable is chosen and assigned a value in its domain. In case of failure (e.g., an empty domain), the search backtracks. When an optimization problem is tackled, a bound constraint on

[1] We restrict the discussion to finite domains.

the cost function is posted every time a new improving solution is found. This way, non-improving assignments are considered infeasible. Metaheuristics usually explore a search space in which states are defined by complete—possibly infeasible—assignments and are mostly guided by local information on the objective function. Conversely, the strength of CP lies in its capability of exploring a search space of partial assignments and finding a solution that satisfies the problem constraints. Also CP makes use of heuristic criteria, for example, for choosing the variable/value assignment to extend a partial solution. Summarizing, metaheuristics have been proven effective in finding good-quality solutions to optimization problems, while they are usually not very powerful in tackling constraint satisfaction problems. On the other side, CP is extremely effective in solving decision problems, while it performs quite poorly on optimization problems with large feasible spaces and loose bounds on the objective function. These two methods have complementary strengths and it is therefore quite natural to try to combine them in order to exploit possible synergies.

In this section, we outline two among the most paradigmatic examples of combinations of metaheuristics and constraint propagation techniques.[2] These two examples are chosen on purpose as they have a quite different nature and they show two of the main perspectives from which the integration of metaheuristics and CP can be conceived.

3.1. Example 1: CP-based Large Neighborhood Search

CP-based large neighborhood search (LNS) denotes a family of problem solving techniques in which local search uses CP for exploring a, typically very large, neighborhood. LNS tries to combine the advantage of a large neighborhood, that usually enhances the explorative capabilities of local search, with an exhaustive CP exploration that is faster than enumeration, especially when most problem variables are already assigned. LNS was first proposed by Shaw in [58] and similar ideas were presented in [59, 60, 61].

The core principle of LNS consists in viewing the exploration of a neighborhood as the solution of a sub-problem. As an example, let us suppose we want to solve a vehicle routing problem with n_t trucks, each starting and ending its tour in a depot. The trucks have to visit n_c customers within a given time window at a minimum cost. Given the current solution, one may fix all routes but one and optimally solve by CP on the free variables a travelling salesman problem with time windows and possibly further side-constraints. The process of fixing part of the current solution (i.e., defining a partial assignment) and finding its optimal completion is guided by a local search that employs CP to find the best candidate in the neighborhood.

In general, let us assume that the problem to be solved is modeled with a set X of n variables $x_1, \ldots, x_n$ which can assume values in discrete and finite domains $D_1, \ldots, D_n$. The feasible set of solutions $\mathcal{S}$ is defined by those

[2]Examples concerning the combination of metaheuristics and general tree-search will be discussed in Section 4.

Algorithm 3 LNS high level algorithm

1: **Input:** Problem $\mathcal{P}(X, D, C, f)$ defined by variables $X = \{x_1, \ldots, x_n\}$, domains $D = \{D_1, \ldots, D_n\}$, constraints $C = \{c_1, \ldots, c_m\}$, objective function f to be minimized
2: **Output:** Best solution found
3: $s \leftarrow \mathsf{InitialSolution}()$; let $s \equiv \{(x_1 = v_1), \ldots, (x_n = v_n)\}$
4: **while** termination condition not met **do**
5: $K \leftarrow \mathsf{Select}(n, k)$ {Define neighborhood $\mathcal{N}_k$ by selecting the indices of k variables}
6: Let $\mathcal{P}_{|K}$ be the problem in which variables x_i with $i \in K$ are free and $x_i = v_i$ if $i \notin K$
7: $s' \leftarrow \mathsf{SolveCP}(\mathcal{P}_{|K})$ {Exhaustively explore $\mathcal{N}_k$ by CP and find solution s' *locally* optimal w.r.t. $\mathcal{N}_k$}
8: **if** $f(s') < f(s)$ **then**
9: $s \leftarrow s'$
10: **end if**
11: **end while**

assignments satisfying the constraints $c_1, \ldots, c_m$. The goal is to find a (nearly) optimal solution w.r.t. an objective function f defined over $\mathcal{S}$. Suppose that the goal is to minimize f. A complete assignment to the variables in X is denoted by $s \equiv \{(x_1 = v_1), \ldots, (x_n = v_n)\}$, where $x_i = v_i$ means that variable x_i is assigned value $v_i \in D_i$. A partial assignment is simply a subset of s. The high level LNS search scheme (see Algorithm 3) starts by generating an initial solution (line 3), that is, a feasible complete assignment s. Then, the main local search cycle is repeated until a termination condition is satisfied. For simplicity, but without loosing generality, let us suppose to apply a first improvement local search—whereas in general, any local search may be used. A subset of k variables is chosen so as to define the neighborhood $\mathcal{N}_k$ (line 5). Let us denote by K the set of indices corresponding to the selected variables. The corresponding sub-problem $\mathcal{P}_{|K}$ is then solved by CP (line 7). The solution s' returned is the optimal completion of the partial assignment $\overline{s} = \{(x_i = v_i) \mid i \in \{1, \ldots, n\} - K\}$. This step corresponds to the neighborhood exploration. Then, the current solution s is compared to the new one s' and it is possibly replaced by it (line 8).

The crucial part of the algorithm is the choice of the k variables which will define the sub-problem to be solved. This choice is usually made on the basis of both heuristic and stochastic criteria. In general, the sub-problem/neighborhood size should be large enough to diversify the search, but the complexity of solving it should be rather low because this operation is performed in each iteration of the algorithm. It has also to be noted that, when CP is applied to extend a partial assignment into a complete one, the propagation of constraints is usually effective and deep, due to the variables that are already instantiated. Indeed, this is one of the main reasons for the effectiveness of LNS.

The high level algorithm we have illustrated shows the usage of a simple

first improvement local search, whilst, in general, more elaborated local search strategies can be used. For example, tabu search can be effectively applied in this context, especially because the tabu conditions can be posted as constraints.

Since its proposal, LNS has been further improved in [62], in which a more efficient mechanism for combining local search and CP has been proposed. Moreover, in [63] LNS is enhanced by a method for automatically generating neighborhoods by means of constraint propagation. LNS also shares some similarities with hybrid approaches in which mathematical programming tools or dynamic programming are employed for exploring the neighborhood (see, for example [64, 65] and Sections 4.2 and 6.1).

When to use this technique? The choice of CP-based LNS as a method for tackling a given problem should primarily be conditioned to the efficiency of solving the sub-problem corresponding to the neighborhood exploration. Therefore, the first point to consider is whether it is possible to define the neighborhood structure in such a way that the resulting sub-problem can be efficiently solved to optimality by CP. This is often the case, for example, when neighborhoods are defined by means of additional constraints which fix some parts of a complete assignment or when the problem has a large number of side constraints. Evidence for the possible advantage of using a large neighborhood to be explored by CP—as opposed to a (small) neighborhood searched by enumeration—can be supported by the observation that, for the problem at hand, a local search exploring the neighborhood by enumeration often stagnates in confined areas of the search space.

3.2. Example 2: Ant Colony Optimization and Constraint Programming

Ant colony optimization (ACO) and CP are constructive techniques with complementary strengths: ACO is characterized by a (reinforcement) learning capability, while CP is efficient in handling constraint.[3] The combination of ACO and CP has recently received more interest especially because of the availability of tools for integrating CP and local search. Moreover, a general framework for combining ACO and CP, along with new insights on the possible ways for integrating these two methods, is the subject of a recent book by Solnon [66].

ACO is a population-based metaheuristic inspired by the ant foraging behavior and further formalized as a model-based search metaheuristic [67, 68]. In ACO, solutions are iteratively built in parallel by a probabilistic constructive procedure. The parameters of the probabilistic model upon which the procedure is based are dynamically adjusted by using a learning mechanism, very similar in spirit to reinforcement learning. In ACO terminology, it is common

[3]It has to be noted that the "learning capability" of ACO is conceptually different from what is called the "no-good learning" typical of CP, which consists in adding to the initial instance constraints representing sets of infeasible assignments.

Algorithm 4 CP-with-ACO

Input: Variables $x_1, \ldots, x_n$, domains $D_1, \ldots, D_n$, constraints $c_1, \ldots, c_m$
Output: A feasible assignment, optimal w.r.t. a given objective function or **nil** if no feasible solution exists

```
Setup initial domains for variables x_1, ..., x_n
Post initial constraints
while search not completed do
  for each ant do
    Propagate_And_Label(x_1, ..., x_n)
  end for
  Update_Probabilistic_Model()
  if new best solution found then
    Post upper bound constraint
  end if
end while
```

to denote the construction of one solution by the action of an *ant* which iteratively adds a solution component to the current partial solution. The choice of the component is stochastic and biased toward components with a higher value of *pheromone*, which accounts for the attractiveness of a solution component. Once the solution is built, its components are rewarded by adding a quantity of pheromone positively correlated with solution quality. This operation increases the probability that a component belonging to a high-quality solution is chosen in the successive iterations.

In the following, we succinctly illustrate the core idea of the hybridization of ACO and CP proposed by Meyer in [69], in which this combination is described along with experimental results for a machine scheduling problem with sequence-dependent setup times. Successively, we outline the other main approaches for achieving such integration.

Let us consider Algorithm 4, which is a slight variation of the original one from [69], in which we show a general scheme of a solver based on CP and ACO. The algorithm is a variant of a classical CP search. The main differences are the procedure Propagate_And_Label($x_1, \ldots, x_n$) that makes use of the probabilistic construction mechanism of ACO for variable and value ordering and the procedure Update_Probabilistic_Model() that updates the pheromone values. The combination of ACO and CP can be achieved by conceiving ACO as the main solution construction procedure and viewing CP as a tool employed by the ants while constructing a solution. In fact, the usual approach for constraint handling in ACO—and metaheuristics in general—is to relax (a subset of) the problem constraints and penalize complete solutions that violate such constraints. This procedure might not be very effective, especially in case of tightly constrained optimization problems. Therefore, if ants use CP for finding a feasible solution, search is concentrated on finding a good-quality solution among the feasible ones and a large amount of computational effort can be saved. It is crucial at

this point to observe that, in this algorithmic scheme, the probabilistic/greedy decision mechanism of ACO comes into play in the context of variable and value selection. In other words, while CP provides filtering, ACO is in charge of performing labelling.

Besides the application to constrained optimization problems, a hybrid method combining ACO and CP can also be applied to constraint satisfaction problems (CSPs). In this case, the goal is to find any complete assignment satisfying all the constraints. As proposed by Prestwich in [70], one of the possible ways of achieving this integration consists in using a metaheuristic algorithm for searching the space of feasible (partial) assignments, trying to maximize the number of assigned variables. A solution to the problem is found when all the variables are assigned. This idea has been used to combine ACO and CP in a hybrid algorithm successfully applied to the car sequencing problem in [71].

Finally, another very promising way of integrating ACO and CP has been presented in [72]. In this work, the problem is solved by a two phase algorithm: in the first phase, a typical ACO is performed and the final pheromone matrix is saved. In the second phase, the CP solver performs a complete search and uses the pheromone matrix as a heuristic information for value selection. A similar idea, which makes use of initial samplings, has been proposed in [73].

When to use this technique? The problems for which the combination of ACO and CP is expected to be effective are those constrained optimization problems for which two conditions hold: *(i)* finding a feasible solution by local search methods is very hard and *(ii)* the space of feasible solutions is large. On one side, when the first condition holds, CP is a promising candidate method for tackling the feasibility part of the problem. On the other side, when the feasible space is large, CP alone is not efficient in finding a good solution and a heuristic method can considerably enhance the performance of the overall technique. In case the two conditions mentioned above are not fulfilled, the combination of ACO and CP might not be particularly effective.

