Combinations of Local Search and Exact Algorithms

Irina Dumitrescu and Thomas Stützle

Darmstadt University of Technology, CS Department, Intellectics Group
Alexanderstr. 10, 64283 Darmstadt, Germany
{irina, tom}@intellektik.informatik.tu-darmstadt.de

Abstract. In this paper we describe the advantadges and disadvantages of local search and exact methods of solving NP-hard problems and see why combining the two approaches is highly desirable. We review some of the papers existent in the literature that create new algorithms from such combinations. In this paper we focus on local search approaches that are strengthened by the use of exact algorithms.

1 Introduction

Integer and combinatorial optimisation problems maximise or minimise functions of many variables subject to some problem specific constraints and integrality restrictions imposed on all or some of the variables. This class of problems comprises many real-life problems arising in airline crew scheduling, production planning, Internet routing, packing and cutting, and many other areas. Often, a combinatorial optimisation problem can be modelled as an integer program [18]. However, these problems are often very difficult to solve, which is captured by the fact that many such problems are NP-hard [9].

Because of their difficulty and enormous practical importance, a large number of solution techniques for attacking NP-hard integer and combinatorial optimisation problems have been proposed. The available algorithms can be classified into two main classes: *exact* and *approximate* algorithms. Exact algorithms are guaranteed to find the optimal solution and to prove its optimality for every finite size instance of a combinatorial optimisation problem within an instance-dependent, finite run-time. If optimal solutions cannot be computed efficiently in practice, the only possibility is to trade optimality for efficiency. In other words, the guarantee of finding optimal solutions can be sacrificed for the sake of getting very good solutions in polynomial time. A class of approximate algorithms is that of *heuristic methods*, or simply *heuristics*, and seek to obtain this goal.

Two techniques from each class that have had significant success are Integer Programming (IP), as an exact approach, and local search and extensions thereof called metaheuristics, as an approximate approach. IP is a class of methods that 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 [18]. In recent years remarkable improvements have been reported for IP when applied to some difficult problems (see for example [3] for the TSP). However, for most of the available IP algorithms the size of the instances solved is relatively small, and the computational time increases strongly with increasing instance

size. Additional problems are often due to the facts that (i) the memory consumption of exact algorithms may lead to the early abortion of a programme, (ii) high performing exact algorithms for one problem are often difficult to extend if some details of the problem formulation change, and (iii) for many combinatorial problems the best performing algorithms are highly problem specific and that they require large development times by experts on integer programming. Nevertheless, important advantages of exact methods from IP are that (i) *proven optimal* solutions can be obtained if the algorithm succeeds, (ii) valuable information on the upper/lower bounds to the optimal solution are obtained even if the algorithm is stopped before completion, and (iii) IP methods allow to prune parts of the search space in which optimal solutions cannot be located. A more practical advantage of IP methods is that powerful, general-purpose tools like CPLEX are available that often reach astonishingly good performance.

Local search has been shown to be the most successful class of approximate algorithms. It yields high-quality solutions by iteratively applying small modifications (local moves) to a solution in the hope of finding a better one. Embedded into metaheuristics designed to escape local optima like Simulated Annealing, Tabu Search, or Iterated Local Search, this approach has been shown to be very successful in achieving near-optimal (and sometimes optimal) solutions to a number of difficult problems [1]. Advantages of local search methods are that (i) in practice they are found to be the best performing algorithms for a large number of problems, (ii) they can examine an enormous number of possible solutions in short computation time, (iii) they are often more easily adapted to variants of problems and, thus, are more flexible, and (iv) they are typically easier to understand and implement than exact methods. However, disadvantages of local search algorithms are that typically (i) they cannot prove optimality, (ii) they cannot provably reduce the search space, (iii) they do not have well defined stopping criteria (this is particularly true for metaheuristics), and (iv) they often have problems with highly constrained problems where feasible areas of the solution space are disconnected. A drawback from a practical point of view is that there are no efficient general-purpose local search solvers available. Hence, most local search algorithms often require considerable programming efforts, although usually less than for 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, an obvious idea is to try to combine these two techniques into more powerful algorithms. In this article we are looking at approaches that strive for an integration of these two worlds. While the notion of *integration* is difficult to define, we are thinking of approaches that, although rooted in one of the worlds, take a significant portion of the other world. For example, here we exclude obvious combinations like those that use preprocessing and bound propagation. In fact, only a few articles strive for a real integration of the two approaches. In what follows we will briefly describe methods that have been proposed so far. Our paper does not claim to be an extensive survey of the area, but rather a personal selection of papers 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 problem, a local search technique is applied to the problem to solve and an exact algo-

rithm to some subproblems. The subproblems are either solved in order to define the neighbourhood to explore, to permit a systematic exploration of a neighbourhood, to exploit certain characteristics of some feasible solutions already discovered, or to find good bounds on the optimal solutions. We will describe these situations in more detail in the following sections. For the sake of brevity, in each section we will present in detail only one method.

2 Defining the neighbourhood to explore

The first method we discuss in this paper combines a local search approach with an exact algorithm that solves some linear programming subproblems defined in order to reduce the search space and to define neighbourhoods for the local search algorithm. The subproblems are relaxations of some kind of an integer programming model of the initial problem, which are strengthened by the addition of some extra constraints. The optimal solutions of these subproblems are then used to define the search space and the neighbourhoods for the local search algorithm.

