A survey of methods that combine local search and exact algorithms

Irina Dumitrescu Thomas Stützle
Darmstadt University of Technology
Alexanderstr. 10, 64283 Darmstadt, Germany
{irina,tom}@intellektik.informatik.tu-darmstadt.de

Abstract

Exact algorithms like Branch-and-Bound or Dynamic Programming and local search techniques are traditionally seen as being two general but distinct approaches for the effective solution of combinatorial optimisation problems, each having particular advantages and disadvantages. It is only rather recently that true hybrid algorithms, which take ideas from both fields, have been proposed. In this article we present the main underlying ideas behind currently explored such combinations, illustrating them with examples from the literature, and we give a classification of the available combinations. Our main focus here is on algorithms that have the main framework given by the local search and use exact algorithms to solve subproblems.

Keywords: Local Search, Exact Methods, Combined Algorithms

1 Introduction

Many real-life problems, arising in airline scheduling, production planning, location and distribution management, Internet routing, and many other areas are combinatorial optimisation problems (COPs). COPs are intriguing because they are often easy to state but often very difficult to solve, which is captured by the fact that many of them are $\mathcal{NP}$-hard [33]. This difficulty and, at the same time, their enormous practical importance, have led to a large number of solution techniques for them. The available solution techniques can be classified into two main classes: *exact* and *approximate* algorithms. Exact algorithms are guaranteed to find an optimal solution and *prove* its optimality for every finite size instance of a COP within an instance-dependent, finite run-time, or prove that no feasible solution exists. If optimal solutions cannot be computed efficiently in practice, the only possibility is to trade the guarantee of optimality for efficiency. In other words, the guarantee of finding optimal solutions can be sacrificed for the sake of getting very good solutions in reasonably short time by using approximate algorithms.

Two solution method classes that have significant success are Integer Programming (IP) methods, as an exact approach, and local search and extensions thereof, often called metaheuristics or stochastic local search [44], as an approximate approach. IP methods rely on the characteristic of the decision variables of being integers. Some well known IP methods are Branch-and-Bound, Branch-and-Cut, Branch-and-Price, Lagrangean Relaxation, and Dynamic Programming [62]. In recent years, remarkable improvements have been reported for IP methods when applied to some problems (see for example [7] for the TSP). Important advantages of exact methods for IP are that (i) *proven* optimal solutions can be obtained if the algorithm succeeds, (ii) valuable information on the upper/lower bounds on the optimal solution are obtained even if the algorithm is stopped before completion (IP methods can become approximate if we define a criterion of stopping them before solving the problem), and (iii) IP methods allow to provably prune parts of the search space in which optimal solutions cannot be located. A more practical advantage of IP methods is that very good research code like Minto [76], or powerful, general-purpose commercial tools like CPLEX [45], which often reach astonishingly good performance, are available. However, despite the known successes, exact method have also a number of disadvantages. Firstly, for many problems the size of the instances that are practically solvable is rather limited and the computational time increases strongly with the instance size. Secondly, the memory consumption of exact algorithms can also be very large and lead to the early abortion of a programme. Thirdly, for many COPs the best performing algorithms are problem specific and they require large development times by experts in integer programming. Finally, high performing exact algorithms for one problem are often difficult to extend if some details of the problem formulation change. The state-of-the art for IP algorithms is that for some problems huge instances can be solved very fast, while for other types of problems even small size instances cannot be solved at all.

Local search is the most successful class of approximate algorithms. When applied to hard COPs, local search yields high-quality solutions by iteratively applying small modifications (local moves) to a solution in the hope of finding a better one. Embedded into higher-level guidance mechanisms like metaheuristics, this approach has been shown to be very successful in achieving near-optimal (and sometimes optimal) solutions to a number of difficult problems [1, 44, 82]. Examples of well-known metaheuristics are simulated annealing, tabu search, memetic algorithms, or iterated local search [37]. Advantages of local search methods are that they (i) are the best performing algorithms that are used in practice for a large number of problems, (ii) can examine an enormous amount of possible solutions in short computation time, (iii) are often more easily adapted to slight variants of problems and, thus, are more flexible, and (iv) are typically easier to understand and implement than exact methods. However, local search based methods have several disadvantages. Firstly, they cannot prove optimality. Secondly, they typically cannot *provably* reduce the search space. Thirdly, they do not have well defined stopping criteria (this is particularly true for metaheuristics). Finally, local search methods often have problems with highly constrained problems where feasible areas of the solution space are disconnected. Another problem that occurs in practice is that there are no efficient general-purpose local search solvers available. Hence, most applications of local search algorithms often require considerable programming efforts, although often less than for special purpose exact algorithms.

It should be clear by now that IP and local search approaches have their particular advantages and disadvantages and can be seen as complementary. Therefore, it appears to be an obvious idea to try to combine these two distinct techniques into more powerful algorithms.

However, most articles present rather trivial combinations of local search and integer programming techniques. These combinations consist of a straightforward two phase approach where in a first stage a method from one class is used and in a second stage a method from the other class is run. In the case of a minimisation problem, the simplest example is the use of a local search algorithm to compute lower bounds that allow an early pruning of non-optimal solutions. Intuitively however, one would expect more advanced combinations of exact and local search methods to result in even stronger algorithms.

Therefore, in this article we will focus on approaches that strive for an integration of IP and local search techniques. Examples of methods that are *not* the focus of this article are preprocessing and bound propagation.

In this paper we will describe ways of integrating local search and IP methods (or methods derived from an IP approach) that have been proposed so far. We will be looking for approaches that, although rooted in one of the areas, take a significant portion of the other one. In fact this goal is also reflected in the selection of the research directions that are described; we will describe in some detail methods that, in our opinion, open new directions of research and put forward ideas that can easily be applied to new problems. The link between the approaches we consider in this paper is that, given an integer programming problem, a local search technique is applied to the problem to solve and an exact algorithm to some subproblems that arise during the solution process. At least one of the local search or exact methods is applied more than once.

We classify the available combined algorithms in five groups. These groups contain methods that

1. use exact algorithms to explore large neighbourhoods in local search algorithms;
2. perform several runs of a local search and exploit information in high quality solutions to define smaller problems that are amenable for solution with exact algorithms;
3. exploit lower bounds in constructive heuristics;
4. use information from relaxations of IP problems to guide local search or constructive algorithms;
5. use exact algorithms for specific procedures in hybrid metaheuristics.

For each of these groups, we will describe the general framework and give one or several detailed examples. We will then conclude each section with a short discussion and refer to other algorithms that belong to the group or use the same general idea. We finish the paper with a summary of the main points discussed and some conclusions.

Before proceeding with our discussion we need to clarify the notion of *solution component* and *decision variable* that we will use in our paper. We explain what we mean by using the example of the symmetric Traveling Salesman Problem (TSP). The TSP can be defined on an undirected graph $G = (V, E)$, where V is the set of nodes and E the set of edges. An edge represents a pair of nodes between which travel is possible. Every edge $\{i, j\}$ has an associated cost d_{ij} and the goal in the TSP is to find a Hamiltonian cicrcuit of minimal length.

When applying local search to the TSP, a solution component usually refers to a single edge in a tour; each tour contains exactly n such solution components. More generally, a solution component can be seen as an atomic item and a set of such atomic items defines a problem solution. The analogue to a solution component in IP formulations is the notion of decision variable. For example, in the usual IP formulations of the TSP (see [62] for an overview), a variable $x_{ij} \in \{0, 1\}$ is associated with each edge $\{i, j\}$. The variable x_{ij} is equal to 1, if the edge $\{i, j\}$ is included in a tour and zero otherwise. The x_{ij} variables are called decision variables. In the following sections, if we say that a decision variable is fixed to one (zero), we mean that we consider only those solutions in which this variable is set to one (zero); analogously, in the local search case we say that a solution component is forced to always occur (not occur) in any solution considered. In the case of the TSP, this corresponds to an edge always (never) being used in any of the solutions considered. If a variable is free, this means that it can take any value in $\{0, 1\}$.

2 Exploring the neighbourhood

Given a current solution S, local search explores the neighbourhood $\mathcal{N}(S)$ of S iteratively and tries to replace S by a solution $S' \in \mathcal{N}(S)$ according to some criterion. The first combination of local search and exact algorithms that we consider uses exact algorithms for finding a best improving neighbouring solution S' efficiently. Such a combination has proved successful, especially in cases where the number of neighbouring solutions can be exponentially large with respect to the instance size.

2.1 Framework

In local search, it is appealing to search large neighbourhoods because much better solutions can be reached in one local search step than when simple, small neighbourhoods are used. However, large neighbourhoods have the associated disadvantage that a considerable amount of time may be spent to search them in order to find an improved or the best neighbouring solution. More than that, many of the large neighbourhoods proposed in the literature are exponentially large with respect to instance size [36, 49, 67, 80, 81]. Hence, the bottleneck for local search in large neighbourhoods is the task of searching for a better or the best solution in the neighbourhood.

The central idea of algorithms that combine local search and exact methods in this case is to model the problem of searching a large neighbourhood as an optimisation problem, which is then solved exactly. The solution of this problem will determine the neighbour that will replace the current solution in the local search.

For this task, two main possibilities can be considered. The first one is to model the exploration of the *full* neighbourhood as an optimisation problem. In this case, a search problem is defined such that each feasible solution of it induces a move of the local search algorithm. Clearly, the task of the exact algorithm in this case will be to solve the resulting *neighbourhood search problem* (NSP). A general algorithmic outline of this combination method can be given as follows:

Algorithm 2.1 *Neighbourhood search*

```
Step 1: Initialisation
            Let $S$ be a feasible initial solution.
Step 2: Local search
            while stopping_criterion is not met do
                Define a search problem $\mathcal{P}(S)$ that depends on $S$.
                if $\mathcal{P}(S)$ is infeasible then go to Step 3.
                else Find $opt(\mathcal{P}(S))$, the optimal solution of $\mathcal{P}(S)$.
                Perform the move induced by $opt(\mathcal{P}(S))$. Let $S'$ be the solution obtained.
                if $S'$ is better than $S$ w.r.t. the objective function then $S = S'$.
            enddo
Step 3: Return $S$.
```

The first two examples that we present, Very Large Scale Neighbourhood Search and Dynasearch, fall in this class of NSP hybrids.