3.3. Literature Overview

The literature on hybrid methods which combine metaheuristics and CP is quite rich and publications range from theory of algorithms to applications. Focacci et al. [74] illustrate the main possibilities for integrating metaheuristics and CP. According to their classification, the combination of metaheuristics and CP can be mainly achieved by interleaving the two methods, by using one as a subordinate technique to the other, or by including some algorithmic component from one of the two methods into the other.

The first approach can be seen as an instance of cooperative search, in which metaheuristics are applied before CP, providing a valuable input, or vice versa. Besides the already mentioned work by Solnon [72], a prominent example of integration in which metaheuristics are run in a first search phase in order to provide heuristic guidance for CP is provided by Solution-Guided Multi-Point Constructive Search [75]. The algorithm starts by building a set of elite solutions which are then used to guide CP solution construction; this set of solutions

is also updated during search and it can be initially derived by collecting the best solutions returned by several runs of the best performing metaheuristic algorithms for the problem at hand. The combination of the solution-guided value heuristic and CP search, with a bound on the objective function, provides an effective and robust solver, as indeed proved by recent results on the job shop scheduling problem [76]. Recently, also evolutionary computation and neural networks have been combined with CP in a hybrid stochastic constraint programming framework [77], in which the goal is to find a policy tree that specifies the decision variable assignments for any (or just a sample of the) possible scenarios. The proposed approach defines the policy by means of a parameterized model, implemented as a neural network. The input of the network is the current partial policy tree and the output is the suggested value for the variable to assign. The neural network is trained by a genetic algorithm and it is used as a heuristic guidance for CP.

The second hybridization approach combines the advantages of a fast search space exploration by means of a metaheuristic with the efficient neighborhood exploration performed by a systematic method. A prominent example of such a kind of integration are large neighborhood search and related approaches and CP-based local branching [78]. Further examples can be found in [60, 61, 79].

The third approach consists in designing hybrid methods by composing algorithmic components both from metaheuristics and CP. A prominent example of this approach is the class of non-systematic backtrack searches, which preserve the search space exploration based on a systematic search (such as tree search), but sacrifice the exhaustive nature of the search. The hybridization is usually achieved by integrating concepts and mechanisms developed for metaheuristics (e.g., probabilistic choices, aspiration criteria, heuristic construction) into tree search methods. For example, instead of a chronological backtracking, a back-jumping based on search history or information retrieved from local search samples can be performed [80, 81, 82, 83]. Another approach, proposed by Dell'Amico and Lodi in [84], is based on the integration of tabu search machinery into CP search. Other examples of this kind of integration can be found in [85, 86, 87].

The implementation of methods integrating metaheuristics and CP are currently supported by software tools which enable the designer to combine both CP and (meta-)heuristic search strategies in the same framework. Among the most known tools we mention the IBM ILOG Solver[4] and Comet[5].

4. Hybridizing Metaheuristics with Tree Search Techniques

The hybridization of metaheuristics with tree search techniques is probably one of the most popular lines of combining different algorithms for optimization. This is because several metaheuristics as well as some of the most prominent

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

complete algorithms are members of the class of *tree search* techniques. In general, optimization techniques may be classified by their way of exploring the search space of the problem at hand. Tree search techniques consider the search space of an optimization problem in form of a tree. Such a search tree is (sometimes only implicitly) defined by a mechanism for the extension of partial solutions. Each path from the root node to one of the leafs corresponds to the construction of a candidate solution. Inner nodes of the tree correspond to partial solutions. The move from an internal node to one of its children is an extension of the corresponding partial solution, also called a *solution construction step*.

As already mentioned above, the class of tree search techniques contains approximate algorithms such as (meta-)heuristics, but also many of the complete techniques. Specific examples of approximate algorithms are greedy heuristics, extensions of greedy heuristics such as roll-out methods [88], and construction-based metaheuristics such as ACO [67] and GRASP [89]. The two mentioned metaheuristics are iterative algorithms that make use of the repeated probabilistic construction of solutions. The main difference is that ACO algorithms are based on a learning component, while GRASP algorithms are not. A prominent example of a complete algorithm from the class of tree search techniques is branch & bound, which can also be applied in several heuristic variants such as, for example, in the form of *beam search* [90]. While branch & bound (implicitly) considers all nodes of a certain level of the search tree, beam search is restricted to a predefined number of nodes.

Two representative examples of hybridizing metaheuristics with tree search techniques are presented in the following. In the first example, which is about *Beam-ACO*, branch & bound concepts are used make the solution construction process of ant colony optimization more effective. The second example is about *large neighborhood search* in the context of mathematical programming. This type of algorithm exploits the fact that sub-problems of the original problem instance can often be efficiently solved by complete techniques such as MIP solvers.

4.1. Example 1: Beam-ACO

The construction of solutions based on greedy information has an important disadvantage: while being correct for many (if not most) decisions, the greedy information may sometimes mislead the solution construction process. Metaheuristics such as ACO and GRASP, which are based on the construction of solutions, partially avoid this disadvantage by extending partial solutions according to probabilistic decisions. Moreover, the learning component of ACO—over time—may undo misleading greedy information. However, as both metaheuristics are still biased by greedy information, the danger of being misled persists, even though to a less extent. Another disadvantage of construction-based metaheuristics—this time in comparison to complete techniques—is the fact that mechanisms for reducing the search space are generally not employed. With both disadvantages in mind, a recent line of research promotes the incorporation of features from branch & bound derivatives such as beam search into

construction-based metaheuristics. Examples are probabilistic beam search [91], incomplete and non-deterministic tree search (ANTS) procedures [92, 93, 94], and Beam-ACO algorithms [95, 96, 97]. Exemplary, the working of Beam-ACO is outlined in more detail in the following.

As already mentioned in Section 3.2, ACO algorithms have the following basic way of working. A certain number of solutions is constructed independently from each other at each iteration. This is done probabilistically by means of a so-called pheromone model, which is a set of numerical values that are generally associated to appropriately defined solution components. After solution construction, some of the generated solutions are used for updating the values of the pheromone model, that is, the pheromone values. The aim of this procedure is to shift the probability distribution defined by the pheromone values to high-quality areas of the search space. In contrast to ACO, the central idea of beam search is to allow the extension of partial solutions in parallel, in potentially more than one way. At each step, the algorithm chooses maximally $\lfloor \mu k_{\text{bw}} \rfloor$ extensions of the partial solutions stored in a set B, called the *beam*. Parameter k_{bw}, known as the *beam width*, limits the size of B, and $\mu \geq 1$ is another parameter of the algorithm. The choice of feasible extensions is performed in a deterministic way using a greedy function assigning a weight to each feasible extension. At the end of each step, the algorithm generates a new beam B by selecting maximally k_{bw} partial solutions from the set of chosen feasible extensions. For doing that, beam search algorithms make use of bounding information. Only the maximally k_{bw} best extensions—with respect to the available bounding information—are included in the new set B. At termination of the algorithm, the best found complete solution is returned.

The main idea of Beam-ACO is the *non-independent and parallel* probabilistic construction of several solutions at each iteration, as done by beam search. However, in contrast to beam search, the choice of feasible extensions is done probabilistically in the way of ACO algorithms. This algorithm has the advantage of using two complementary types of information about the problem at hand: greedy information as well as bounding information. The potential benefits of using bounding information in addition to greedy information can be easily explained by means of a simple example: Let us consider the search tree shown in Figure 2. The unique optimal solution is depicted in light gray. For simplicity let us assume that all extensions have the same greedy value. Moreover, let us assume that the available bounding information excludes the black nodes from being further examined. Based on the greedy values we may now assign the probability of 0.5 to all extensions of partial solutions. Based on these probabilities we now consider a probabilistic solution construction process. On the one side, an algorithm not considering bounding information has, for each solution construction, a probability of 0.0625 to generate the unique optimal solution. On the other side, a (probabilistic) beam search algorithm with $k_{\text{bw}} \geq 2$ will solve this problem in only one run. A recent article [98] gives theoretical evidence of the advantage of algorithms using the non-independent and parallel probabilistic constructions of solutions over algorithms that only employ the repeated independent probabilistic construction of solutions.

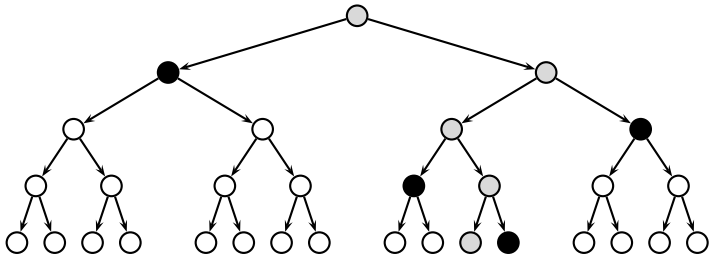

Figure 2: Search tree example. Assume that the solution highlighted by light gray nodes is the unique optimal solution. Moreover, black nodes indicate that bounding information excludes them from the search process.

When to use this technique? Several pre-conditions must be satisfied for a successful application of Beam-ACO. First, the problem at hand must be suitable for the application of ACO. A good indication for this is the existence of successful greedy algorithms. Second, it must be possible to identify bounding information that is computionally inexpensive. This is because Beam-ACO algorithms employ a probabilistic beam search at each iteration, which means that bounding information must be computed many times. Finally, the bounding information should not be misleading in the sense that the bound should correctly identify the best partial solution most of the times, when comparing the bounds of two partial solutions of equal size. On the other hand, it is not necessarily required that the bound is tight.

4.2. Example 2: Large Neighborhood Search Based on Mathematical Programming

For many combinatorial optimization problems the field of mathematical programming and (mixed) integer linear programming (MIP) in particular provides powerful tools; for comprehensive introductions into this area see e.g. [99, 100]. MIP-solvers are in general based on a tree search framework but further include the solution of linear programming relaxations of a given MIP model for the problem at hand (besides primal heuristics) in order to obtain lower and upper bounds. To tighten these bounds, various kinds of additional inequalities are typically dynamically identified and added as cutting planes to the MIP-model, yielding a *branch & cut* algorithm. Frequently, such MIP approaches are highly effective for small to medium sized instances of hard problems; however, they often do not scale well enough to large instances relevant in practice.

Similarly to CP, see Section 3.1, MIP might therefore be very useful for searching large neighborhoods within a metaheuristic framework. Especially the availability of effective general purpose MIP-solvers such as IBM ILOG

CPLEX[6], GUROBI[7], XPRESS[8], or the freely available SCIP[9] and their relatively easy applicability makes this approach particularly interesting in practice, providing the problem at hand can be expressed by a MIP model.

The large neighborhoods that are to be solved by a MIP-solver can be defined in different ways. In the simplest case, an appropriate portion of the decision variables is fixed to the values they have in an incumbent solution, and only the remaining ("free") variables are optimized by the MIP-solver. If the MIP-solver finds an improved solution, it becomes the new incumbent, a new large neighborhood is defined around it, and the process is iterated. Obviously, the selection of the variables that remain fixed and that are subject to optimization, respectively, plays a crucial role: The number of free variables directly implies the size of the neighborhood. Too restricted neighborhoods—that is, subproblems—are unlikely to yield improved solutions, while too large neighborhoods might result in excessive running times for solving the subproblem by the MIP-solver. Therefore, a strategy for dynamically adapting the number of free variables is sometimes used. Furthermore, the variables to be optimized might be selected either purely at random or in a more sophisticated, guided way by considering the variables' potential impact on the objective function and their relatedness.

For example, Mitrović-Minić and Punnen [101] describe such an approach for solving general mixed integer programming problems, called *variable intensity local search*, and tested it specifically on the generalized assignment problem and its multi-resource variant.