The second possibility is that at each step of the local search only *a part* of the full neighbourhood is examined. This is typically done by keeping a part of the current solution S, which defines a partial solution, while the values of the rest of the decision variables are left free. The free part is then rearranged optimally, subject to the constraint that the fixed part cannot be altered. In this sense, the task of the exact algorithm is to solve the *partial neighbourhood search problem* (PNSP). The following algorithmic outline gives a high level view of such a procedure, in the case of an iterative improvement algorithm that only accepts better neighbouring solutions and, hence, stops in the first locally optimal solution encountered.

Algorithm 2.2 *Partial neighbourhood search*

```
Step 1: Initialisation
            Let $S$ be an initial solution.
Step 2: Neighbourhood search
            while improvement found do
Step 3: Neighbourhood scan
                while not full neighbourhood examined do
                    Delete part of the solution $S$ such that a partial solution is obtained: $S_p = S \setminus R$.
                    Define a search problem $\mathcal{P}(R)$.
                    Find $opt(\mathcal{P}(R))$, the optimal solution of $\mathcal{P}(R)$.
                    Perform the move induced by $opt(\mathcal{P}(R))$. Let $\bar{S}'$ be the solution obtained.
                    $S' = S_p \cup \bar{S}'$.
```

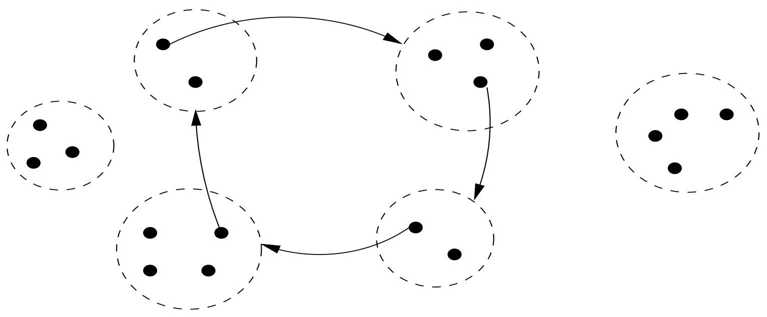
Figure 1: Example of a cyclic exchange for a partitioning problem.

```
        if $S'$ is better than $S$ w.r.t. the objective function then $S = S'$.
    enddo
enddo
Step 4: Return $S$.
```

We illustrate the partial neighbourhood search principle on the Hyperopt local search algorithm; in fact, Hyperopt is only one recent example of such methods, which can be traced back to Applegate and Cook's shuffle heuristic [10].

2.2 NSP Example: Very Large Scale Neighbourhood Search Methods

One application of NSP is the Very Large Scale Neighbourhood (VLSN) search that was introduced by Ahuja et al. [5, 6] in the context of solving partitioning problems, a large and difficult class of problems. Examples of partitioning problems are Vehicle Routing, Capacitated Minimum Spanning Tree, Generalised Assignment, Parallel Machine Scheduling, Clustering, Aggregation, and Graph Partitioning Problems. Next, we will briefly describe the type of problems to which VLSN search is applied and how the neighbourhood search problem is defined.

As mentioned before, VLSN search applies to partitioning problems [6]. Given a set W of n elements, a partition $\mathcal{T}$ of W is sought, where $\mathcal{T}$ is defined to be a set of subsets of W, $\mathcal{T} = \{T_1, \ldots, T_K\}$ such that $W = T_1 \cup \cdots \cup T_K$ and $T_k \cap T_{k'} = \emptyset$, $k, k' = 1, \ldots, K$. A cost $c_k(T_k)$ is associated with any set T_k. The partitioning problem seeks the partition of W that minimises the sum of the costs of the subsets in the partition. Formally, the partitioning problem can be defined as:

$$\min c(\mathcal{T}) = \sum_{k=1}^{K} c_k(T_k)$$
$$\text{s.t. } \mathcal{T} \text{ is a partition of } W.$$

The characteristics of the cost function are not important for the time being; the only property of it that needs to be taken into consideration is its separability over subsets.

Frequently, partitioning problems are solved using local search algorithms that are in most cases based on the two exchange neighbourhood. Cyclic exchange neighbourhoods are a generalisation of the two exchange neighbourhood [6, 80, 81] instead of swapping only two elements from two subsets, several elements, each belonging to a different subset, are moved (see Figure 1).

Formally, a cyclic exchange between k subsets (without loss of generality we can assume them to be $T_1, \ldots, T_k$ and the elements we look at to be $a_1, \ldots, a_k$, with $a_i \in T_i$) is represented by a cyclic permutation π of length k, $\pi \neq \mathbf{1}$, where $\pi(i) = j$ means that the element a_i from subset T_i moves into subset T_j. We recall that a cyclic permutation is a permutation that has a decomposition which consists of one single cycle. A partition $\mathcal{T}'$ is said to be a *neighbour* of the partition $\mathcal{T}$ if it is obtained from $\mathcal{T}$ by performing a cyclic exchange and feasible with respect to some problem specific constraints. The set of all neighbours of $\mathcal{T}$ defines the *cyclic exchange neighbourhood* of $\mathcal{T}$. The cyclic exchange modifies the sets of the partition and therefore their cost. The cost difference for each subset will be the difference between the cost of the subset before performing the cyclic exchange and the cost of the subset after the exchange. The *cost of the cyclic exchange* is the sum of all the cost differences over all subsets in the partition.

In order to find the next move, Ahuja et al. define the *improvement graph*. The construction of the improvement graph depends on the current feasible partition of the partitioning problem considered. The set of nodes V of the improvement graph is defined as the collection of integers $1, \ldots, n$, each corresponding to an element of the partition of W. The improvement graph contains the arc (i, j) if the elements corresponding to i and j do not belong to the same subset in $\mathcal{T}$ and the subset to which the element corresponding to j belongs remains feasible after the removal of the element corresponding to j and the addition of the element corresponding to i. We define the partition $\mathcal{U}$ of V to be the collection of subsets $U_1, \ldots, U_K$ corresponding to the subsets in $\mathcal{T}$, i.e. the elements of T_k are in one-to-one correspondence with the elements of U_k for each $k = 1, \ldots, K$. Therefore a subset disjoint cycle in the improvement graph, with respect to $\mathcal{U}$, will correspond to a cyclic exchange in $\mathcal{T}$. The arc (i, j) will have an associated cost, equal to the difference between the cost of the set after the removal of the element corresponding to j and the addition of the element corresponding to i, and the cost of the original set that contains the element corresponding to j. Thompson and Orlin [80] showed that there is a one-to-one correspondence between the cyclic exchanges with respect to $\mathcal{T}$ and the subset-disjoint cycles in the improvement graph (with respect to $\mathcal{U}$) and that both have the same cost.

The problem of finding the best neighbour within the cyclic exchange neighbourhood can be modelled as a new problem, the Subset Disjoint Minimum Cost Cycle Problem (SDMCCP), or in some cases the Subset Disjoint Negative Cost Cycle Problem (SDNCCP). The SDMCCP is the problem of finding the minimum cost subset disjoint cycle (a cycle that uses at most one node from every subset of the partition of the set of nodes) in the improvement graph. The SDNCCP is the problem of finding a minimum subset disjoint cycle with the additional constraint that its cost is negative. The only real difference between SDMCCP and SDNCCP is that the feasible set of the latter is a subset of the feasible set of the former. Also note that if the network contains negative cycles, then the set of optimal solutions of both problems is the same.

If SDNCCP has a solution, then this solution corresponds to an improvement in the solution quality of the partitioning problem considered. Hence, the SDNCCP is the problem to be solved when using VLSN in an iterative improvement algorithm. The SDMCCP is important if the VLSN search technique is embedded, for example, into a tabu search procedure; then finding the best neighbour may be important regardless of whether or not this neighbour is better than the current solution. Ahuja et al. [3] solve the SDNCCP by a heuristic that takes advantage of the fact that only negative cycles are needed. However they make no attempt to solve the SDNCCP exactly. Several exact methods for both SDMCCP and SDNCCP are proposed by Dumitrescu [29]. They can be viewed as generalisations of dynamic programming algorithms commonly used for shortest path problems. In the case of the SDMCCP an acceleration method that takes advantage of symmetry is given, and in the case of SDNCCP the author takes advantage of an elegant theorem of Lin and Kernighan [50], which, as noted by Ahuja et al. [5], can be used to accelerate solution. Although the SSDNCCP and SDMCCP are $\mathcal{NP}$-hard, the proposed methods are very efficient for the subproblems that arise in practical applications of VLSN search methods. Dumitrescu also proposes some heuristics that are derived from the exact methods by limiting or truncating them in some way.

However, although research exists on both applying VLSN search to partitioning problems and on solving the SDMCCP and SDNCCP that arise as subproblems, no work has yet been done on the integration of the *exact* algorithms that solve the subproblems with the local search framework. This needs to be investigated and implemented in the future.

2.3 NSP Example: Dynasearch

Dynasearch is another example of how an exact algorithm can be used to search an exponentially large neighbourhood. The neighbourhood searched consists of all possible combinations of mutually independent simple search moves. One dynasearch move consists of a set of independent simple search moves that are executed in parallel in a single local search iteration. Independence in the context of dynasearch means that the individual simple search moves do not interfere with each other, especially with respect to the objective function; this means that the gain incurred by a dynasearch move must be the sum of the gains of the individual simple search moves.

So far, Dynasearch has only been applied to problems where solutions are represented as permutations. Independence between simple search moves is given if the last element involved in one move occurs before the first element of the next move. This means, that if $\pi = (\pi(1), \ldots, \pi(n))$ is the current permutation, two moves involving elements from $\pi(i)$ to $\pi(j)$ and $\pi(k)$ to $\pi(l)$, with $1 \leq i < j \leq n$ and $1 \leq k < l \leq n$ are independent, if either $j < k$ or $l < i$. If overlapping moves are not allowed, the best combination of independent moves can be found by a straightforward dynamic programming algorithm.

Let $\Delta(j)$ be the maximum total cost reduction incurred by independent moves involving only elements from position 1 to j of the current permutation and $\delta(i, j)$ be the cost reduction resulting from a move involving positions between i and j, including i and j. The maximum total cost reduction is obtained either by appending $\pi(j)$ to the current partial permutation or by appending element $\pi(j)$ and applying a move involving element $\pi(j)$ (and possibly elements from $\pi(i)$ onwards).

We assume that the current solution is given by π and set $\Delta(0) = 0$ and $\Delta(1) = 0$. Then, $\Delta(j),\ j = 1, \ldots, n-1$ can, in general, be computed in a forward evaluation using the following, recursive formula:

$$\Delta(j+1) = \max_{1 \leq i \leq j} \{\Delta(i-1) + \delta(i, j+1)\}\} \tag{1}$$

The largest reduction in solution cost is then given by $\Delta(n)$ and the single moves to be performed can be found by tracing back the computation steps. In order to apply dynasearch using the forward evaluation, the value of $\Delta(j)$ should not depend on positions $k > j$. If the evaluation of moves depends only on elements $k > j$, a backward version of the dynamic programming algorithm can be applied. This version that starts with $\pi(n)$ and then generates the sequence of $\Delta(j),\ j = n, n-1, \ldots, 1$.

When applying the algorithm, the particularities of the moves have to be taken into account when using (1). Consider, for example, the application of dynasearch to the TSP using 2–exchange moves as the underlying simple moves. Here we assume that a tour is represented as $\pi = (\pi(1), \ldots, \pi(n+1))$, where we define $\pi(n+1) = \pi(1)$. If $1 < i+1 < j \leq n$ and $\pi(j+1) \neq \pi(i)$, a 2–exchange move is obtained by removing edges $(\pi(i), \pi(i+1))$ and $(\pi(j), \pi(j+1))$ from the current tour π and introducing edges $(\pi(i), \pi(j))$ and $(\pi(i+1), \pi(j+1))$. The move is accepted if the new tour is shorter, i.e. if and only if

$$d(\pi(i), \pi(i+1)) + d(\pi(j), \pi(j+1)) > d(\pi(i), \pi(j)) + d(\pi(i+1), \pi(j+1)),$$

where $d(\dot{,})$ is the cost function defined on the set of edges.

The dynasearch 2–exchange algorithm presented by Congram [24] tries to improve a Hamiltonian path between two fixed end points $\pi(1)$ and $\pi(n)$. In this case, two 2–exchange moves that delete edges $(\pi(i), \pi(i+1))$ and $(\pi(h), \pi(h+1))$, where $1 < i+1 < h \leq n$, $\pi(j+1) \neq \pi(i)$, and $(\pi(k), \pi(k+1))$ and $(\pi(l), \pi(l+1))$, where $1 < k+1 < l \leq n$, $\pi(l+1) \neq \pi(k)$, are independent if we have $h \leq k$ or $l < i$, because in this case the segments that are rearranged by the two moves do not have any edges in common. An example of such an independent dynasearch move is given in Figure 2.

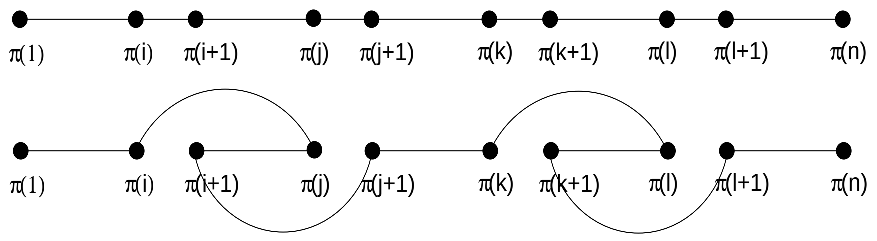

Figure 2: Example of a dynasearch move that is composed of two independent 2–exchange moves.

In this case, $\Delta(j)$ is the largest reduction of the length of the path between the endpoints $\pi(1)$ and $\pi(j)$ obtained by independent 2–exchange moves. The initialisation of the dynamic program is $\Delta(1) = \Delta(2) = \Delta(3) = 0$, because we need at least four nodes to apply a 2–exchange move. The recursion formula then becomes

$$\Delta(j+1) = \max_{1 \leq i \leq j-2} \{\Delta(j) + \delta(i, j+1)\}$$

$\Delta(n+1)$ gives the maximal improvement.

Current applications of dynasearch comprise the travelling salesman problem [24], the single machine total weighted tardiness problem (SMTWTP) [24, 25], and the linear ordering problem (LOP) [24]. A general observation is that dynasearch, on average, is faster than a standard best-improvement descent algorithm and returns slightly better quality solutions. Particularly good performance is reported, if dynasearch is used as a local search routine inside an iterated local search [53] or guided local search algorithm [85]. Currently, iterated dynasearch is the best performing metaheuristic for the SMTWTP and very good results were also obtained for the TSP and the LOP [24].

2.4 PNSP Example: Hyperopt Neighbourhoods

As mentioned before, hyperopt is an example of a hybrid algorithm of the PNSP type. Here, we present the application of hyperopt to the TSP. This application was examined by Burke et al. for the symmetric and the asymmetric TSP [19, 20]. To keep the presentation as simple as possible, we will only consider the application to the TSP.

The *hyperopt neighbourhood* is based on the notion of hyperedges. Given a tour of the TSP, a *hyperedge* is defined to be a subpath of the tour; in other words, a sequence of successive edges of the tour [19]. If i is the start node and j the end node of the hyperedge, we denote the hyperedge by $\mathcal{H}(i, j)$. The *length* of a hyperedge is given by the number of edges in it. Let t be a feasible tour of the TSP and $\mathcal{H}(i_1, i_{k+1})$ and $\mathcal{H}(j_1, j_{k+1})$ two hyperedges of length k such that $\mathcal{H}(i_1, i_{k+1}) \cap \mathcal{H}(j_1, j_{k+1}) = \emptyset$ with respect to the nodes contained. We assume that the tour t can be written $t = (i_1, \ldots, i_{k+1}, \ldots, j_1, \ldots, j_{k+1}, \ldots, i_1)$. It is obvious that the tour t can be described completely by four hyperedges: $\mathcal{H}(i_1, i_{k+1})$, $\mathcal{H}(i_{k+1}, j_1)$, $\mathcal{H}(j_1, j_{k+1})$, and $\mathcal{H}(j_{k+1}, i_1)$, as shown in Figure 3(a). Burke et al. define a *k-hyperopt move* as a composition of the two following steps: remove $\mathcal{H}(i_1, i_{k+1})$ and $\mathcal{H}(j_1, j_{k+1})$ from the tour t, then add edges to $\mathcal{H}(i_{k+1}, j_1)$ and $\mathcal{H}(j_{k+1}, i_1)$ such that a new feasible tour is constructed (see Figure 3 for an example). The *k-hyperopt neighbourhood* consists of all k-hyperopt moves.

The size of the k-hyperopt neighbourhood increases exponentially with k. Burke et al. propose an “optimal” construction of a k-hyperopt move by solving exactly a subproblem: the TSP defined on the graph $G' = (V', E')$, where V' is the set of nodes included in $\mathcal{H}(i_1, i_{k+1})$ and $\mathcal{H}(j_1, j_{k+1})$ and E' is the set of edges in the original TSP that have both ends in V'. However,

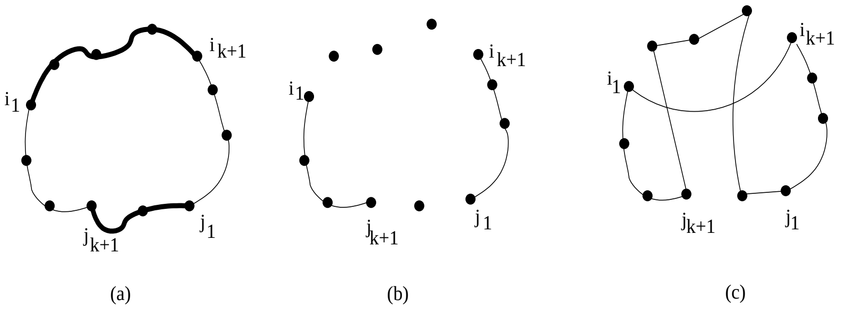

Figure 3: (a) A feasible tour can be seen as the union of $\mathcal{H}(i_1, i_{k+1})$, $\mathcal{H}(i_{k+1}, j_1)$, $\mathcal{H}(j_1, j_{k+1})$, and $\mathcal{H}(j_{k+1}, i_1)$. (b) The hyperedges $\mathcal{H}(i_1, i_{k+1})$ and $\mathcal{H}(j_1, j_{k+1})$ are removed. (c) A new feasible tour is constructed.

this approach is bound to be efficient only when k is relatively small. Otherwise, a large size TSP would have to be solved as a subproblem.

Following this observation, Burke et al. consider only the cases when $k = 2$ and $k = 3$ in the case of the TSP and do not use any algorithm to solve the subproblems [19]. They obtain the optimal solution of the subproblems simply by enumeration. An interesting direction for future research can be the investigation of considering larger values of k combined with the use of more sophisticated tools for obtaining the optimal solution of the subproblems generated. In the case of the asymmetric TSP numerical results are given for $k = 3$ and $k = 4$ [20]; a dynamic programming algorithm was used to explore the neighbourhood, but no details of the algorithm are given.

For a given hyperedge $\mathcal{H}(i_1, i_{k+1})$ the k-hyperopt move to be performed is determined in [19, 20] by evaluating the best k-hyperopt move over the set of *all* possible hyperopt moves. Therefore a sequence of subproblems is generated and solved, one subproblem for every hyperedge that does not intersect with $\mathcal{H}(i_1, i_{k+1})$. It is not clear how $\mathcal{H}(i_1, i_{k+1})$ is chosen in the first place or if that hyperedge also changes. This is clearly an expensive approach from the point of view of the computational time. To speed up computation Burke et al. use "don't look bits" [15] that ban some hyperedges from being considered.

As a further extension, the use of k-hyperopt moves inside a variable neighbourhood search procedure [39, 41] was proposed. However, the currently available results suggest that the resulting hyperopt local search for the ATSP is somewhat inferior to current state-of-the-art local search techniques [46]. In spite of this, we believe that the k-hyperopt approach is a promising research direction, which deserves to be further investigated. A clear first research direction would be to enlarge the neighbourhood searches by increasing the value of k, while using truly efficient algorithms to solve the resulting subproblems.

2.5 Other approaches

The two NSP techniques for exploring a neighbourhood that we described in this section have been applied to a variety of problems; for an overview of VLSN search applications we refer to a webpage maintained by Ahuja and Orlin [4]. Another example of such a method is the constraint programming approach by Pesant and Gendreau [65, 66], where the exploration of the neighbourhood is modelled as a problem that is then solved with constraint programming techniques [57, 83]. Constraint programming is an especially promising approach, if the resulting search problems are tightly constrained.