A successful and more problem-specific, practical example for the above definition of large neighborhoods has been described by Prandtstetter and Raidl for the car sequencing problem [102]. Here, the goal is to find a cost-effective arrangement of commissioned cars along a production line, that is, a permutation. Each car requires particular components to be installed by different working bays along the assembly line, and the objective is to smooth the workload at the working bays. More formally, no more than l_c cars are allowed to require component c in any subsequence of m_c consecutive cars, and violations of this constraint are penalized by additional costs in the objective function. Prandtstetter and Raidl describe a generalized variable neighborhood search that makes use of eight different types of neighborhood structures. Besides the more straight-forward simple move and swap neighborhoods, more powerful κ-exchange neighborhoods are considered: A set of κ cars is selected either uniformly at random or by a greedy strategy that prefers cars involved in conflicts, thus, inducing higher costs. These cars are then released from their current positions and reassigned in an optimal way by solving a corresponding MIP. The number κ of cars to be reassigned is varied within the variable neighbor-

[6] http://www-01.ibm.com/software/integration/optimization/cplex-optimizer
[7] http://www.gurobi.com/
[8] http://www.aimms.com/features/solvers/xpress
[9] http://scip.zib.de/

hood search, starting with a small value and thus, small neighborhoods, and increasing it up to 65 when no improved solution can be found. To avoid too long running times for larger κ, the MIP solver is aborted when a certain time limit is exceeded and the so far best solution (if available) is considered. Empirical investigations have shown that the utilization of the MIP-neighborhood substantially improves the overall solution quality.

An alternative way for defining large neighborhoods is to reduce the whole search space by including additional constraints. Among the more generally applicable ones are *local branching constraints* [103], which in the basic version are suited for MIPs with binary variables $(x_1, \ldots, x_n) \in \{0,1\}^n$. For a current incumbent solution $\overline{x} = (\overline{x}_1, \ldots, \overline{x}_n)$ this neighborhood is defined by

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

where $\overline{S}$ represents the index set of the variables that are set to one in $\overline{x}$, i.e. $\overline{S} = \{j = 1, \ldots, n \mid \overline{x}_j = 1\}$. For discrete values $\Delta(x, \overline{x})$ resembles the Hamming distance, and thus, the neighborhood induced by the local branching constraint corresponds to the classical k-opt neighborhood. Parameter k controls the size of the neighborhood and its choice is critical. Frameworks that dynamically adapt k are therefore common, e.g. by utilizing variable neighborhood search [104]. Fischetti and Lodi [103] also showed how the local branching constraints can be generalized to non-binary integer variables. However, the major advantage of local branching constraints—namely that no variables must be explicitly selected for fixing—also comes with a downside: Local branching constraints are dense, i.e. they involve all binary variables, and their inclusion increases the complexity of the MIP model in terms of variables and constraints. In particular, including reverse local branching constraints to exclude already searched neighborhoods from consideration in further iterations has not turned out to be fruitful [105].

In contrast to these relatively general local branching approach, special problem-specific neighborhoods can sometimes be identified which are promising to be searched by MIP methods. For example, Archetti et al. [106] consider the selective arc routing problem with penalties, which is a generalization of the directed rural postman problem in which a minimum cost cycle traversing a subset of arcs at least once is sought; costs arise for unvisited arcs. After performing tabu search, a large neighborhood is defined based on the solutions visited by tabu search: First a minimal tour containing a set of "good" arcs that are most likely contained in an optimal solution, i.e., those arcs that appeared in most of the visited solutions, is built. The large neighborhood, which is then searched by means of a MIP-solver, consists of all possible extensions of this minimal tour by further sequences of so-called "questionable" arcs. Due to the relatively high running time of the MIP-solver, the large neighborhood search is not iterated here, but only applied once as a final refinement phase. Experimental results document the positive impact of this approach.

A similar methodology has been proposed by De Franceschi et al. [107] for the directed capacitated vehicle routing problem. Further special MIP-based neighborhoods have, for example, been described by Oncan et al. [108] for partitioning problems, Ropke and Pisinger [109] for pickup and delivery problems with time windows, and Pirkwieser and Raidl [110] for a Periodic Location-Routing Problem.

When to use this technique? Large neighborhood search by means of MIP solvers is nowadays a relatively frequent approach which is promising in many cases. A particular advantage is the relative ease of application, providing the problem to solve can be conveniently expressed by a MIP model and a corresponding general purpose solver is available. Also, when one first aims at solving the problem exactly by means of a MIP solver and encounters too high running times for practical instances, MIP-based large neighborhood search is an obvious possibility to consider.

4.3. Literature Overview

The hybridization of metaheuristics with tree search techniques is surely one of the most popular hybridization approaches. Instead of trying to mention all the articles that have appeared in this field, we focus on a representative selection of works, different to the ones mentioned already above, that is, different to Beam-ACO and MIP-based large neighborhood search.

The work by Nagar et al. [111] for a two-machine flow-shop scheduling problem was probably one of the first works on the combination of branch & bound with an evolutionary algorithm. The presented algorithm, which acts on permutations of all jobs, may be seen as a *multi-stage approach*. In the first stage of the algorithm, branch & bound is executed up to a fixed tree level k. The calculated bounds are stored at each node of the branch & bound tree. The second stage consists in the execution of the evolutionary algorithm. Hereby, each generated partial solution is mapped to its corresponding tree node. If the bounds indicate that the partial solution cannot be part of an optimal solution, guided mutation operators are applied for changing the partial solution.

Meaningful restricted subproblems that are solved by tree search methods for finding improved solutions do not necessarily have to be defined based on a single incumbent solution only. The concept of *solution merging* is based on the idea of deriving hopefully better solutions from the attributes originating from two or more input solutions. Applegate et al. [112, 113] were among the first to suggest tree search methods in the context of merging and applied it to the traveling salesman problem (TSP): A set of promising solutions is derived by a series of runs of the chained Lin-Kernighan iterated local search. The sets of edges of all these solutions are merged and the TSP finally solved to optimality on this resulting reduced graph. Solution merging further is sometimes used as a replacement for naive crossover operators of evolutionary algorithms. Cotta and Troya [114] discuss this aspect in the context of a more general framework for combining branch & bound with evolutionary algorithms and show the usefulness of identifying optimal offspring for different problems. Eremeev [115]

studies the computational complexity of producing a best possible offspring from two parents for binary representations from a theoretical point of view. He concludes that the optimal recombination problem is polynomially solvable for the maximum weight set packing problem, the minimum weight partitioning problem, and linear Boolean programming problems with at most two variables per inequality. On the other hand, determining an optimal offspring is NP-hard for 0/1 integer programming with three or more variables per inequality, the knapsack problem, set covering, the p-median problem, and others.

So far, we have considered approaches where tree search is used as a subordinate within a metaheuristic. However, the literature also offers examples where metaheuristics are used for guiding the search process of tree search. For mixed integer programming, Rothberg [116] suggests a tight integration of an evolutionary algorithm in a branch & cut based MIP solver. The evolutionary algorithm is applied as branch & bound tree node heuristic in regular intervals, and MIP-based optimal merging is done by first fixing all variables that are common in a set of selected elite solutions. Mutation selects a solution, fixes a randomly chosen subset of variables, and calls the MIP-solver for determining optimal values for the remaining problem. Experimental results indicate that this hybrid can significantly improve the MIP-solver's performance in finding good solutions for very difficult MIPs, and this method therefore has been integrated in the commercial MIP-solver CPLEX in version 10.

An example in which the spirit of local search is used to boost the heuristic power of branch & bound is *diving*. Here, the strategy for selecting the next tree node to be processed is modified in such a way that the search is focused on the neighborhoods of promising incumbent solutions in order to quickly identify high-quality solutions. Danna et al. [117] describe *guided dives*: The branch to be processed next is chosen to be the one in which the branching variable is allowed to take the value it has in an incumbent solution. Guided dives are repeatedly applied at regular intervals during the whole optimization process. Again, this concept is included in recent versions of CPLEX.

Further examples are the works by Gallardo et al. [118] and Blum et al. [119], in which the control flows of beam search and a memetic algorithm are intertwined, i.e., phases of beam search and the memetic algorithm alternate. Beam search purges its queue of open partial solutions by excluding those whose upper bounds are worse than the value of the best solution found by the memetic algorithm. On the other side, the memetic algorithm is guided by injecting information about promising regions of the search space identified by beam search into the population.

Apart from branch & bound, metaheuristics have also been hybridized with backtracking techniques. In [120] the authors describe various hybrid metaheuristics applied to problems ranging from car sequencing and graph coloring to scheduling. One example is the application of a tabu search algorithm to the job shop scheduling problem where local search is combined with complete enumeration as well as limited backtracking search. *Nested partitioning* proposed by Shi and Ólafsson [121] is another example were breadth-first search combined with backtracking is used to explore the search space under the guidance of a

metaheuristic. However, the search tree of nested partitioning corresponds to an explicit search space partitioning, rather than an implicit one obtained by variable-value assignments. The obtained sub-spaces are usually evaluated by a metaheuristic. In [122], ACO is applied for this purpose, whereas in [123] local search is used.

5. Hybridizing Metaheuristics With Problem Relaxation

Enhancing metaheuristics with information gained from *problem relaxation* has turned into a quite popular hybridization approach in recent years. Hereby, a relaxed version of a given problem is obtained by simplifying and/or removing constraints. When removing constraints, they may either be dropped, or they may, for example, be transformed to additional terms of the objective function. In case the relaxed problem can be efficiently solved, the hope is that the structure of an optimal solution to the relaxed problem together with its objective function value may facilitate somehow the solution of the original problem. Problem relaxations are also heavily used, for example, in complete techniques such as branch & bound. This is because the optimal solution value of a relaxed problem can be regarded as a bound for the optimal solution value of the original problem, and hence, it can be used for pruning the search tree. An important type of relaxation in combinatorial optimization concerns dropping the integrality constraints of the involved integer variables of a MIP model. The resulting relaxation, which is a linear program (LP), can then be solved to optimality by efficient methods such as the well-known *simplex algorithm*.

In the following we present two examples of hybrid metaheuristics based on problem relaxations. In the first one a search algorithm is guided by Lagrangian relaxation. The second example concerns iterative relaxation based heuristics where LP relaxations and MIP relaxations are used separately and in combination to solve 0–1 mixed integer programming problems.