PNSP algorithms have been more widely applied. As mentioned before, one of the first PNSP algorithms is the shuffle heuristic of Applegate and Cook [10], which was applied to the job-shop scheduling problem (see Section 6.2 for a description of this problem). Briefly, their approach consists of defining a partial solution by leaving the sequence fixed for one or a few machines. The subproblem of completing the schedule is then solved by branch-and-bound. Another example is the MIMAUSA algorithm, introduced by Mautor and Michelon [59, 60, 58], applied to the quadratic assignment problem (see Section 4.2 for more information on this problem). In MIMAUSA, at each step k variables are ”freed”. The subproblem of assigning values to the free variables, subject to the constraint that the rest of the solution is kept fixed, is solved to optimality. A similar approach has been applied by Büdenbender, Grünert, and Sebastian to the direct flight network design problem [18]. PNSP methods also have been successfully applied in combination with constraint programming techniques, with some noteworthy examples being the applications of Caseau and Laburthe [23] for the job-shop scheduling problem or that of Shaw [77] for the vehicle routing problem, where a Branch-and-Bound algorithm was integrated into tabu search. For a detailed overview of combinations of local search and constraint programming we refer to the recent overview articles by Focacci, Laburthe, and Lodi [32].

2.6 Discussion

In general, the use of exact algorithms for exploring large neighbourhoods is appealing. For some problems the resulting neighbourhoods can be explored fully by polynomial time algorithms, the most noteworthy example being here the dynasearch approach. If the large neighbourhoods cannot be explored in *provably* polynomial time, often the resulting neighbourhood search problems, whether corresponding to full or partial neighbourhood, can be solved efficiently by exact algorithms. This is mainly due to the exact algorithms being, in most cases, rather quick if the problem size is not too large and to the fact that the resulting subproblems often have a special structure that can be exploited efficiently.

Despite the usefulness of exact algorithms in these methods, it is clear that the resulting subproblems could also be solved by approximate algorithms. In fact, a general framework for this idea of defining subproblems and exploring them heuristically, POPMUSIC, was defined in a paper of Taillard and Voss [78]. Similar ideas were also put forward in variable neighbourhood decomposition search [42]. In fact, it would be interesting to carefully examine the use of local search algorithms or heuristics versus the use of exact algorithms for exploring the large neighbourhoods; however, to the best of our knowledge, no study on this issue has been conducted so far.

3 Exploiting the structure of good solutions

Many optimisation problems show some type of structure in the sense that high quality solutions have a large number of solution components in common with optimal solutions. This observation can be exploited in several ways by defining appropriate subproblems of the original problem by using the information obtained from high quality solutions. Often the resulting subproblems are small enough to be solved by exact algorithms.

3.1 Framework

This approach consists of two phases. In the first phase, an approximate algorithm is used repeatedly to collect a number of solutions of the problem under consideration. Based on the solution components that are included in the set of solutions, a subproblem of the original problem is defined. It is expected that the subproblem still contains all or, if not all, most of the ”important” decision variables and that the subproblem can be solved relatively easily by an exact algorithm. The optimal solution for the subproblem is then an upper bound for the optimal solution of the original problem.

An algorithmic outline of the type of methods falling into this framework is given next.

Algorithm 3.1 *Exploiting structure by collecting information*

```
Step 1: Initialisation
          Let $\mathcal{I} = \emptyset$, where $\mathcal{I}$ is the set of collected solutions.
Step 2: Approximate algorithm
          while stopping_criterion is not met do
              Let $S$ be the solution returned after running an approximate algorithm
              Add $S$ to $\mathcal{I}$.
          enddo
          Reduce $\mathcal{I}$ according to some criteria (optional).
Step 3: Optimisation
              Define a subproblem $\mathcal{P}(\mathcal{I})$ that depends on $\mathcal{I}$.
              Find $opt(\mathcal{P}(\mathcal{I}))$, the optimal solution of $\mathcal{P}(\mathcal{I})$.
Step 4: Return $opt(\mathcal{P}(\mathcal{I}))$.
```

Next we give two well known examples where this idea was successfully applied to obtain optimal or close to optimal solutions for the TSP and for the p-median problem.

3.2 Example: Tour Merging

Applegate, Bixby, Chvàtal, and Cook proposed the tour merging approach [9], which was subsequently studied in more detail by Cook and Seymour [26]. Tour merging is a two phase procedure following the algorithm outlined above.

The first phase consists of generating a number of high quality tours for the TSP instance that needs to be solved. In order to do that, Applegate et al [9] used the chained Lin-Kernighan algorithm, an effective iterated local search algorithm [53] for the TSP. However, it is clear that any other algorithm for generating high quality solutions for the TSP can be used, for example the iterative application of Helsgaun's effective Lin-Kernighan implementation [43]. This was the heuristic applied in [26]. In what follows we will denote the set of tours obtained during the first stage by $\mathcal{I}$.

The second phase consists of solving the TSP on the graph induced by the set of tours $\mathcal{I}$ on the original graph. In other words, a TSP is solved on a restricted graph that has the same set of nodes as the original graph, but its set of edges consists of the edges that appear at least once in any of the tours in $\mathcal{I}$. Formally, if the original graph is $G = (V, A)$, where V is the set of nodes and A the set of arcs, the reduced graph can be described as $G' = (V, A')$, where $A' = \{a \in A : \exists t \in \mathcal{I}, a \in t\}$.

Several possibilities of solving the TSP on the reduced graph G' can be considered. In [9], a general purpose exact algorithm for the TSP was used: the Concorde code available online [8]. When Concorde was applied to medium sized instances with up to 5000 cities, the tour merging method could identify optimal solutions for all instances at least twice out of ten independent trials. However, on some few instances the computation time was large, with most of it being taken by the optimisation code applied in the second stage. Additional experiments investigated how the quality and the number of tours in $\mathcal{I}$ affected the solution quality and the computation time required in the second stage. Another possibility of solving the TSP in G' is to use special purpose algorithms that exploit the structure of the graph G'. In particular, G' is a very sparse graph and it is likely to have a low *branch-width* [70]. In [26], a dynamic programming algorithm that exploits this property is presented and computational results on a wide range of large TSP instances (2,000 up to about 20,000 cities) show excellent performance, resulting in computation times for the second stage that are much shorter than times required by a general purpose TSP solver. In fact, the tour merging approach, especially in the way it is applied by Cook and Seymour [26], is currently one of the best performing approximate algorithms for medium size to large TSP instances.

3.3 Example: Heuristic Concentration

A similar idea is used by Rosing and ReVelle for the solution of the p-median problem [72]. Given a graph $G = (V, A)$, where $V = \{1, \ldots, n\}$, each node having an associated weight, each arc having an associated distance, and given a positive integer p, the p-median problem consists of finding p nodes in the graph, called *facilities*, such that the sum of the weighted distances between every node and its closest facility is minimised. The nodes that are not facilities are called *demand nodes*. This problem can be modelled as a binary integer problem (BIP) [69] as follows:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} w_i d_{ij} x_{ij}$$

$$\text{s.t.} \sum_{j=1}^{n} x_{ij} = 1, \forall i \tag{2}$$

$$x_{jj} - x_{ij} \geq 0, \forall i, j, i \neq j \tag{3}$$

$$\sum_{j=1}^{n} x_{jj} = p \tag{4}$$

$$x \in \{0, 1\}^n, \tag{5}$$

where w_i is the weight associated with demand node i, d_{ij} the distance from node i to node j, i the index of demand nodes, and j the index of facility sites. The decision variables are $x_{ij} = 1$, if demand node i is associated to facility j, and $x_{ij} = 0$ otherwise. An important characteristic of this model is that in most cases (over 95%) the optimal solution of its linear programming relaxation is integer [61]. When the optimal solution is fractional, an integer solution can be found quickly using branch-and-bound [74]. However, since the difficulty of any LP approach increases with the size of the problem, other solution methods are needed for large values of n.

The technique developed by Rosing and ReVelle is called *heuristic concentration*. In the first phase, a local search procedure, in this case the Teitz and Bart heuristic [79], is run repeatedly with different random starts. Every solution obtained in this process is recorded in a set $\mathcal{I}$. The number of times the heuristic is run is determined after conducting numerical experiments.

The second phase starts with the selection of a subset $\mathcal{I}'$ of the solutions in $\mathcal{I}$, that is $\mathcal{I}' \subset \mathcal{I}$. $\mathcal{I}'$ contains only the best solutions of $\mathcal{I}$ with respect to the objective function; the number of solutions in $\mathcal{I}'$ is again determined by numerical experiments. The facility locations used in these "best" solutions form a subset of the set of all nodes and they will be collected in what the authors call the *concentration set* (CS). Finally, a restricted p-median problem, with the facility locations restricted to those contained in the concentration set, is solved. The integer model of the restricted problem is similar to the original model; the only difference is that i can only take values from CS, i.e. $i \in CS$.

Clearly, this method is built on the assumption that a near-optimal or optimal solution must have the facilities located at sites from CS. However, this is only an assumption based on experimental results. An even stronger assumption is made by the authors in order to further reduce the dimension of the subproblem being solved. This is the claim that the nodes that are facilities in *all* solutions considered *must* be facilities. The rest of the CS contains *possible* locations. Therefore, they split CS into two subsets: one contains the locations where there are facilities in all solutions, called *CS open* (CS_0), and the other one contains the nodes where there is a facility in at least one but not all solutions. The latter subset of CS is called *CS free* (CS_f). The authors also exploit the fact that each demand node not in CS must be assigned to the closest facility; for every node, variables are needed only for that node in CS_0 that is closest to the demand node, and only for those nodes in CS_f that are even closer than that member of the CS_0. The binary integer model of this restricted p-median problem can be written as follows:

$$\begin{aligned}
\min\ & \sum_{i=1}^{n} \sum_{j \in R_i} w_i d_{ij} x_{ij} \\
\text{s.t.}\ & \sum_{j \in R_i} x_{ij} = 1, \forall i \notin CS_0 && (6) \\
& x_{jj} = 1, \forall j \in CS_0 && (7) \\
& x_{jj} - x_{ij} \geq 0, \forall i, \forall j \in R_i, i \neq j && (8) \\
& \sum_{j \in CS} x_{jj} = p && (9) \\
& x_{jj} \in \{0,1\}, \forall j \in CS_f, && (10)
\end{aligned}$$

where for all $i \notin CS_0$ we define $M_i = \{j : d_{ij} = \min_{k \in CS_0} d_{ik}\}$ and $R_i = M_i \cup \{j : d_{ij} < d_{ik}, j \in CS_f, k \in M_i\}$.