5.1. Example 1: Hybrid Metaheuristics Based on Lagrangian Relaxation

In [124, 125], Boschetti et al. identify simple metaheuristic frameworks based on the guidance of problem relaxations such as Benders decomposition, the Dantzig-Wolfe decomposition, and Lagrangian relaxation. In fact, it turns out that such algorithms have been used since quite a while in the Operations Research community. However, most of these approaches have not been developed from a metaheuristic perspective. Therefore, the authors of [124, 125] see a large potential for enhancing this type of algorithm with algorithmic components from the metaheuristics field. As an example, we outline a metaheuristic framework based on Lagranging relaxation. For this purpose we start by shortly discussing the main ideas of Lagrangian relaxation. Consider the following general mixed integer program P:

$$z_P := \min\ c_1 x + c_2 y \tag{2}$$

subject to:

$$\begin{aligned} Ax + By &\geq b && (3)\\ Dy &\geq d && (4)\\ x &\geq 0 && (5)\\ y &\geq 0 \text{ and integer} && (6) \end{aligned}$$

Hereby, x is the vector of continuous variables, y is the vector of integer variables, b and d are constant vectors and A, B, and D are matrices. Moreover, z_P is the optimal solution value of problem P. A Lagrangian relaxation is obtained by moving some of the constraints—for example, constraints (2)—in the following way to the objective function, resulting in a problem labelled as $LR(\lambda)$:

$$z_{LR}(\lambda) := \min\ c_1x + c_2y + \lambda(b - Ax - By) \quad (7)$$

subject to:

$$\begin{aligned} Dy &\geq d && (8)\\ x &\geq 0 && (9)\\ y &\geq 0 \text{ and integer} && (10) \end{aligned}$$

Hereby, λ is a given weight vector with weights greater or equal to zero. These weights are also called the *Lagrangian multipliers*. It is easy to show that $z_{LR}(\lambda) \leq z_P$ for all possible weight vectors. Therefore $LR(\lambda)$ is, for each possible λ, a relaxation of P. In order to obtain the best relaxation possible, it is necessary to find the weight vector $\lambda^* \geq 0$ such that $z_{LR}(\lambda^*)$ is maximal, that is:

$$z_{LR}(\lambda^*) = \max\ \{z_{LR}(\lambda) \mid \lambda \geq 0\} \quad (11)$$

A practical and efficient way of finding a good vector λ of Lagrangian multipliers is the so-called *sub-gradient optimization procedure* (see, for example, [126]), which iteratively solves problems $LR(\lambda)$ and updates the Lagrangian multipliers in a systematic but simple way. This procedure can also be used in the following way for solving the original problem P. At each iteration, the optimal solution (x^λ, y^λ) of the relaxation $LR(\lambda)$ may be used in combination with λ for the generation of feasible solutions to the original problem P. This may be done by means of simple heuristics, or alternatively by means of metaheuristic concepts. This way of tackling a problem, which is sketched in Algorithm 5, was labelled *Lagrangian Metaheuristic* in [124, 125]. The authors of these works also provide example implementations for combinatorial problems such as the single source capacitated facility location problem. A nicely working example of a Lagrangian metaheuristic applied to the generalized assignment problem can be found in [127].

When to use this technique? The potential advantages of hybrid metaheuristics of the type of the Lagrangian metaheuristic over standard metaheuristics can

Algorithm 5 Lagrangian Metaheuristic

1: $\lambda \leftarrow$ InitialLagrangianMultipliers()
2: **repeat**
3: $(x^\lambda, y^\lambda) \leftarrow$ Solve($LR(\lambda)$)
4: $(x, y) \leftarrow$ DeriveFeasibleSolution($(x^\lambda, y^\lambda), \lambda$)
5: $\lambda \leftarrow$ Update(λ)
6: **until** termination conditions are satisfied
7: **output:** the best feasible solution obtained for problem P

be summarized as follows. First, due to the fact that both lower and upper bounds are improved during the search process, quality conditions may be derived for the obtained solutions. In addition, whenever the lower and the upper bounds coincide, optimality conditions are satisfied and the search can safely be terminated. The availability of a constantly improving lower bound also allows the potential pruning of the search space. Finally, a precondition for the use of this hybrid technique is that function Solve($LR(\lambda)$) (see line 3 of Algorithm 5) is not too time consuming.

5.2. Example 2: Iterative Relaxation Based Heuristics

The MIPLIB[10] benchmark library, for example, contains a large amount of 0–1 mixed integer programming problem instances that originate in real-life applications ranging from railway line planning over protein folding to VLSI design. Wilbaut and Hanafi [128] present several iterative relaxation based heuristics to solve 0–1 mixed integer programming problems. They combine LP as well as MIP relaxations into a powerful set of heuristics. These are related to the linear programming based algorithm (LPA) for solving 0–1 integer programs by Soyster et al. [129], that was further enhanced by Hanafi and Wilbaut [130]. The main principle of LPA is to solve, in a first step, the LP relaxation of the original problem. In a second step the variables with integral values in the LP relaxation are fixed and this reduced problem is solved to integer optimality. Finally a cut is added to the problem excluding the already visited search space. This process is repeated until the lower bound and the current best feasible solution have a difference smaller than one.

Wilbaut and Hanafi then introduce three new heuristics based on MIP relaxations, improving the upper bounds and introducing intensification and diversification and thereby possibly improving the lower bounds. The MIP relaxation is obtained by enforcing integrality on a subset of the binary variables only. This leads to the new iterative MIP relaxation algorithm (MIPA). In MIPA, first an LP relaxation of the problem is solved yielding a solution x_{LP}. Secondly, the MIP relaxation where integrality is enforced on those binary variables with non-integral values in x_{LP} is solved, yielding a solution x_{MIPR}. Thirdly, as in the LPA algorithm, all variables with integral values are fixed, forming a reduced

[10] http://miplib.zib.de/

problem that is solved to integer optimality. Finally a cut is added to the problem excluding the already visited search space and all integrality constraints are removed. In the next iteration integrality is enforced on those binary variables with non-integral values in x_{MIPR} of the previous iteration. This process is repeated as in LPA. The main advantage of this method is the solution diversification obtained by the fact that the sets of variables on which integrality is enforced are disjoint from one iteration to the next. In practice, however, the number of binary variables in the MIP relaxation will be limited as will be the number of iterations.

Based on those ideas the authors then propose two new heuristics, the iterative relaxation based heuristic (IRH) and the iterative independent relaxation based heuristic (IIRH). In IRH, both the LP relaxation as well as the MIP relaxation obtained by enforcing integrality on those binary variables that have non-integral values in the LP relaxation are solved at each iteration. The lower bounds obtained by both relaxations are compared and the better one is retained. Furthermore, as in the previous algorithms, the solutions obtained by both relaxations are used to define two reduced problems that are solved yielding a feasible solution and thus an upper bound. Cuts obtained by the solutions of both relaxations are then added to the problem and the process is repeated until a stopping criterion is reached. In IIRH the LP relaxation and the MIP relaxation are independent of each other. The LP and MIP relaxations are working with separate problems to be solved, therefore keeping the obtained cuts separated. Only the current best solution and the lower bound are shared between this interleaved version of LPA and MIPA.

LPA, IRH, and IIRH are evaluated on the multidimensional knapsack problem (MKP) and on a benchmark set of binary MIP problems. New improved solutions for some of the MKP benchmarks as well as encouraging results on the binary MIP problems demonstrate the potential of the algorithms.

The combined use of LP and MIP relaxations in a metaheuristic context is a very promising direction and has already led to many successful algorithmic approaches. In a theoretical article [131] Glover proposes different ways of using cuts obtained from relaxations in metaheuristic algorithms. Among others, the cuts used by Wilbaut and Hanafi are further extended and strengthened in the article. Glover shows how such inequalities can be used to embed target solutions and target objectives as well as to obtain intensification and diversification in metaheuristic search.

When to use this technique? The main advantage to be expected from combining relaxations with metaheuristics is the global problem view that is often achieved by solving a problem relaxation. This allows to lead metaheuristics and local search towards promising regions of the search space and possibly obtain higher quality solutions requiring less run-time. Just like in the case of the first example of this section, this hybrid technique should only be considered if solving the corresponding problem relaxation is not computationally expensive.

5.2.1. Literature Overview

Metaheuristics that are guided by problem relaxation can be found quite frequently in the literature. In the following we present representative examples. A more general overview on combinations of metaheuristics with LP and ILP techniques is given in [13].

A straightforward way to make use of an optimal solution to the LP relaxation of a problem at hand is to directly derive a heuristic integer solution which is feasible for the original problem. Depending on the tackled problem, this can be achieved by simple rounding or by more sophisticated repair strategies. For example, Raidl and Feltl [132] present a hybrid genetic algorithm (GA) for the generalized assignment problem. In their GA, the initial population is obtained by a randomized rounding procedure for generating feasible integer solutions from the LP relaxation. As these solutions are often infeasible, randomized repair and improvement operators are applied as well.

Optimal solutions to LP relaxations may also be exploited for guiding local search or for repairing infeasible candidate solutions. In [133] the multi-dimensional knapsack problem, a popular test-case for hybrid algorithms, is considered. The items are sorted according to increasing LP-values of their corresponding variables. Then, a greedy repair procedure removes the items in this order from the knapsack until all constraints are fulfilled. Finally, a greedy improvement procedure considers the items in reverse order and includes them in the knapsack as long as no constraint is violated. In contrast to the above mentioned approaches, Chu and Beasley [134] present an evolutionary algorithm for the MKP that exploits the dual variable values, coming as a by-product of solving LP relaxations. On the basis of the dual variable values they calculate pseudo-utility ratios for the variables. Interestingly, these pseudo-utility ratios tend to give good indications of the likeliness of the corresponding items to be included in an optimal solution.

A last example for the use of LP relaxations is the algorithm by Vasquez and Hao [135, 136], which was also applied to the MKP. The basic idea consists in solving a series of LP relaxations which are obtained by adding constraints with the aim of biasing solutions towards a certain number of items. This is done in a first phase. Afterwards, in a second phase, tabu search is used to search around the optimal solutions to these relaxed problems. Hereby, tabu search is enforced to search within a certain distance to the non-integral solutions.

In contrast to the above-mentioned examples that use LP relaxations, the hybrid GA for the prize collecting Steiner tree problem by Haouari and Siala [137] makes use of a Lagrangian relaxation. More specifically, it is based on a Lagrangian decomposition of a minimum spanning tree like ILP formulation of the problem. The volume algorithm is used for solving the Lagrangian dual [138]. Afterwards, a GA is applied which exploits information provided by the volume algorithm. The original graph is reduced by cutting edges, meaningful initial solutions are generated, and the objective function is modified by considering reduced costs.

A similar combination of Lagrangian decomposition with genetic algorithms

is described in Pirkwieser et al. [139] in the context of the knapsack constrained maximum spanning tree problem. Moreover, a combination of a Lagrangian relaxation approach and VND, which is based on similar ideas, has been developed by Leitner and Raidl [140] for a real-world fiber optic network design problem.

A different use of Lagrangian relaxation is proposed in Tamura et al. [141], where a job-shop scheduling problem is tackled. Given the MIP formulation of the problem, the domain of the variables is split into sub-domains, which are then indexed. Moreover, the original domains are replaced by the indices of the sub-domains. Then, a GA is applied to this reduced problem version and the fitness of the solutions is estimated by Lagrangian relaxation which gives an indication on the quality of the search space region represented by the corresponding solution. When the GA terminates, an exhaustive search of the region identified as the most promising one is carried out.

Reimann [142] introduces an ACO algorithm for the symmetric TSP where an optimal solution to the minimum spanning tree (MST) relaxation is used for biasing the search of the artificial ants towards edges that form part of the minimum spanning tree. The proposed algorithm is based on computational experience indicating that an optimal solution to the symmetric TSP has about 70-80% of the edges in common with an optimal MST solution.

6. Hybridizing Metaheuristics With Dynamic Programming

Dynamic programming (DP) [143] is an algorithmic scheme for optimization that solves a combinatorial problem as follows. First, the given problem is divided into subproblems. Then a solution to the given problem is obtained by combining the solutions to already solved subproblems into solutions to larger subproblems until the original problem is solved. A crucial point of DP is that the solutions to already solved subproblems are stored. This has the advantage that they do not have to be re-computed every time the solution is required. Basically, an optimization problem must exhibit two properties in order to be solved by DP:

1. Optimal solutions to the problem must contain optimal solutions to subproblems. In this case, a problem is said to show optimal substructure.
2. The space of subproblems should be relatively small. Typically, the total number of distinct subproblems is polynomial in the input size.

The existing literature offers examples for the successful integration of DP with metaheuristics, both in the case of constructive and local search techniques. In this section we illustrate two representative examples of hybrid solvers obtained by integrating DP with metaheuristics: Iterated dynasearch and the corridor method.