It is clear that when CS can be split, the size of the second subproblem is smaller than the size of the first one. However, there is no guarantee that such a partition of CS exists. When $CS_0 = \emptyset$, the only possibility is to use the first subproblem. Since in most cases the solution of the LP relaxation of the binary integer model of a p-median problem is integer, Rosing and ReVelle solve the LP relaxation of the restricted problem using an LP solver (in this case CPLEX). Even when the solution is fractional, an integer solution is found very easily. Numerical results are provided in both [72] and subsequent papers [71, 73]. They prove that the heuristic concentration is a viable technique that obtains very good results in practice.

3.4 Discussion

The two examples we presented in this section show that approaches based on collected information can result in highly efficient hybrid algorithms in practice. Intuitively, it is clear that this approach strongly relies on some characteristics of the high quality solutions returned by the approximate algorithms. Firstly, it is important that these solutions have many components in common so that the resulting subproblem is of reasonably small size in order to be amenable to solving with exact algorithms. In fact, for the TSP it is a well known property that local minima cluster in a small region of the search space and that the better their quality, the more edges they have in common with an optimal solution [16, 47]; some evidence exists that similar conclusions hold for the p-median problem [40]. Secondly, the resulting subproblem is required to contain (all) the important decision variable to ensure that the solution found in the second stage solved by an exact algorithm can return a very high quality solution.

A similar approach is followed by Finger, Stützle, and Lourenço for the set covering problem (SCP) [31]. Finger et al. start by making a fitness distance correlation analysis of the SCP search space. Based on the result that for some classes of SCP instances the characteristics mentioned above were satisfied, a subproblem is defined by the solutions returned by several local searches and then solved using a simulated annealing approach for the SCP [17]. The reason why a local search (and not an exact) algorithm is used in the second stage is that for the largest SCP instances tackled some of the resulting subproblems are still quite large and require considerable computation time.

Finally, we mention that the collected information approach may run into problems if the approximate algorithms have difficulties in finding feasible solutions or when the problem is essentially a feasibility problem. In this case, it is an open issue how the information based on running approximate algorithms can be used. However, several approaches that try to identify infeasible subproblems that induce the infeasibility of the original problem were proposed. An example is the algorithm by Eisenberg and Faltings, where information obtained from running a local search algorithm is used to derive variable ordering strategies for quickly identifying infeasible subproblems by a backtracking type algorithm as quickly as possible. Initial results obtained by Eisenberg and Faltings on graph colouring problems appear to be promising [30].

4 Using bounds in construction heuristics

We first mention that throughout this section, we assume that we need to solve minimisation problems. For maximisation problems the notation and nomenclature has to be adapted in the standard way. The use of lower bounds is essential in exact algorithms, but lower bound related information is typically not exploited in approximate algorithms that are applied iteratively. Here, we present a recent proposal, where approximate algorithms make use of lower bounds.

4.1 Framework

Construction heuristics build solutions by starting from an empty partial solution and iteratively adding solution components until a complete solution is obtained. Construction heuristics typically rate the desirability of adding a solution component based on a heuristic measure of its objective function contribution. Then, they either add a solution component greedily, that is by choosing the best rated component, or probabilistically, biased by the heuristic information.

Instead of using the direct contribution of solution components to the objective function value, a different approach is to use lower bound computations to direct the solution construction, as it is described in the following algorithmic outline.

Algorithm 4.1 *Solving subproblems in order to obtain lower bounds*

```
Step 1: Initialisation
        Let $S_p$ be an empty partial solution.
Step 2: Construction phase
        while $S_p$ is not a complete solution do
            for all solution components $s_c$ do
            Compute a lower bound for $S_p \cup \{s_c\}$.
            endfor
            Choose the solution components $s_c$ to be added either greedily or probabilistically
            biased by the lower bound information.
            Update $S_p$.
        enddo
Step 3: Return completed solution.
```

If lower bounds are used to direct the construction of the solution, the desirability of extending a solution in a specific direction is stronger, when the lower bounds are smaller. Lower bounds in construction heuristics are especially useful if a large number of solutions is constructed, as it is the case in the following example.

4.2 Example: ANTS

Ant Colony Optimisation (ACO) [27, 28] is a recent metaheuristic approach for solving hard COPs. In ACO, (artificial) ants are stochastic solution construction procedures that probabilistically build a solution by iteratively adding solution components to partial solutions while taking into account (i) heuristic information on the problem instance being solved, if available, and (ii) (artificial) pheromone trails. Pheromone trails in ACO serve as distributed, numerical information which is adapted during the execution of the algorithm to reflect the search experience. Of all the available ACO algorithms (see [28] for an overview), the **A**pproximate **N**ondeterministic

Tree **S**earch (ANTS) algorithm [54] follows closely the idea outlined above, since it uses information obtained from lower bound computations as heuristic information for guiding the solution construction of the ants. According to [54] the term ANTS derives from the fact that the algorithm can be interpreted as an **A**pproximate **N**ondeterministic **T**ree **S**earch since it shares several elements with an approximated Branch-and-Bound procedure. In fact, in [54] the ANTS algorithm is extended to an exact algorithm; we refer the interested reader to the original reference for details; here we only present the heuristic algorithm.

ANTS was first applied to the quadratic assignment problem (QAP). In the QAP, a set of objects has to be assigned to a set of locations with given distances between the locations and given flows between the objects; the goal in the QAP is to place the objects on locations in such a way that the sum of the product between flows and distances is minimal. More formally, in the QAP one is given n objects and n locations, values a_{ij} representing the distance between locations i and j and b_{rs} representing the flow between objects r and s. Let x_{ij} be a binary variable which takes value 1 if object i is assigned to location j and 0 otherwise. The problem can be formulated as follows:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n} a_{ij} b_{kl} x_{ik} x_{jl} \tag{11}$$

subject to the constraints

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1,\ldots,n \tag{12}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1,\ldots,n \tag{13}$$

$$x_{ij} \in \{0,1\} \qquad i,j = 1,\ldots,n \tag{14}$$

In ANTS, each ant constructs a solution by iteratively assigning objects to a free location. Hence, the solution components to be considered are the couplings between objects and locations (there are n^2 such solution components). Given a location j, an ant decides to assign object i to this location with the following probability:

$$p^k_{ij}(t) = \frac{\alpha \cdot \tau_{ij}(t) + (1-\alpha)\cdot \eta_{ij}}{\sum_{l \in \mathcal{N}^k_j} \alpha \cdot \tau_{lj}(t) + (1-\alpha)\cdot \eta_{lj}} \qquad \text{if } i \in \mathcal{N}_j \tag{15}$$

Here, $\tau_{ij}(t)$ is the pheromone trail associated to the assignment of object i to a location j, which gives the "learned" desirability of choosing this assignment (pheromones vary at run-time), η_{ij} is the heuristic desirability of this assignment, α is a weighting factor between pheromone and heuristic, and $\mathcal{N}_j$ is the feasible neighbourhood, that is, the set of objects that are not yet assigned to a location.

Lower bound computations are exploited at various places in ANTS. Before starting the solution process, ANTS first computes the Gilmore-Lawler lower bound (GLB) [34, 48]. The GLB is defined to be the solution to the assignment problem

$$\min \sum_{i=1}^{n}\sum_{j=1} c_{ij} x_{ij}$$

subject to constraints (12) to (14), where the coefficients c_{ij} are defined as a function of distances and flows and the variables x_{ij} are binary. It is known that the assignment problem has the integrality property and therefore a solution of it can be obtained by simply solving its linear programming relaxation. Along with the lower bound computation one gets the values of the dual variables u_i and v_i, $i = 1, \dots, n$, corresponding to the constraints (12) and (13), respectively. The dual variables v_i are used to define a pre-ordering on the locations: the higher the value of the dual variable associated to a location, the higher the impact of the location on the QAP solution cost is assumed to be. Hence, the assignment of an object to that locationis tried earlier.

The essential idea of ANTS is to use computations on the lower bounds on the completion cost of a partial solution in order to define the heuristic information about the attractiveness of adding a specific solution component (i, j) in (15). This is achieved by tentatively adding the solution component to the current partial solution and by estimating the cost of a complete solution, (containing that solution component), by means of a lower bound. This estimate is then used to influence the probabilistic decisions taken by the ant during the solution construction: the lower the estimate, the more attractive the addition of a specific pair.

A disadvantage of the GLB, which is computed in an initialisation phase of the algorithm, is that its computation is $\mathcal{O}(n^3)$. This is clearly expensive, since a lower bound has to be computed at each step during a solution construction of an ant. Therefore, Maniezzo proposes a lower bound weaker than GLB, the so-called LBD bound, which exploits the values of the dual variables to estimate the completion cost of a partial solution. Although LBD can be shown to be weaker than GLB, the main advantage of using LBD is its low computational complexity, which is $\mathcal{O}(n)$. For details on the lower bound computation we refer to [54]. Experimental results have shown that ANTS is currently one of the best available algorithms for the QAP. The good performance of the ANTS algorithm has also been confirmed in a variety of further applications [55, 56].

4.3 Discussion

In general, the use of lower bounds in a construction algorithm has several advantages. Firstly, a substantial amount of research effort in the area of exact algorithms has been spent on obtaining and improving lower bounds and for many problems good lower bounds are available. Secondly, lower bounds can give a good indication of the desirability of specific ways of extending solutions. Thirdly, if the solution construction is repeated often, several solution components may be eliminated from consideration (if they lead to solutions that are worse than the best solution identified so far). So far however, lower bounds have not been used much in constructive heuristics. One exception is the Beam-search heuristic, which is a tree search algorithm that keeps at each iteration a set of nodes of a search tree and expands them in several directions according to a selection based on lower bounds [64]. In fact, beam search is derived from ideas from Branch-and-Bound algorithms. In the future we expect more approaches that are based on these ideas, possibly combined with other ways of exploring or defining search trees such as in iterated greedy algorithms [68]. However, since lower bounds are used at each construction step, efficient computation of them will be essential for the success of such approaches.

Information based on lower bound computations through Lagrangean relaxation have been exploited in several algorithms, for example, for the set covering problem. The central idea in these approaches is to use Lagrangean multipliers and costs to guide construction heuristics. Among the first approaches that use Lagrangean relaxation is the heuristic by Beasley [14]. Currently, several of the state-of-the-art algorithms for the set covering problem are, at least in part, based on essentially the same ideas [21].