6.1. Example 1: Iterated Dynasearch

Iterated dynasearch is a hybrid metaheuristic that uses DP as a neighborhood exploration strategy inside iterated local search [4]. The rationale behind

this integration is the same as for LNS, as described in Sections 3.1 and 4.2. In some cases, DP can make it possible to completely explore a neighborhood of exponential size in polynomial time and space. In this paragraph, we will illustrate the principles of *iterated dynasearch* with respect to its application to the single-machine total weighted tardiness scheduling problem (SMTWTSP) [144]. Further contributions to this work can be found in the recent literature [145, 146].

The SMTWTSP can be defined as the problem of finding the processing order of n jobs on one machine such that the total tardiness is minimized. More formally, for each of the n jobs, a processing time p_j, a positive weight w_j and a due date d_j are given. Jobs are available at time zero and must be processed one at a time without interruption. Once a job ordering is provided, for each job a completion time C_j can be computed, along with its tardiness $T_j = \max\{\mathrm{C_j} - \mathrm{d_j}, 0\}$. Therefore, the function to be minimized is $\sum_{j=1}^{n} w_j T_j$.

For the moment, let us simply focus on the design of a suitable neighborhood structure for a best-improvement local search. A natural neighborhood structure can be defined in terms of job permutations. Any permutation of n objects can be obtained by the repeated application of *swaps*. Each swap consists in exchanging two objects. The resulting neighborhood is called the $2-$exchange neighborhood. In general, the $k-$exchange neighborhood, defined by sequences of swaps involving k objects, has an $O(n^k)$ size. Therefore, for efficiency concerns, usually only the cases of $k \in \{2, 3\}$ are considered.

The *dynasearch swap* neighborhood of a job sequence $\sigma = (\sigma(1), \ldots, \sigma(n))$ is composed of all the permutations of σ that can be generated by a series of *independent* swaps. Two swap moves $\{i, j\}$ and $\{k, l\}$ are independent if $\max\{i, j\} < \min\{k, l\}$ or $\min\{i, j\} > \max\{k, l\}$. This neighborhood has size $2^{n-1} - 1$. However, the independence of moves makes it possible to define a recursive enumeration algorithm based on DP such that the resulting exploration is polynomial in time and space.

Let σ_k be the partial job sequence ordering with minimum total weighted tardiness among the possible allowed orderings of the sequence $(\sigma(1), \ldots, \sigma(k))$ and let $F(\sigma_k)$ be the total weighted tardiness of σ_k. This partial sequence can be obtained from a partial optimal sequence σ_i, $0 \leq i < k$, by adding job σ_k. Two cases must be considered:

1. $i = k - 1$: job σ_k is simply appended to σ_i.
2. $i < k - 1$: job σ_k is first appended to σ_i and then immediately swapped with job $\sigma(i+1)$; hence the final sequence is $(\sigma(1), \ldots, \sigma(i), \sigma(k), \ldots, \sigma(i+1))$.

In both cases, the total tardiness $F(\sigma_k)$ can easily be determined by choosing the minimum of the tardiness values computed as a sum of independent contributions. The best sequence σ_n can be computed recursively by a DP algorithm that runs in $O(n^3)$ and requires $O(n)$ space.[11]

[11] For brevity, we omit the details and point the interested reader to [144].

Algorithm 6 Iterated dynasearch

1: $s \leftarrow$ GenerateInitialSolution()
2: $\hat{s} \leftarrow$ BestImprovement(s ; dynasearch swap neighborhood)
3: **while** termination conditions not met **do**
4: $s' \leftarrow$ Perturbation($\hat{s}$; sequence of random swaps)
5: $\hat{s'} \leftarrow$ BestImprovement(s' ; dynasearch swap neighborhood)
6: $\hat{s} \leftarrow$ ApplyAcceptanceCriterion($\hat{s'}, \hat{s}, history$)
7: **end while**

A best-improvement local search based on the dynasearch neighborhood has, on average, a better performance than a best-improvement local search using the 2−exchange or the 3−exchange neighborhoods. In other words, the average total tardiness of the local optimum returned in the case of the dynasearch neighborhood is lower. Furthermore, this local search can be taken as the inner local search component for an iterated local search (ILS) algorithm [31], as illustrated in Algorithm 6. The algorithm iteratively perturbs the current solution s to provide an initial solution for a best improvement local search[12]. The local optimum found by the local search replaces the current solution s depending on the given acceptance criterion.

When to use this technique? The characteristics of iterated dynasearch are very similar to those of LNS methods in general. In this specific case, since the technique devoted to exploring the neighborhood is DP, it is obvious that a precondition for the applicability of this hybrid method is the availability of an efficient DP approach for solving the sub-problem corresponding to neighborhood exploration. In particular, this sub-problem must have an *optimal sub-structure*, that is, an optimal solution is made of optimal solutions to its sub-parts.

6.2. Example 2: Corridor Method Based on Dynamic Programming

The so-called *corridor method* [147] is a hybrid metaheuristic inspired by DP. It has its origins in attempts to deal with the *curse of dimensionality* [148] in large-scale DP applications. Conceptually, the idea is to optimize the objective function over a corridor constructed around the state trajectory generated by the incumbent feasible solution. The best solution found in this corridor is then chosen to be the new incumbent solution for the next iteration. This process is repeated until the new incumbent solution is identical to the old one. At this point the procedure either stops, or a new incumbent solution is generated in some way, and the search process is continued. The first algorithms of that kind were devised in the context of reservoir control and operation problems (see, for example, [149]). However, the corridor method is not restricted to the use of DP.

[12] In general, any local search algorithm can be used.

Algorithm 7 Corridor Method

```
1: s ← GenerateInitialSolution()
2: while termination conditions not satisfied do
3:    X ← ConstructCorridor(s) {Note that X is a subspace of the search space}
4:    s′ ← ApplyCompleteOptimizationMethod(X)
5:    if f(s′) < f(s) then
6:       s ← s′
7:    else
8:       s ← GenerateNewSolution()
9:    end if
10: end while
11: output: the best solution found
```

In the case of branch & bound, for example, corridors—that is, neighborhoods—are constructed around the incumbent solutions themselves, rather than around the DP state trajectories. This may be done by adding exogenous constraints. Ideally, the neighborhoods should be exponentially large and designed in such a way that the chosen complete method can explore them in (pseudo-)polynomial time. The pseudo-code of a general corridor method is shown in Algorithm 7.

In a way, the corridor method is similar to large neighborhood search (LNS) (see Sections 3.1 and 4.2). However, while—at least the early—LNS approaches were developed with the aim of making local search based metaheuristics more efficient, the corridor method was designed with the aim of supporting complete techniques such as dynamic programming in a heuristic way when applied to large scale problems. In fact, the neighborhoods used in local search based metaheuristic and in early LNS methods (see, for example, [150]) are generally *move-based*. This refers to the fact that the neighborhood around the incumbent solution is usually generated on a topological basis of moves, that is, relatively small changes are applied to the incumbent solution. On the contrary, neighborhoods used in the context of the corridor method are *method-based*, which means that these neighborhoods are designed in order to fulfill the needs and the requirements of the complete optimization technique used to explore the neighborhood.

When to use this technique? The corridor method should be considered as an option in cases in which efficient complete methods are known for solving sub-problems of the original problem at hand. If this is given, the algorithm designer is required to specify a way in which these sub-problems can effectively be utilized for neighborhood exploration. Although not yet widely in use, the corridor method has been successfully applied, for example, to a blocks relocation problem [151], to a DNA sequencing problem [152], and to a pre-marshalling problem [153].

6.3. Literature Overview

Apart from the examples outlined above, a few other hybrids involving DP have been proposed in the literature. In this section we discuss a representative sample of them. In [154], for example, Blum and Blesa present the use of a DP algorithm in two different metaheuristics for the k-cardinality tree (KCT) problem. The general idea of their approaches is not limited to the KCT problem and can, potentially, be used for other subset problems. Basically, the idea is to let the metaheuristic generate objects that are bigger than solutions. Ideally, these objects contain an exponential number of solutions to the problem under consideration. DP is then used to efficiently find for each object the best solution that it contains.

Another example is the article by Hu and Raidl [155], where DP is used in the context of an evolutionary algorithm for obtaining the best solution that can be generated from an incomplete solution. The problem considered in this article is the generalized TSP in which a clustered graph is given and a shortest tour visiting exactly one node from each cluster is required. Hu and Raidl study a representation where solutions are stored as a permutation of the given clusters, representing the order in which the clusters are to be visited. A DP procedure is then used to derive a corresponding optimal selection of particular nodes from each cluster.

The algorithm proposed in [156] combines an evolutionary technique and DP for the application to a dynamic facility layout problem with unequal sizes of departments, which may even change from one period to the next. A number of T evolutionary algorithms is run in parallel, one for each of T periods. In each case, a solution represents a layout for the respective period. However, as a solution to the original problem is a sequence of T periods, the evaluation of a layout of a single period must take into account the best combination of layouts that can be generated given the current populations. This is done by DP. A related approach is presented in [157] for a dynamic plant layout problem. Here, solutions are sequences of layouts for different planning periods. Given several solutions, DP is used as a crossover operator for finding the best combination of the layouts for the different planning periods.

The following examples represent hybrid algorithms based on problem decomposition. In [158], the authors propose a hybrid method combining adaptive memory, sparse DP, and reduction techniques to reduce and explore the search space. First, a bi-partition of the variables is generated, which leads to the identification of small core problems with at most 15 variables. These small problems are solved using the forward phase of DP. The space defined by the remaining variables is explored using tabu search. Hereby, partial solutions are completed using the information stored during the forward phase of DP. The authors indicate that their approach can be seen as a global intensification mechanism, since at each iteration, the move evaluations involve solving a reduced problem implicitly.

The application of DP to subproblems is also proposed in [159], where the authors introduce and tackle a multi-drug cancer chemotherapy model to simulate the possible response of the tumor cells under drug administration. The

objective is to minimize the tumor size under a set of constraints. A so-called adaptive elitist GA is combined with a local search technique called *iterative dynamic programming*. This local search technique works by subdividing the problem into subproblems, and optimizing the subproblems separately by DP.

Another application from the bioinformatics field concerns the approach presented in [160], where the authors tackle the multiple sequence alignment problem. One of the main approaches for multiple sequence alignment uses DP to align sequences as follows. First, two of the sequences are optimally aligned. Then, the outcome is aligned with a third sequence. This process is repeated until all sequences have been considered. In this article, Juang and Su propose the application of particle swarm optimization for improving the alignment result at each step of the afore-mentioned iterative process.

A recent heuristic version of DP labelled *bounded dynamic programming* was proposed in [161] for the simple assembly line balancing problem. Hereby, the number of states is heuristically reduced at each level. In this way, the authors were able to find optimal solutions in a reduced amount of computation time.

Finally, in [162] DP is purely used as a solution decoder in the context of the rectangle packing problem with general spatial costs, which consists in packing given rectangles without overlap in the plane so that the maximum cost of the rectangles is minimized.

7. Discussion and Conclusions

In this article we have provided a survey on the hybridization of metaheuristics with other techniques for optimization. We divided this growing research area into five different lines of hybridization. For each of these lines, two representative examples have been outlined in more detail. In addition, a literature review has been provided for each research line. We hope that this work will serve as a starting point for researchers aiming to develop hybrid metaheuristics. However, we would recommend that, before starting to develop a hybrid metaheuristic, researchers carefully consider whether a hybrid metaheuristic technique is the appropriate solver method for the problem at hand. The following questions should be answered:

1. **What is the optimization goal?** Do I need a reasonably good solution very quickly, or can I afford to spend implementation and computation time in order to obtain very good solutions? If man-power and computation time are critical, hybrid metaheuristics are, in general, not advisable. Only when very good solutions are needed which cannot be obtained by any complete method in a feasible time frame, the development of a hybrid metaheuristic is advised.
2. **Is there still room to improve over the results of existing metaheuristic approaches and/or complete techniques?** In some cases existing pure metaheuristic strategies might work already very well for the problem instances that are to be tackled. Or, alternatively, the problem instances under consideration could be solvable by complete techniques

in a reasonable amount of computation time. In these cases it does not make sense to spend time and effort into the development of a hybrid metaheuristic.

3. **Which type of hybrid metaheuristic might work well for my problem?** Unfortunately, the current state of research does not provide conclusive answers to this question. It is hard to find general guidelines. The process of designing and implementing effective hybrid metaheuristics can be rather complicated and involves knowledge about a broad spectrum of algorithmic techniques, programming and data structures, as well as algorithm engineering and statistics. For the development of well-performing algorithms the authors can only recommend (1) a careful literature search with the aim of identifying the most successful optimization approaches for the problem at hand or for similar problems, and (2) the study of different ways of combining the most promising features of the identified approaches.

For the extraction of useful guidelines for the development of hybrid metaheuristics it will be necessary to improve the research methodology that is nowadays commonly used in the metaheuristics field. Unfortunately, the used research methodology is often characterized by a rather *ad hoc* approach that consists in mixing different algorithmic components without any really serious attempts to identify the contribution of different components to the algorithms' performance. In our opinion, the research community should make an effort to move towards a sound scientific methodology consisting of theoretical models for describing properties of hybrid metaheuristics and using an experimental methodology as done in natural sciences. In fact, among the key points of the engineering process of a hybrid metaheuristic are *scientific testing* [163, 164] and the *statistical assessment* of the results [165].

Researchers interested in this topic can find useful contributions in the literature about Artificial Intelligence and Operations Research addressing the issues of experimental methodology. Besides the already cited papers and book, we mention the well known article by Johnson [166] that can be seen as an introduction to empirical testing from a theoretician's point of view. Furthermore, discussions on the overall experimental methodology or just one of its issues, such as parameter tuning or the statistical assessment of results, can be found in [167, 164, 168, 169].

We are convinced that research on hybrid metaheuristics is still in its early days. In the years to come, most publications on metaheuristic applications will be concerned with hybrids. We hope that this work contributes to give some more structure and guidance to this interesting line of research.

References

[1] C. R. Reeves (Ed.), Modern heuristic techniques for combinatorial problems, John Wiley & Sons, New York, USA, 1993.

[2] F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, Vol. 57 of International Series in Operations Research & Management Science, Kluwer Academic Publishers, 2003.

[3] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: Overview and conceptual comparison, ACM Computing Surveys 35 (3) (2003) 268–308.

[4] H. Hoos, T. Stützle, Stochastic Local Search – Foundations and Applications, Morgan Kaufmann Publishers, 2005.

[5] L. Perron, M. A. Trick (Eds.), Proceedings of CPAIOR 2008 – 5th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Vol. 5015 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2008.

[6] W. J. van Hoeve, J. N. Hooker (Eds.), Proceedings of CPAIOR 2009 – 6th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Vol. 5547 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2009.

[7] A. Lodi, M. Milano, P. Toth (Eds.), Proceedings of CPAIOR 2010 – 7th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Vol. 6140 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2010.

[8] M. J. Blesa Aguilera, C. Blum, C. Cotta, A. J. Fernández, J. E. Gallardo, A. Roli, M. Sampels (Eds.), Proceedings of HM 2008 – Fifth International Workshop on Hybrid Metaheuristics, Vol. 5296 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2008.

[9] M. J. Blesa Aguilera, C. Blum, L. di Gaspero, A. Roli, M. Sampels, A. Schaerf (Eds.), Proceedings of HM 2009 – Sixth International Workshop on Hybrid Metaheuristics, Vol. 5818 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2009.

[10] V. Maniezzo, P. Hansen, S. Voss (Eds.), Proceedings of Matheuristics 2006: First International Workshop on Mathematical Contributions to Metaheuristics, Bertinoro, Italy, 2006.

[11] P. Hansen, V. Maniezzo, M. Fischetti, T. Stuetzle (Eds.), Proceedings of Matheuristics 2008: Second International Workshop on Model Based Metaheuristics, Bertinoro, Italy, 2008.

[12] K. Dörner, et al. (Eds.), Proceedings of Matheuristics 2010: Third International Workshop on Model Based Metaheuristics, Vienna, Austria, 2010.

[13] V. Maniezzo, T. Stützle, S. Voß (Eds.), Matheuristics, Vol. 10 of Annals of Information Systems, Springer Verlag, Berlin, Germany, 2010.

[14] C. Blum, M. J. Blesa Aguilera, A. Roli, M. Sampels (Eds.), Hybrid Metaheuristics – An Emerging Approach to Optimization, Vol. 114 of Studies in Computational Intelligence, Springer Verlag, Berlin, Germany, 2008.

[15] C. Cotta, A study of hybridisation techniques and their application to the design of evolutionary algorithms, AI Communications 11 (3–4) (1998) 223–224.

[16] I. Dumitrescu, T. Stuetzle, Combinations of local search and exact algorithms, in: G. R. Raidl, et al. (Eds.), Applications of Evolutionary Computation, Vol. 2611 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2003, pp. 211–223.

[17] G. R. Raidl, A unified view on hybrid metaheuristics, in: F. Almeida, M. J. Blesa Aguilera, C. Blum, J. M. Moreno Vega, M. P. Pérez, A. Roli, M. Sampels (Eds.), Proceedings of HM 2006 – Third International Workshop on Hybrid Metaheuristics, Vol. 4030 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2006, pp. 1–12.

[18] G. R. Raidl, J. Puchinger, C. Blum, Metaheuristic hybrids, in: M. Gendreau, J. Y. Potvin (Eds.), Handbook of Metaheuristics, 2nd Edition, Vol. 146 of International Series in Operations Research & Management Science, Springer Verlag, Berlin, Germany, 2010, pp. 469–496.

[19] C. Blum, J. Puchinger, G. R. Raidl, A. Roli, A brief survey on hybrid metaheuristics, in: B. Filipič, J. Šilc (Eds.), Proceedings of BIOMA 2010 – 4th International Conference on Bio-Inspired Optimization Methods and their Applications, Jozef Stefan Institute, Ljubljana, Slovenia, 2010, pp. 3–18.

[20] C. Blum, J. Puchinger, G. R. Raidl, A. Roli, Hybrid metaheuristics, in: M. Milano, P. van Hentenryck (Eds.), Hybrid Optimization: The 10 Years of CPAIOR, Springer Verlag, Berlin, Germany, 2010, pp. 305–336.

[21] M. Ehrgott, X. Gandibleux, Hybrid Metaheuristics for Multi-objective Combinatorial Optimization, Ch. 8, Vol. 114 of Blum et al. [14], pp. 221–259.

[22] Z. Michalewicz, P. Siarry, Special issue on adaptation of discrete metaheuristics to continuous optimization, European Journal of Operational Research 185 (2008) 1060–1273.

[23] K. V. Price, R. M. Storn, J. A. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, Springer Verlag, Berlin, Germany, 2005.

[24] D. Molina, M. Lozano, C. García-Martínez, F. Herrera, Memetic algorithms for continuous optimisation based on local search chains, Evolutionary Computation 18 (1) (2010) 27–63.

[25] A. P. Engelbrecht, Fundamentals of Computational Swarm Intelligence, Wiley & Sons, Chichester, England, 2005.

[26] C. Cotta, E.-G. Talbi, E. Alba, Parallel Metaheuristics—A New Class of Algorithms, Wiley & Sons, Hoboken, New Jersey, 2005, Ch. Parallel Hybrid Metaheuristics, pp. 347–370.

[27] E.-G. Talbi, Metaheuristics: From Design to Implementation, Wiley & Sons, Hoboken, New Jersey, 2009.

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

[29] P. Moscato, Memetic algorithms: A short introduction, in: D. Corne, et al. (Eds.), New Ideas in Optimization, McGraw Hill, 1999, pp. 219–234.

[30] N. Krasnogor, J. Smith, A tutorial for competent memetic algorithms: model, taxonomy, and design issues, IEEE Transactions on Evolutionary Computation 9 (5) (2005) 474–488.

[31] T. Stützle, Iterated local search for the quadratic assignment problem, European Journal of Operational Research 174 (3) (2006) 1519–1539.

[32] T. Stützle, Local Search Algorithms for Combinatorial Problems - Analysis, Algorithms and New Applications, DISKI - Dissertationen zur Künstlichen Intelligenz, infix, Sankt Augustin, Germany, 1999.

[33] H. R. Lourenço, O. Martin, T. Stützle, Iterated local search, in: F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, Vol. 57 of International Series in Operations Research & Management Science, Kluwer Academic Publishers, Norwell, MA, 2002, pp. 321–353.

[34] C. Walshaw, Multilevel refinement for combinatorial optimization problems, Annals of Operations Research 131 (2004) 325–372.

[35] C. Walshaw, Multilevel refinement for combinatorial optimisation: Boosting metaheuristic performance, in: Blum et al. [14], pp. 261–289.

[36] A. Brandt, Multilevel computations: Review and recent developments, in: S. F. McCormick (Ed.), Multigrid Methods: Theory, Applications, and Supercomputing, Proceedings of the 3rd Copper Mountain Conference on Multigrid Methods, Vol. 110 of Lecture Notes in Pure and Applied Mathematics, Marcel Dekker, New York, 1988, pp. 35–62.

[37] C. Walshaw, M. Cross, Mesh partitioning: A multilevel balancing and refinement algorithm, SIAM Journal on Scientific Computing 22 (1) (2000) 63–80.

[38] C. Walshaw, A multilevel approach to the travelling salesman problem, Operations Research 50 (5) (2002) 862–877.

[39] I. O. Oduntana, M. Toulouse, R. Baumgartner, C. Bowman, R. Somorjai, T. G. Crainic, A multilevel tabu search algorithm for the feature selection problem in biomedical data, Computers & Mathematics with Applications 55 (5) (2008) 1019–1033.

[40] S. Pirkwieser, G. R. Raidl, Multilevel variable neighborhood search for periodic routing problems, in: P. I. Cowling, P. Merz (Eds.), Proceedings of EvoCOP 2010 – 10th European Conference on Evolutionary Computation in Combinatorial Optimization, Vol. 6022 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2010, pp. 226–238.

[41] M. Lozano, C. García-Martínez, Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: Overview and progress report, Computers & Operations Research 37 (3) (2010) 481–497.

[42] M. G. C. Resende, R. Martí, M. Gallego, A. Duarte, GRASP and path relinking for the max-min diversity problem, Computers & Operations Research 37 (3) (2010) 498–508.

[43] D. Pisinger, Core problems in knapsack algorithms, Operations Research 47 (1999) 570–575.

[44] J. Puchinger, G. R. Raidl, U. Pferschy, The core concept for the multidimensional knapsack problem, in: J. Gottlieb, G. R. Raidl (Eds.), Evolutionary Computation in Combinatorial Optimization – EvoCOP 2006, Vol. 3906 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2006, pp. 195–208.

[45] J. Lazić, S. Hanafi, N. Mladenović, D. Urošević, Variable neighbourhood decomposition search for 0-1 mixed integer programs, Computers & Operations Research 37 (6) (2010) 1055–1067.

[46] S. Gilmour, M. Dras, Kernelization as heuristic structure for the vertex cover problem, in: M. Dorigo, et al. (Eds.), Proceedings of ANTS 2006 – 5th International Workshop on Ant Colony Optimization and Swarm Intelligence, Vol. 4150 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2006, pp. 452–459.