5 Defining the search space areas to explore

The class of methods that we introduce in this section the areas of the search space to be explored by a local search procedure, by solving one or more relaxations of the initial problem.

5.1 Framework

Another method that we discuss in this paper combines a local search approach with an exact algorithm that solves some linear programming subproblems, which are created in order to reduce the search space and to define areas of the search space for the local search algorithm. The subproblems are relaxations of an integer programming model of the initial problem, which are (optionally) strengthened by the addition of some extra constraints. The optimal solutions of these subproblems are then used to define the search space for the local search algorithm. Formally, the general procedure can be described as follows:

Algorithm 5.1 *Solving subproblems in order to define the search space areas*

```
Step 1:Relaxation
        Define a relaxation of the original problem.
Step 2: Generate and solve subproblems
        (i) Using the relaxation from Step 1, define some subproblems to solve.
        (ii) Eliminate the subproblems that cannot provide useful information.
        (iii) Solve the remaining problems by an exact algorithm.
Step 3: Local search
        (i) Define the search space using the optimal solutions obtained at Step 2 (iii).
        (ii) Apply local search.
```

An example of such a method is given by Vasquez and Hao for the 0-1 multidimensional knapsack problem [84]. They combine tabu search with an exact method, in this case a simplex algorithm that solves some linear programming subproblems.

5.2 Example: Simplex and Tabu Search hybrid

Given m resources, each having a corresponding resource limit, n objects, each with an associated profit and an associated vector of resource consumption, the 0-1 multidimensional knapsack problem seeks to maximise the total profit made by using the objects such that the amount of resources consumed in within the resource limits. In what follows we will denote the profit associated with object i by p_i, the amount of resource j consumed by selecting object i by r_{ji}, and the limit of resource j by R_j. The standard integer programming formulation of the 0-1 multidimensional knapsack problem can be written as follows:

$$\max \sum_{i=1}^{n} p_i x_i$$

$$\text{s.t.} \sum_{i=1}^{n} r_{ji} x_i \leq R_j, \quad \forall j = 1, \ldots, m \tag{16}$$

$$x_i \in \{0,1\}, \qquad \forall i = 1, \ldots, n, \tag{17}$$

where $x_i = 1$ if object i is used and $x_i = 0$ otherwise. We will refer to this formulation as (MK).

The method we discuss here starts by relaxing the integrality constraint. Since the solution of the LP relaxation of an integer programming problem can be far away from the integer optimal solution, the LP relaxation is then “strenghtened” by the addition of an extra constraint. It is clear that the integer optimal solution of the 0-1 multidimensional knapsack problem can have only a certain number of components that are *not* zero. Based on this observation, the constraint added to the LP relaxation enforces the number of non-zero components to be equal to k, where $0 \leq k \leq n$ (see constraint (18) below). Therefore $n + 1$ linear programming problems are obtained (one for each value of k):

$$\begin{aligned} \max \ & \sum_{i=1}^{n} p_i x_i \\ \text{s.t.} \ & \sum_{i=1}^{n} r_{ji} x_i \leq R_j, \quad \forall j = 1, \ldots, m \\ & \sum_{i=1}^{n} x_i = k && (18) \\ & x_i \in [0,1], \qquad \forall i = 1, \ldots, n && (19) \end{aligned}$$

We denote such a subproblem by $\mathcal{P}(k)$. Clearly, the solutions of these problems can be fractional, however it is hoped that the optimal solution of the original problem is close to one of the optimal solutions obtained after solving the relaxed problems.

Vasquez and Hao propose a reduction of the number of the subproblems that need to be solved by calculating bounds on k. This is done by rounding up or down the optimal solutions of two new LP problems. Let x is a feasible solution of the LP relaxation of (MK) and LB a lower bound on the optimal solution of the original problem. The two LP problems seek to minimise, respectively maximise $\sum_{i=1}^{n} x_i$, such that $\sum_{i=1}^{n} p_i x_i \geq LB + 1$. The integer values obtained after rounding up, respectively down, the optimal solutions of the problems described above provide a lower, respectively upper bound on k. We note that while Vasquez and Hao mention that a lower bound LB can be found heuristically, they do not provide any details about how they obtained such a bound.

After the reduction, the LP subproblems that need to be considered are solved by the simplex algorithm. The solutions returned are then used to generate starting solutions to a following tabu search phase. More precisely, an initial solution is obtained from a possibly fractional solution $x^{[k]}$ to $\mathcal{P}(k)$ by fixing its greatest k elements to one and the rest to zero. The solution $x^{[k]}$ of $\mathcal{P}(k)$ has the additional use to restrict the search space explored by a tabu search procedure. In fact, the subsequently run tabu search algorithm, which is based on the reverse elimination method [35], is restricted to explore only solutions x for which it holds that $\sum_{i=1}^{n} |x_i - x_i^{[k]}| \leq \delta_{max}$, that is, the solutions explored by the tabu search are limited to a ball of radius δ_{max} around $x^{[k]}$. The radius δ_{max} is computed heuristically as follows. Let no_int denote the number of integer components of $x^{[k]}$ and no_frac the number of fractional components; then, the radius is set to a value of $\delta_{max} \approx 2 \times (no\_int + no\_frac - k)$. Note that $\delta_{max} = 0$ corresponds to the case when an integer solution is returned by the simplex. The search space is further reduced by limiting the number of objects taken into configurations; for each k, the number of non-zero components of a candidate solution is considered to be exactly k and only the *integer* candidate solutions are considered. The neighbourhoods are defined as add/drop neighbourhoods; the neighbourhood of a given configuration is the set of configurations that differ by exactly two elements from the initial configuration, while keeping the number of non-zero elements constant.

An extensive computational study of the final algorithm is provided. The authors report many improved results compared to the best known results at the time of publication; however they acknowledge large computational time of about three days on a set of computers with Pentium 300MHz or 500 MHz computers for very large instances with as many as 2,500 items and 100 constraints.

5.3 Discussion

The paper of Hao and Vasquez is an example of how information of solutions of relaxations of IP formulations can be used to restrict the area of the search space examined by local search algorithms and to provide good initial solutions for the local search. Here, the underlying conjecture is that the information obtained from the relaxation helps to restrict the search space to areas where good (or ideally optimal) solutions are located. Certainly, the crucial part of the algorithm of Hao and Vazques is the extent to which this hypothesis holds; in fact, the very high quality solution reached by the method of Hao and Vasquez suggests that for the multi-dimensional knapsack problem this hypothesis is met. However, an additional (search space) analysis would be welcome to understand why it is successful in this case.

6 Enhancing metaheuristics

Occasionally, exact methods are employed when metaheuristics try to add specific features of diversification or intensification of the search that are beyond the use of exploring neighbourhoods in iterative improvement algorithms. In this section we discuss one explicit example in the context of Iterated Local Search.

6.1 Framework, Iterated Local Search

An approach to solving difficult COPs that has been very successful is the Iterated Local Search (ILS) [53]. ILS consists of running a local search procedure starting from solutions obtained by perturbing previously found local optima. A simple form of an ILS is given in Algorithm 6.1.

Algorithm 6.1 *Iterated Local Search*

```
Step 1: Let $S$ be an initial solution.
Step 2: while stopping_criterion is not met do
(i)         Let $S'$ be the solution obtained from $S$ after a perturbation.
(ii)        Call local_search($S'$) to produce the solution $S''$.
(iii)       if $S''$ is accepted as the new incumbent solution then $S = S''$.
        enddo
Step 3: Return $S$.
```

There are many implementation choices for ILS. These mainly concern the way the perturbation in Step 2(i) is performed, the choice of the local search algorithm in Step 2(ii), and the particular acceptance criterion that is used in Step 2(iii).

In what follows we focus on the choice of the perturbation. The main role of the perturbation in ILS is to introduce significant changes in the current solution in order to allow the local search to explore different local optima, while still conserving good characteristics of a current solution. Simple ways of introducing such perturbations, like applying a random move in a large neighbourhood, are usually enough to obtain good performance. However, state-of-the-art performance is often reached by more problem specific perturbations and when the perturbations introduce some structural changes that cannot be reversed easily by a local search [53].

One central possibility of combining exact algorithms with ILS is to let an exact algorithm determine the perturbation. This is done by first fixing some part of the solution and leaving the rest free. Next, the free part is optimised, conditioned to the fixed part and the solution to the free part is reinserted into the fixed part, possibly after restoring feasibility, in case this should be necessary. In other words, at Step 2(i) of Algorithm 6.1 a subproblem is solved to optimality. Formally, Step 2(i) is replaced by:

Procedure 6.1 *Solving a subproblem to determine the perturbation (new Step 2(i) of ILS)*

```
begin
  Let $S'' = S \setminus R$, where $R \subset S$.
  Define $\mathcal{P}(R)$, a subproblem that depends on $R$.
```

```
    Let $opt(\mathcal{P}(R))$ be the optimal solution of $\mathcal{P}(R)$.
    Let $\bar{S}' = S'' \cup opt(\mathcal{P}(R))$.
    Modify $\bar{S}'$ such that feasibility is restored.
    Return $S'$, be the modified feasible solution.
end
```

The use of an exact algorithm is likely to introduce larger structural changes, which cannot be easily undone by the local search.

6.2 Example: Perturbation in ILS for job-shop scheduling

An example of how to determine a perturbation by using exact algorithms to solve a subproblem is given by Lourenço [52]. In her paper, Lourenço conducts an extensive computational study of ILS methods applied to the job-shop scheduling problem (JSP). The JSP is defined for m machines and n jobs. We denote the jobs by $J_1, \dots, J_n$. Each job consists of a sequence of operations that have to be performed in a given order. Each operation has to be executed for a specified, uninterrupted time (i.e., preemption is not allowed). We describe a job J_k as the sequence of operations $o_1^k, \dots, o_{r_k}^k$, where r_k is the number of operations of J_k. The total number of operations is $r = r_1 + \dots + r_n$. In the JSP, precedence constraints induce a total order on the operations of each job. Additional constraints require that each machine handles at most one job at a time. A feasible schedule is a schedule that satisfies all the precedence and capacity constraints. Then, the JSP consists in finding a feasible schedule that minimises the overall job completion time.