[47] E. K. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, S. Schulenburg, Hyper-heuristics: An emerging direction in modern search technology, in: Glover and Kochenberger [2], pp. 457–474.

[48] F. Glover, Surrogate constraints, Operations Research 16 (4) (1968) 741–749.

[49] C. Fleurent, F. Glover, Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory, INFORMS Journal on Computing 11 (1999) 198–204.

[50] S. Binato, W. J. Hery, D. Loewenstern, M. G. C. Resende, A GRASP for job shop scheduling, in: C. C. Ribeiro, P. Hansen (Eds.), Essays and Surveys on Metaheuristics, Kluwer Academic Publishers, 2001, pp. 59–79.

[51] T. Kanazawa, K. Yasuda, Proximate optimality principle based tabu search, IEEJ Transactions on Electronics, Information and Systems 124 (3) (2004) 912–920.

[52] R. Montemanni, D. H. Smith, Heuristic manipulation, tabu search and frequency assignment, Computers & Operations Research 37 (3) (2010) 543–551.

[53] A. A. Chaves, F. A. Correa, L. A. N. Lorena, Clustering search heuristic for the capacitated p-median problem, in: E. Corchado, J. M. Corchado, A. Abraham (Eds.), Innovations in Hybrid Intelligent Systems, Vol. 44 of Advances in Soft Computing, Springer Verlag, Berlin, Germany, 2008, pp. 136–143.

[54] Y.-W. Chen, Y.-Z. Lu, G.-K. Yang, Hybrid evolutionary algorithm with marriage of genetic algorithm and extremal optimization for production scheduling, The International Journal of Advanced Manufacturing Technology 36 (9-10) (2008) 959–968.

[55] S. Boettcher, A. G. Percus, Nature's way of optimizing, Artificial Intelligence 119 (1-2) (2000) 275–286.

[56] X. H. Shi, Y. C. Liang, H. P. Lee, C. Lu, L. M. Wang, An improved GA and a novel PSO-GA-based hybrid algorithm, Information Processing Letters 93 (5) (2005) 255–261.

[57] P. Greistorfer, A tabu scatter search metaheuristic for the arc routing problem, Computers & Industrial Engineering 44 (2) (2003) 249–266.

[58] P. Shaw, Using constraint programming and local search methods to solve vehicle routing problems, in: M. Maher, J.-F. Puget (Eds.), Principle and Practice of Constraint Programming – CP98, Vol. 1520 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 1998, pp. 417–431.

[59] D. Applegate, W. Cook, A computational study of the job-shop scheduling problem, ORSA Journal on Computing 3 (2) (1991) 149–156.

[60] G. Pesant, M. Gendreau, A view of local search in Constraint Programming, in: E. Freuder (Ed.), Principles and Practice of Constraint Programming - CP'96, Vol. 1118 of Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg, Germany, 1996, pp. 353–366.

[61] G. Pesant, M. Gendreau, A Constraint Programming Framework for Local Search Methods, Journal of Heuristics 5 (1999) 255–279.

[62] P. Shaw, B. De Backer, V. Furnon, Improved local search for CP toolkits, Annals of Operations Research 115 (2002) 31–50.

[63] L. Perron, P. Shaw, V. Furnon, Propagation guided large neighborhood search, in: M. Wallace (Ed.), Principles and Practice of Constraint Programming – CP 2004, Vol. 3258 of Lecture Notes in Computer Science, Springer, 2004, pp. 468–481.

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

[65] M. Chiarandini, I. Dumitrescu, T. Stützle, Very large-scale neighborhood search: Overview and case studies on coloring problems, in: Blum et al. [14], pp. 117–150.

[66] C. Solnon, Ant Colony Optimization and Constraint Programming, Wiley-ISTE, 2010.

[67] M. Dorigo, T. Stützle, Ant Colony Optimization, MIT Press, Cambridge, MA, 2004.

[68] C. Blum, Ant colony optimization: Introduction and recent trends, Physics of Life Reviews 2 (4) (2005) 353–373.

[69] B. Meyer, Hybrids of constructive meta-heuristics and constraint programming: A case study with ACO, Ch. 6, Vol. 114 of Blum et al. [14], pp. 151–183.

[70] S. Prestwich, The Relation Between Complete and Incomplete Search, Ch. 3, Vol. 114 of Blum et al. [14], pp. 63–83.

[71] M. Khichane, P. Albert, C. Solnon, Integration of ACO in a constraint programming language, in: Proceedings of ANTS 2008 – 6th International Workshop on Ant Colony Optimization and Swarm Intelligence, Vol. 5217 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2008, pp. 84–95.

[72] M. Khichane, P. Albert, C. Solnon, Strong combination of ant colony optimization with constraint programming optimization, in: A. Lodi, M. Milano, P. Toth (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems – CPAIOR 2010,

Vol. 6140 of Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg, Germany, 2010, pp. 232–245.

[73] M. Lombardi, M. Milano, A. Roli, A. Zanarini, Deriving information from sampling and diving, in: R. Serra, R. Cucchiara (Eds.), Emergent Perspectives in Artificial Intelligence – AI*IA 2009, Vol. 5883 of Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg, Germany, 2009, pp. 82–91.

[74] F. Focacci, F. Laburthe, A. Lodi, Local search and constraint programming, in: Glover and Kochenberger [2], pp. 369–403.

[75] J. C. Beck, Solution-guided multi-point constructive search for job shop scheduling, Journal of Artificial Intelligence Research 29 (2007) 49–77.

[76] J.-P. Watson, J. Beck, A hybrid constraint programming/local search approach to the job-shop scheduling problem, in: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Vol. 5015 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2008, pp. 263–277.

[77] S. Prestwich, S. Tarim, R. Rossi, B. Hnich, Evolving parameterised policies for stochastic constraint programming, in: I. Gent (Ed.), rinciples and Practice of Constraint Programming – CP 2009, Vol. 5732 of Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg, Germany, 2009, pp. 684–691.

[78] Z. Kiziltan, A. Lodi, M. Milano, F. Parisini, CP-based local branching, in: C. Bessiere (Ed.), Principles and Practice of Constraint Programming – CP 2007, Vol. 4741 of Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg, Germany, 2007, pp. 847–855.

[79] M. Trick, H. Yildiz, A large neighborhood search heuristic for graph coloring, in: P. Van Hentenryck, L. Wolsey (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems – CPAIOR 2007, Vol. 4510 of Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg, Germany, 2007, pp. 346–360.

[80] M. L. Ginsberg, Dynamic backtracking, Journal of Artificial Intelligence Research 1 (1993) 25–46.

[81] W. D. Harvey, Nonsystematic backtracking search, Ph.D. thesis, CIRL, University of Oregon, Eugene, Orgeon (1995).

[82] W. D. Harvey, M. L. Ginsberg, Limited discrepancy search, in: C. S. Mellish (Ed.), Proceedings of IJCAI 1995 – 14th International Joint Conference on Artificial Intelligence, Vol. 1, Morgan Kaufmann Publishers, San Mateo, CA, 1995, pp. 607–615.

[83] M. Milano, A. Roli, On the relation between complete and incomplete search: an informal discussion, in: N. Jussien, F. Laburthe (Eds.), Proceedings of CP-AI-OR'02 – Fourth Int. Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems, 2002, pp. 237–250.

[84] M. Dell'Amico, A. Lodi, On the integration of metaheuristic strategies in constraint programming, in: C. Rego, B. Alidaee (Eds.), Metaheuristic Optimization via Memory and Evolution, Vol. 30 of Operations Research/Computer Science Interfaces Series, Springer Verlag, Berlin, Germany, 2002, pp. 357–371.

[85] N. Jussien, O. Lhomme, Local search with constraint propagation and conflict-based heuristics, Artificial Intelligence 139 (2002) 21–45.

[86] A. Schaerf, Combining local search and look-ahead for scheduling and constraint satisfaction problems, in: M. Pollack (Ed.), Proceedings of IJCAI 1997 – 15th International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers, San Mateo, CA, 1997, pp. 1254–1259.

[87] S. Prestwich, Combining the scalability of local search with the pruning techniques of systematic search, Annals of Operations Research 115 (2002) 51–72.

[88] D. P. Bertsekas, J. N. Tsitsiklis, C. Wu, Rollout algorithms for combinatorial optimization, Journal of Heuristics 3 (1997) 245–262.

[89] T. A. Feo, M. G. C. Resende, Greedy randomized adaptive search procedures, Journal of Global Optimization 6 (1995) 109–133.

[90] P. S. Ow, T. E. Morton, Filtered beam search in scheduling, International Journal of Production Research 26 (1988) 297–307.

[91] C. Blum, C. Cotta, A. J. Fernández, J. E. Gallardo, A probabilistic beam search algorithm for the shortest common supersequence problem, in: C. Cotta, J. I. van Hemert (Eds.), Proceedings of EvoCOP 2007 – Seventh European Conference on Evolutionary Computation in Combinatorial Optimisation, Vol. 4446 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2007, pp. 36–47.

[92] V. Maniezzo, Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem, INFORMS Journal on Computing 11 (4) (1999) 358–369.

[93] V. Maniezzo, A. Carbonaro, An ANTS heuristic for the frequency assignment problem, Future Generation Computer Systems 16 (2000) 927–935.

[94] V. Maniezzo, M. Roffilli, Very strongly constrained problems: an ant colony optimization approach, Cybernetics and Systems 39 (4) (2008) 395–424.

[95] C. Blum, Beam-ACO–hybridizing ant colony optimization with beam search: an application to open shop scheduling, Computers and Operations Research 32 (2005) 1565–1591.

[96] C. Blum, Beam-ACO for simple assembly line balancing, INFORMS Journal on Computing 20 (4) (2008) 618–627.

[97] M. López-Ibáñez, C. Blum, Beam-ACO for the travelling salesman problem with time windows, Computers & Operations Research 37 (9) (2010) 1570–1583.

[98] M. Mastrolilli, C. Blum, On the use of different types of knowledge in metaheuristics based on constructing solutions, Engineering Applications of Artificial Intelligence 23 (5) (2010) 650–659.

[99] G. L. Nemhauser, L. A. Wolsey, Integer and Combinatorial Optimization, John Wiley & Sons, 1988.

[100] L. A. Wolsey, Integer Programming, Wiley-Interscience, 1998.

[101] S. Mitrović-Minić, A. P. Punnen, Variable intensity local search, in: Maniezzo et al. [13], pp. 245–252.

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

[103] M. Fischetti, A. Lodi, Local Branching, Mathematical Programming, Series B 98 (2003) 23–47.

[104] P. Hansen, N. Mladenović, D. Urosević, Variable neighborhood search and local branching, Computers and Operations Research 33 (10) (2006) 3034–3045.

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

[106] C. Archetti, G. Guastaroba, M. G. Speranza, An ILP-refined tabu search for the selective arc routing problem with penalties, in: Dörner et al. [12], pp. 61–82.

[107] R. De Franceschi, M. Fischetti, P. Toth, A new ILP-based refinement heuristic for vehicle routing problems, Mathematical Programming, Series B 105 (2) (2006) 471–499.

[108] T. Oncan, S. N. Kabadi, K. P. N. Nair, A. P. Punnen, VLSN search algorithms for partitioning problems using matching neighbourhoods, The Journal of the Operational Research Society 59 (2008) 388–398.

[109] S. Ropke, D. Pisinger, An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows, Transportation Science 40 (4) (2006) 455–472.