The job-shop scheduling can be modelled using the disjunctive graph $G = (V, A, E)$ representation, where V is the vertex set corresponding to the operations, A is the arc set corresponding to the job precedence constraints, and E is the edge set corresponding to the machine capacity constraints [75].

In order to formally define the components of the graph we use the following notation. We write $i \prec j$ if operation i precedes operation j, and O is the set of operations, where $|O| = r$. We use $\mathcal{U}$ to denote set of operations that start the n jobs, i.e. $\mathcal{U} = \{o_1^1, \dots, o_1^n\}$. Similarly, let $\mathcal{T}$ be the set of operations that end the jobs, i.e $\mathcal{T} = \{o_{r_1}^1, \dots, o_{r_n}^n\}$. Finally, we use (i, j) to denote an arc between nodes i and j and $[i, j]$ for an edge (un-oriented arc) between i and j.

We can now give a formal description of G. The set of nodes is $V = \{1, \dots, r\} \cup \{s, t\}$, where node $i \in \{1, \dots, r\}$ corresponds to the ith operation from O, and s and t are two nodes called *start* and *end* node, respectively. The set of arcs can be described by: $A = \{(s, i) : i \text{ corresponds to a job in } \mathcal{U}\} \cup \{(i, t) : i \text{ corresponds to a job in } \mathcal{T}\} \cup \{(i, j) : i \prec j, i = 1, \dots, r\}$. Finally, $E = \{[i, j] : i, j \text{ are executed on the same machine}\}$. If no extra constraints are given, each node corresponding to an operation has an associated weight, which is equal to the processing time of that operation, while the start and end nodes have weights zero.

In this graph, the scheduling decision corresponds to orienting each edge in E. A feasible schedule is obtained when all edges are oriented and the resulting graph is acyclic. The overall completion time is given by the total weight of the critical path from s to t, where the total weight of a path is defined to be the sum of the weight associated with all nodes visited by that path.

Lourenço experimentally tested several variants of ILS, including several variants of the perturbation. The first perturbation procedure proposed by Lourenço is making use of Carlier's algorithm [22], which is a branch-and-bound method applied to the one-machine scheduling problem. This problem can be seen as a very simple version of the job-shop scheduling problem: a number of operations need to be scheduled on one machine in the presence of temporal constraints. Lourenço's idea is to modify the directed graph corresponding to the current solution of the job-shop scheduling problem by removing all the directions given to the edges associated with two random machines. Then Carlier's algorithm is applied to one of the machines; the problem to solve is therefore a one-machine scheduling problem. Next, the edges corresponding to that machine are oriented according to the optimal solution obtained. Finally, the same treatment is applied to the second machine. Lourenço mentions that this perturbation idea can create cycles in the graph and suggests a way of obtaining a feasible schedule from the graph with cycles (see [52] for details). In conclusion, at each iteration of the ILS, two subproblems are solved in order to construct a new initial solution for the local search procedure. The subproblems are of a different type than the original problem and of reduced size. However they belong to the same class of job scheduling problems.

A similar perturbation proposed by Lourenço is making use of the early-late algorithm [51] as an exact method. In order to do that preemption is allowed and two one-machine problems with lags on a chain are solved. Lourenço also gives a simple technique for eliminating cycles. We note that in this case the subproblems solved are of a different type compared to the original problem.

Finally, it should be said that the computational results of Lourenço cannot be considered state-of-the-art performance; the main reason for this is that she used a rather weak local search algorithm when compared to the effective tabu search algorithms proposed by Nowicki and Smutnicki [63] or the local search procedure of Balas and Vazacopoulos [13]. However, it would be interesting to see how the performance of these two algorithms would be affected by the addition of the perturbation steps proposed by Lourenço.

6.3 Other approaches

Exact algorithms can be used within particular metaheuristics for specific operations that are to be done while searching for solutions. Apart from the perturbations in ILS, other examples can be found in applications of genetic algorithms; here, exact algorithms are typically used in the recombination operation, in which two solutions S and S' are combined to form one or several new solutions. The central idea is to define a subproblem `Recombination`(S, S') that comprises all the solutions that can result following a particular way of combining S and S' and then to search for the best solution of the resulting subproblem. The subproblem can be obtained by keeping common solution features of the two ”parent” solutions fixed and try to find the best solution for the free parts (similar to the tour merging or the heuristic concentration approach, which was presented in Section 3) or to define a subproblem consisting of the union of the solution components contained in the two parent solutions S and S'. An example for the first approach is presented by Yagiura and Ibaraki [86]. They apply this idea to permutation problems using a dynamic programming algorithm for finding an optimal permutation subject to the constraint that a partial order common to the two parent solutions S and S' is maintained by the new solution. Examples of the second idea are presented in the paper by Balas and Niehaus [12] on the maximum clique problem (MCP) and the paper by Aggarwal, Orlin, and Tai for the closely related maximum independent set problem (MISP) [2]. The MCP and MISP are subset problems in graphs, where a subset of the vertices needs to be chosen. Hence, in both cases, a solution corresponds to a subset of the vertices of the original graph $G = (V, A)$, that is, we have $S, S' \subset V$. Then, in both cases, the subproblem consists in a subgraph comprising all the vertices in $S \cup S'$ and all edges that connect them. It can be shown that the resulting subproblem is a matching problem in a bipartite graph and, hence, it can be solved in polynomial time [2, 11].

6.4 Discussion

Clearly, the techniques discussed here are specific to particular metaheuristics. However, these examples indicate possible areas of interest, where such combinations can be useful. In general, mainly hybrid metaheuristics will be candidates for possible combinations; we call hybrid metaheuristics those metaheuristics that consist of the combination of several clearly distinct procedures like perturbation and local search in the ILS case or recombination, mutation, and (possibly) local search in the genetic algorithm case. In fact, the two examples illustrate well the potential of such combinations. Other metaheuristics that could profit from such combinations are ant colony optimisation [28], for example by optimising partial solutions, or scatter search [38], in a way analogous to what is described for the genetic algorithms.

7 Conclusions

In this paper we reviewed existing approaches that combine combine local search and exact algorithms derived from IP methods for the solution of $\mathcal{NP}$-hard combinatorial optimisation problems. We focused on techniques where the ”master” routine is based on local search and strengthened by the use of exact algorithms. In these approaches exact algorithms are used to solve subproblems that arise during the search process. In addition, we restricted our survey to methods where at least one of the local search or exact algorithms is used more than once.

We classified the combination algorithms in five different groups. The probably most important one consists of methods in which exact algorithms are used to explore large (typically exponential size) neighbourhoods in local search algorithms. Here, two fundamentally different ways of combining local search and exact algorithms were identified, namely those where exact algorithms scan the full neighbourhood and the solution found by the exact algorithm induces the next step of the local search to be executed (as examples we presented VLSN and Dynasearch) and those where exact algorithms explore parts of the full neighbourhood (like in the hyperopt approach). This first goup is also the one on which a significant amount of research has been done. A number of methods that use constraint programming on subproblems instead of an exact algorithm could also be classified as being strongly related to this group.

The second group of methods uses several runs of local search algorithms to collect information about solution components that are likely to be contained in very high quality solutions and defines reduced problems that can be succesfully tackled by exact algorithms. So far, few applications of this type of approaches exist, but at least for the TSP the resulting algorithms are currently among the best performing approximate algorithms.

A third group comprises methods that exploit lower bound computations (in the case of minimisation problems) to guide construction heuristics. These ideas have been exploited in a variety of state-of-the-art algorithms (like the ANTS algorithm).

The final two groups consist of algorithms that are more tailored to the particular application problem or the particular local search techniques used. In the first of these two group, the optimal solutions of subproblems are used to reduce the search space for the local search. Here, the only example we are aware of is the simplex–tabu search hybrid by Hao and Vazques. The second of these two groups contains methods that run exact algorithms to solve some subproblems that arise when specific metaheuristics are used. Examples are the implementation of a solution perturbation in iterated local search or the determination of an offspring through recombination in genetic algorithms.

Clearly, a different way of combining local search and exact algorithms is to run an exact algorithm as the master routine and the local search as a subroutine. However, this is a different issue and will possibly be treated in a separate future paper.

The main conclusion of our paper is that there are many research opportunities to develop algorithms that integrate local search and exact techniques. Despite the fact that local search and exact algorithms have somewhat complementary advantages and disadvantages, surprisingly few algorithms exists that try to combine the two areas. One reason for this may be that such combined methods are often rather complex and hence they involve long development times. A possibly more important obstacle is that they require strong knowledge about the techniques available in two widely different techniques that, even worse, are often treated in different research streams in combinatorial optimisation. Therefore, another conclusion of this paper is that a closer integration of these different research streams is highly desirable. In general, we strongly believe that combinations of exact methods, which need not necessarily be restricted to IP and local search methods are a very promising direction for future research in combinatorial optimisation.

Acknowledgements

This work was supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106 and by a European Community Marie Curie Fellowship, contract HPMF-CT-2001. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, UK, 1997.

[2] C. C. Aggarwal, J. B. Orlin, and R. P. Tai. An optimized crossover for the maximum independent set. *Operations Research*, 45:226–234, 1997.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Inc. Englewood Cliffs, NJ, 1993.

[4] R. K. Ahuja and J. B. Orlin. VLSN search papers. `http://www.ise.ufl.edu/ahuja/vlsn/`, 2003.

[5] R. K Ahuja, J. B. Orlin, and D. Sharma. Multi-exchange neighbourhood structures for the capacitated minimum spaning tree problem. Working Paper, 2000.

[6] R .K. Ahuja, J. B. Orlin, and D. Sharma. Very large-scale neighbourhood search. *International Transactions in Operational Research*, pages 301–317, 2000.

[7] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. On the solution of traveling salesman problems. *Documenta Mathematica*, Extra Volume ICM III:645–656, 1998.

[8] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Concorde: a code for solving traveling salesman problems. http://www.math.princeton.edu/tsp/concorde.html, 1999.

[9] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding Tours in the TSP. Technical Report 99885, Forschungsinstitut für Diskrete Mathematik, University of Bonn, Germany, 1999.

[10] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3:149–156, 1991.

[11] E. Balas and W. Niehaus. Finding large cliques in arbitrary graphs by bipartite matching. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993*, volume 26, pages 29–53. American Mathematical Society, 1996.

[12] E. Balas and W. Niehaus. Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems. *Journal of Heuristics*, 4(2):107–122, 1998.

[13] E. Balas and A. Vazacopoulos. Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 44(2):262–275, 1998.

[14] J. E. Beasley. A lagrangian heuristic for set covering problems. *Naval Research Logistics*, 37:151–164, 1990.

[15] J. L. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4(4):387–411, 1992.

[16] K. D. Boese, A. B. Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial global optimization. *Operations Research Letters*, 16:101–113, 1994.

[17] M. J. Brusco, L. W. Jacobs, and G. M. Thompson. A morphing procedure to supplement a simulated annealing heuristic for cost- and coverage-corrleated set covering problems. *Annals of Operations Research*, 86:611–627, 1999.

[18] K. Büdenbender, T. Grünert, and H.-J. Sebastian. A hybrid tabu search/branch-and-bound algorithm for the direct flight network design problem. *Transportation Science*, 34(4):364–380, 2000.

[19] E. K. Burke, P. Cowling, and R. Keuthen. Embedded local search and variable neighbourhood search heuristics applied to the travelling salesman problem. Technical report, University of Nottingham, 2000.

[20] E. K. Burke, P.I. Cowling, and R. Keuthen. Effective local and guided variable neighbourhood search methods for the asymmetric travelling salesman problem. *EvoWorkshop 2001, LNCS*, 2037:203–212, 2001.

[21] A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. *Operations Research*, 47(5):730–743, 1999.

[22] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11:42–47, 1982.

[23] Y. Caseau and F. Laburthe. Disjunctive scheduling with task intervals. Technical Report LIENS 95-25, Ecole Normale Superieure Paris, France, July 1995.

[24] R. K. Congram. *Polynomially Searchable Exponential Neighbourhoods for Sequencing Problems in Combinatorial Optimisation*. PhD thesis, University of Southampton, Faculty of Mathematical Studies, UK, 2000.

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

[26] W. Cook and P. Seymour. Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248, 2003.

[27] M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw Hill, London, UK, 1999.

[28] M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications and advances. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 251–285. Kluwer Academic Publishers, Norwell, MA, 2002.

[29] I. Dumitrescu. *Constrained Shortest Path and Cycle Problems*. PhD thesis, The University of Melbourne, 2002. http://www.intellektik.informatik.tu-darmstadt.de/˜irina.

[30] C. Eisenberg and B. Faltings. Making the breakout algorithm complete using systematic search. In G. Gottlob and T. Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1374–1375. Morgan Kaufmann Publishers, San Francisco, CA, 2003.

[31] M. Finger, T. Stützle, and H. R. Lourençou. Exploiting fitness distance correlation of set covering problems. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. Raidl, editors, *Applications of Evolutionary Computing*, volume 2279 of *Lecture Notes in Computer Science*, pages 61–71. Springer Verlag, Berlin, Germany, 2002.

[32] F. Focacci, F. Laburthe, and A. Lodi. Local search and constraint programming. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 369–403. Kluwer Academic Publishers, Norwell, MA, 2002.

[33] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco, CA, 1979.

[34] P. C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the SIAM*, 10:305–313, 1962.

[35] F. Glover. Tabu search. *ORSA Journal on Computing*, 2:4–32, 1990.

[36] F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1–3):223–253, 1996.

[37] F. Glover and G. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic Publishers, Norwell, MA, 2002.

[38] F. Glover, M. Laguna, and R. Martí. Scatter search and path relinking: Advances and applications. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 1–35. Kluwer Academic Publishers, Norwell, MA, 2002.

[39] P. Hansen and N. Mladenovic. Variable neighborhood search: Principles and applications. Invited papers at Euro XIV, 1998. Brussels, Belgium.

[40] P. Hansen and N. Mladenoviç. Variable neighbourhood search for the p-median. *Location Science*, 5(4):207–226, 1998.

[41] P. Hansen and N. Mladenovic. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, chapter An Introduction to Variable Neighborhood Search, pages 433–458. Kluwer Academic Publishers, Boston, MA, 1999.

[42] P. Hansen, N. Mladenovic, and D. Perez-Britos. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4):335–350, 2001.

[43] K. Helsgaun. An effective implementation of the lin-kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126:106–130, 2000.

[44] H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann Publishers, 2004.

[45] ILOG. ILOG CPLEX 8.0, Reference manual, 2002.

[46] D. S. Johnson and L. A. McGeoch. Experimental analysis of heuristics for the STSP. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 369–443. Kluwer Academic Publishers, 2002.

[47] S. Kirkpatrick and G. Toulouse. Configuration space analysis of travelling salesman problems. *Journal de Physique*, 46(8):1277–1292, 1985.

[48] E. L. Lawler. The quadratic assignment problem. *Management Science*, 9:586–599, 1963.

[49] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the travelling salesman problem. *Operations Research*, 21:498–516, 1973.

[50] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the Travelling Salesman Problem. *Operations Research*, 21:498–516, 1973.

[51] H. R. Lourenço. *A Computational Study of the Job-Shop and the Flow-Shop Scheduling Problems*. PhD thesis, School of Or & IE, Cornell University, Ithaca, NY, 1993.

[52] H. R. Lourenço. Job-shop scheduling: Computational study of local search and large-step optimization methods. *European Journal of Operational Research*, 83:347–367, 1995.

[53] H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321–353. Kluwer Academic Publishers, Norwell, MA, 2002.

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

[55] V. Maniezzo, A.Carbonaro, M.Golfarelli, and S.Rizzi. An ANTS algorithm for optimizing the materialization of fragmented views in data warehouses: Preliminary results. In E. J. W. Boers, J. Gottlieb, P. L. Lanzi, R. E. Smith, S. Cagnoni, E. Hart, G. R. Raidl, and H. Tijink, editors, *Applications of Evolutionary Computing*, volume 2037 of *Lecture Notes in Computer Science*, pages 80–89. Springer Verlag, Berlin, Germany, 2001.

[56] V. Maniezzo and A. Carbonaro. An ANTS heuristic for the frequency assignment problem. *Future Generation Computer Systems*, 16(8):927–935, 2000.

[57] K. Marriott and P. Stuckey. *Programming with Constraints*. MIT Press, Cambridge, MA, 1998.

[58] T. Mautor. Intensification neighbourhoods for local search methods. In C. C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 493–508. Kluwer Academic Publishers, Norwell, MA, 2002.

[59] T. Mautor and P. Michelon. MIMAUSA: A new hybrid method combining exact solution and local search. In *Extended abstracts of the 2nd International Conference on Metaheuristics*, page 15, Sophia-Antipolis, France, 1997.

[60] T. Mautor and P. Michelon. MIMAUSA: an application of referent domain optimization. Technical Report 260, Laboratoire d'Informatique, Université d'Avignon et des Pays de Vaucluse, Avignon, France, 2001.

[61] J. D. Morris. On the extent to which certain fixed-charge depot location problems can be solved by LP. *Journal of the Operational Research Society*, 29:71–76, 1978.

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

[63] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job-shop problem. *Management Science*, 42(2):797–813, 1996.

[64] P. S. Ow and T. E. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26:297–307, 1988.

[65] G. Pesant and M. Gendreau. A view of local search in constraint programming. In E. Freuder, editor, *Proceedings of Constraint Programming 1996*, volume 1118 of *Lecture Notes in Computer Science*, pages 353–366. Springer Verlag, Berlin, Germany, 1996.

[66] G. Pesant and M. Gendreau. A constraint programming framework for local search methods. *Journal of Heuristics*, 5:255–279, 1999.

[67] C. N. Potts and S. van de Velde. Dynasearch: Iterative local improvement by dynamic programming; part I, the traveling salesman problem. Technical Report LPOM–9511, Faculty of Mechanical Engineering, University of Twente, Enschede, The Netherlands, 1995.

[68] M. Pranzo and T. Stützle. Iterated greedy: Methodology, algorithms, and applications. Manuscript, 2003.

[69] C. S. ReVelle and R. Swain. Central facility location. *Geographical Analysis*, 2:30–42, 1970.

[70] N. Robertson and P. D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory*, 52:153–190, 1991.

[71] K. E. Rosing. Heuristic concentration: a study of stage one. *ENVIRON PLANN B*, 27(1):137–150, 2000.

[72] K. E. Rosing and C. S. ReVelle. Heuristic concentration: Two stage solution construction. *European Journal of Operational Research*, pages 955–961, 1997.

[73] K. E. Rosing and C. S. ReVelle. Heuristic concentration and tabu search: A head to head comparison. *European Journal of Operational Research*, 117(3):522–532, 1998.

[74] K. E. Rosing, C. S. ReVelle, and H. Rossing-Vogelaar. The p-median and its linear programing relaxation: An approach to large problem. *Journal of the Operational Research Society*, 30:815–823, 1979.

[75] B. Roy and B. Sussmann. Les problemes d'ordonnancement avec constraintes disjonctives. Notes DS no. 9 bis, SEMA.

[76] M. Savelsbergh. MINTO - Mixed INTeger Optimizer. `http://www.isye.gatech.edu/faculty/Martin_Savelsbergh/software/`, 2002.

[77] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. Technical report, ILOG S.A., Gentily, France, 1998.

[78] É. D. Taillard and S. Voss. POPMUSIC: Partial optimization metaheuristic under special intensification conditions. In C. Ribeiro and P. Hansen, editors, *Essays and Surveys in metaheuristics*, pages 613–629. Kluwer Academic Publishers, Boston, MA, 2002.

[79] M. B. Teitz and P. Bart. Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research*, 16:955–961, 1968.

[80] P. M. Thompson and J. B. Orlin. The theory of cycle transfers. Working Paper No. OR 200-89, 1989.

[81] P. M. Thompson and H. N. Psaraftis. Cyclic transfer algorithm for multivehicle routing and scheduling problems. *Operations Research*, 41:935–946, 1993.

[82] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial & Applied Mathematics, Philadelphia, PA, 2001.

[83] P. van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, Cambridge, MA, 1999.

[84] M. Vasquez and J-K. Hao. A hybrid approach for the 0-1 multidimensional knapsack problem. In *Proceedings of the IJCAI-01*, pages 328–333, 2001.

[85] C. Voudouris. *Guided Local Search for Combinatorial Optimization Problems*. PhD thesis, Department of Computer Science, University of Essex, Colchester, UK, 1997.

[86] M. Yagiura and T. Ibaraki. The use of dynamic programming in genetic algorithms for permutation problems. *European Journal of Operational Research*, 92:387–401, 1996.