[110] S. Pirkwieser, G. R. Raidl, Variable neighborhood search coupled with ILP-based large neighborhood searches for the (periodic) location-routing problem, in: M. J. Blesa Aguilera, C. Blum, G. R. Raidl, A. Roli, M. Sampels (Eds.), Proceedings of HM 2010 – Seventh International Workshop on Hybrid Metaheuristics, Vol. 6373 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2010, pp. 174–189.

[111] A. Nagar, S. S. Heragu, J. Haddock, A meta-heuristic algorithm for a bi-criteria scheduling problem, Annals of Operations Research 63 (1995) 397–414.

[112] D. L. Applegate, R. E. Bixby, V. Chvátal, W. J. Cook, On the solution of the traveling salesman problem, Documenta Mathematica Extra Volume ICM III (1998) 645–656.

[113] D. L. Applegate, R. E. Bixby, V. Chvátal, W. J. Cook, The Traveling Salesman Problem: A Computational Study, Princeton Series in Applied Mathematics, Princeton University Press, 2007.

[114] C. Cotta, J. M. Troya, Embedding branch and bound within evolutionary algorithms, Applied Intelligence 18 (2003) 137–153.

[115] A. V. Eremeev, On complexity of optimal recombination for binary representations of solutions, Evolutionary Computation 16 (1) (2008) 127–147.

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

[117] E. Danna, E. Rothberg, C. Le Pape, Integrating mixed integer programming and local search: A case study on job-shop scheduling problems, in: Fifth International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'2003), 2003, pp. 65–79.

[118] J. E. Gallardo, C. Cotta, A. J. Fernández, On the hybridization of memetic algorithms with branch-and-bound techniques, IEEE Transactions on Systems, Man, and Cybernetics – Part B 37 (1) (2007) 77–83.

[119] C. Blum, C. Cotta, A. J. Fernández, J. E. Gallardo, M. Mastrolilli, Hybridization of metaheuristics with branch & bound derivatives, in: Blum et al. [14], pp. 85–116.

[120] P. V. Hentenryck, L. Michel, Constraint-Based Local Search, MIT Press, Cambridge, MA, 2005.

[121] L. Shi, S. Ólafsson, Nested partitions method for global optimization, Operations Research 48 (3) (2000) 390–407.

[122] S. Al-Shihabi, S. Ólafsson, A hybrid of nested partition, binary ant system, and linear programming for the multidimensional knapsack problem, Computers & Operations Research 37 (2) (2010) 247–255.

[123] L. Shi, S. Ólafsson, Q. Chen, An optimization framework for product design, Management Science 47 (12) (2001) 1681–1692.

[124] M. Boschetti, V. Maniezzo, Benders decomposition, Lagrangian relaxation and metaheuristic design, Journal of Heuristics 15 (2009) 283–312.

[125] M. Boschetti, V. Maniezzo, M. Roffilli, Decomposition techniques as metaheuristic frameworks, in: Maniezzo et al. [13], pp. 135–158.

[126] J. E. Beasley, Lagrangian relaxation, in: Reeves [1], pp. 243–303.

[127] V. Jeet, E. Kutanoglu, Lagrangian relaxation guided problem space search heuristic for generalized assignment problems, European Journal of Operational Research 182 (3) (2007) 1039–1056.

[128] C. Wilbaut, S. Hanafi, New convergent heuristics for 0-1 mixed integer programming, European Journal of Operational Research 195 (1) (2009) 62–74.

[129] A. L. Soyster, B. Lev, W. Slivka, Zero-one programming with many variables and few constraints, European Journal of Operational Research 2 (3) (1978) 195–201.

[130] S. Hanafi, C. Wilbaut, Improved convergent heuristics for the 0-1 multidimensional knapsack problem, Annals of Operations Research 183 (1) (2011) 125–142.

[131] F. Glover, Inequalities and target objectives for metaheuristic search – part i: Mixed binary optimization, in: P. Siarry, Z. Michalewicz (Eds.), Advances in Metaheuristics for Hard Optimization, Natural Computing Series, Springer Verlag, Berlin, Germany, 2008, pp. 439–474.

[132] G. R. Raidl, H. Feltl, An improved hybrid genetic algorithm for the generalized assignment problem, in: H. M. Haddadd, et al. (Eds.), Proceedings of the 2003 ACM Symposium on Applied Computing, ACM Press, 2004, pp. 990–995.

[133] G. R. Raidl, An improved genetic algorithm for the multiconstrained 0–1 knapsack problem, in: D. B. Fogel, et al. (Eds.), Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, IEEE Press, 1998, pp. 207–211.

[134] P. C. Chu, J. E. Beasley, A genetic algorithm for the multidimensional knapsack problem, Journal of Heuristics 4 (1998) 63–86.

[135] M. Vasquez, J.-K. Hao, A hybrid approach for the 0–1 multidimensional knapsack problem, in: B. Nebel (Ed.), Proceedings of the 17th International Joint Conference on Artificial Intelligence, IJCAI 2001, Morgan Kaufman, Seattle, Washington, 2001, pp. 328–333.

[136] M. Vasquez, Y. Vimont, Improved results on the 0–1 multidimensional knapsack problem, European Journal of Operational Research 165 (1) (2005) 70–81.

[137] M. Haouari, J. C. Siala, A hybrid Lagrangian genetic algorithm for the prize collecting Steiner tree problem, Computers & Operations Research 33 (5) (2006) 1274–1288.

[138] F. Barahona, R. Anbil, The volume algorithm: Producing primal solutions with a subgradient method, Mathematical Programming, Series A 87 (3) (2000) 385–399.

[139] S. Pirkwieser, G. R. Raidl, J. Puchinger, Combining Lagrangian decomposition with an evolutionary algorithm for the knapsack constrained maximum spanning tree problem, in: C. Cotta, J. I. van Hemert (Eds.), Evolutionary Computation in Combinatorial Optimization – EvoCOP 2007, Vol. 4446 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2007, pp. 176–187.

[140] M. Leitner, G. R. Raidl, Lagrangian decomposition, metaheuristics, and hybrid approaches for the design of the last mile in fiber optic networks, in: Blesa Aguilera et al. [8], pp. 158–174.

[141] H. Tamura, A. Hirahara, I. Hatono, M. Umano, An approximate solution method for combinatorial optimisation, Transactions of the Society of Instrument and Control Engineers 130 (1994) 329–336.

[142] M. Reimann, Guiding ACO by problem relaxation: A case study on the symmetric TSP, in: T. Bartz-Beielstein, M. J. Blesa Aguilera, C. Blum, B. Naujoks, A. Roli, G. Rudolph, M. Sampels (Eds.), Proceedings of HM 2007 – Fourth International Workshop on Hybrid Metaheuristics, Vol. 4771 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2007, pp. 45–55.

[143] D. P. Bertsekas, Dynamic Programming and Optimal Control, 3rd Edition, Athena Scientific, Nashua, NH, 2007.

[144] R. K. Congram, C. N. Potts, S. L. van de Velde, An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem, INFORMS Journal on Computing 14 (1) (2002) 52–67.

[145] A. Grosso, F. Della Croce, R. Tadei, An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem, Operations Research Letters 32 (1) (2004) 68–72.

[146] E. Angel, E. Bampis, A multi-start dynasearch algorithm for the time dependent single-machine total weighted tardiness scheduling problem, European Journal of Operational Research 162 (1) (2005) 281–289.

[147] M. Sniedovich, S. Voß, The corridor method: a dynamic programming inspired metaheuristic, Control and Cybernetics 35 (3) (2006) 551–578.

[148] R. E. Bellman (Ed.), Dynamic Programming, Princeton University Press, New Jersey, USA, 1957.

[149] M. Heidari, T. Chow, P. V. Kokotovic, Discrete differential dynamic programming approach to water resources systems optimization, Water Resources Research 7 (2) (1971) 273–282.

[150] R. K. Ahuja, Ö. Ergun, J. B. Orlin, A. P. Punnen, A survey of very large-scale neighborhood search techniques, Discrete Applied Mathematics 123 (1-3) (2002) 75–102.

[151] M. Caserta, S. Voß, M. Sniedovich, Applying the corridor method to a blocks relocation problem, OR SpectrumIn press.

[152] M. Caserta, S. Voß, A math-heuristic algorithm for the dna sequencing problem, in: C. Blum, R. Battiti (Eds.), Proceedings of LION 2010 – 4th International Conference on Learning and Intelligent Optimization, Vol. 6073 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2010, pp. 25–36.

[153] M. Caserta, S. Voß, A corridor method-based algorithm for the premarshalling problem, in: M. Giacobini et al. (Ed.), Proceedings of the EvoWorkshops 2009 – Applications of Evolutionary Computing, Vol. 5484 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2009, pp. 788–797.

[154] C. Blum, M. J. Blesa, Solving the KCT problem: Large-scale neighborhood search and solution merging, in: E. Alba, C. Blum, P. Isasi, C. León, J. A. Gómez (Eds.), Optimization Techniques for Solving Complex Problems, Wiley & Sons, Hoboken, NJ, 2009, pp. 407–421.

[155] B. Hu, G. R. Raidl, Effective neighborhood structures for the generalized traveling salesman problem, in: J. I. van Hemert, C. Cotta (Eds.), Evolutionary Computation in Combinatorial Optimisation – EvoCOP 2008, Vol. 4972 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2008, pp. 36–47.

[156] T. Dunker, G. Radons, E. Westkämper, Combining evolutionary computation and dynamic programming for solving a dynamic facility layout problem, European Journal of Operational Research 165 (1) (2005) 55–69.

[157] J. Balakrishnan, C. H. Cheng, D. G. Conway, C. M. Lau, A hybrid genetic algorithm for the dynamic plant layout problem, Production Economics 86 (2) (2003) 107–120.

[158] C. Wilbaut, S. Hanafi, A. Fréville, S. Balev, Tabu search: global intensification using dynamic programming, Control and Cybernetics 35 (3) (2009) 579–598.

[159] S.-M. Tse, Y. Liang, K.-S. Leung, K.-H. Lee, T. S.-K. Mok, A memetic algorithm for multiple-drug cancer chemotherapy schedule optimization, IEEE Transactions on Systems, Man, and Cybernetics – Part B 37 (1) (2007) 84–91.

[160] W.-S. Juang, S.-F. Su, Multiple sequence alignment using modified dynamic programming and particle swarm optimization, Journal of the Chinese Institute of Engineers 31 (4) (2008) 659–673.

[161] J. Bautista, J. Pereira, A dynamic programming based heuristic for the assembly line balancing problem, European Journal of Operational Research 194 (3) (2009) 787–794.

[162] S. Imahori, M. Yagiura, T. Ibaraki, Improved local search algorithms for the rectangle packing problem with general spatial costs, European Journal of Operational Research 167 (1) (2005) 48–67.

[163] J. N. Hooker, Testing heuristics: We have it all wrong, Journal of Heuristics 1 (1).

[164] C. C. McGeoch, Experimental analysis of algorithms, Notices of the American Mathematical Society 48 (3) (2001) 304–311.

[165] P. R. Cohen, Empirical methods for Artificial Intelligence, The MIT Press, 1995.

[166] D. S. Johnson, A theoretician's guide to the experimental analysis of algorithms, in: D. S. J. M. H. Goldwasser, C. C. McGeoch (Eds.), Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, American Mathematical Society, Providence, RI, 2002, pp. 215–250.

[167] M. Birattari, Tuning Metaheuristics: A machine learning perspective, Vol. 197 of Studies in Computational Intelligence, Springer Verlag, Berlin, Germany, 2009.

[168] C. C. McGeoch, Toward an experimental method for algorithm simulation, INFORMS Journal on Computing 8 (1) (1996) 1–15.

[169] T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (Eds.), Empirical Methods for the Analysis of Optimization Algorithms, Springer Verlag, Berlin, Germany, 2009.