An example of such a method is given in the paper of Vasquez and Hao [26], proposed for the 0-1 multidimensional knapsack problem.

Given n objects each of a known volume, m knapsacks of given capacities, and a value associated with each object, the 0-1 multidimensional knapsack seeks to maximise the total value of the objects put in the knapsacks such that the capacity of each knapsack is not exceeded. This problem can be formulated as:

$$\begin{aligned} &\min \sum cx \\ &\text{s.t. } Ax \leq b && (1)\\ &\quad x \in \{0,1\}^n, && (2) \end{aligned}$$

where $c \in \mathbf{N}^n$, $A \in \mathbf{N}^{m \times n}$, and $b \in \mathbf{N}^m$, $x \in \mathbf{B}^n$, with $x_i = 1$ if the object i is in the knapsack and $x_i = 0$, otherwise.

The method proposed by Vasquez and Hao starts by solving an LP relaxation of the integrality constraint (2) plus an extra constraint that ensures that the solution will be a good initial solution for the local search procedure. The extra constraint is based on the fact that the LP relaxation of an IP problem can be far away from the integer optimal solution, while it is clear that the integer optimal solution can have only a certain number of components that are not zero. This indeed is the extra constraint used by Vasquez and Hao: $\sum_{i=1}^{n} x_i = k$, where $k = 1, \ldots, n$. Clearly, the solutions of these problems will not be integer, 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. In order not to miss good search areas, every possible k needs to be considered, and therefore a series of linear programming subproblems need to be solved. Vasquez and Hao propose a reduction of the number of these problems by calculating some bounds on k. These bounds are the optimal solutions of two new LP problems solved by the simplex method. The two problems seek to minimise, respectively maximise, the sum of the components of x, $(\sum_{i=1}^{n} x_i)$, where x is a feasible solution for the LP relaxation of the original problem, $(Ax \leq b)$, such that the value of the objective function evaluated for x

is greater than or equal to the value of a known integer lower bound LB for the original problem plus one, ($cx \geq LB + 1$). While Vasquez and Hao mention that such a lower bound can be found by using a different heuristic, they do not provide any details about how they obtained such a bound.

The LP subproblems that remain to be solved after the reduction are solved by the simplex algorithm. Finally, Tabu Search based on the reverse elimination method [11] is used to search the reduced solution space defined as a collection of spheres of specified radii centred at optimal solutions of the LP subproblems solved previously. Vasquez and Hao propose a formula for calculating the radius of such a sphere, which depends on the number of integer and fractional non-zero components of the solution that is the centre of the sphere. 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 kept. In addition to this, only candidate solutions that improve the current best value of the objective function are kept. 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 could identify many improved results compared to the best known results at the time of publication; however, Vasquez and Hao acknowledge large computational time, especially for large instances.

In conclusion, the paper of Vasquez and Hao is an example of how easily solvable problems are used to reduce the search space explored by a local search for a difficult problem. While a branch-and-bound approach would also reduce the search space, the advantage of the method presented is that the number of subproblems solved is limited by $n + 2$, as opposed to being possibly exponential in the case of branch-and-bound. However, a drawback of this method is that after reducing the number of subproblems that need to be solved, no further reduction is possible. While in the case of a branch-and-bound approach the computation can be stopped early and information obtained up to that point can still be used, the method of Vasquez and Hao requires finding a solution for all the subproblems identified.

3 Exploring the neighbourhoods

In this section we present an approach which is extremely useful when the neighbourhoods defined are *very* large and efficient ways of exploring them are needed. When the local search procedure needs to move from the current solution to a neighbouring one, a subproblem is solved. The solution of the subproblem will determine the neighbour that will be used by the local search.

Hyperopt Neighbourhoods. Burke et al. give such a method in the context of the symmetric and asymmetric Travelling Salesman Problem (TSP) [5, 6] and of the Asymmetric Travelling Salesman Problem [6]. We recall that the symmetric TSP is defined on a undirected graph $G = (V, A)$, where $V = \{1, \ldots, n\}$ is the set of nodes, with the nodes representing cities, and A the set of edges. An edge represents a pair of cities between which travel is possible. Every edge has an associated cost. The TSP

consists of finding the cheapest tour that visits all cities exactly once. The Asymmetric Travelling Salesman Problem (ATSP) differs from the TSP simply by being defined on a directed graph. In this case, the edges of the graph represent *ordered* pairs of cities between which travel is possible. The oriented edges are usually called arcs.

Burke et al. [5, 6] propose a local search method based on a *hyperopt neighbourhood*, which we will exemplify using its example application to the ATSP. Given a feasible tour of the ATSP [6], a *hyperedge* is a subpath of the current tour, in other words, a sequence of successive arcs of the tour. Let i be 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 arcs contained. Let t be a feasible tour for the ATSP, considered to start and end at node 1. We introduce a relation of order $\prec_t$ on the set of nodes V as follows: for any $i, j \in V$, we say that $i \prec_t j$ if i is a predecessor of j on the tour, i.e. $t = (1, \ldots, i, \ldots, j, \ldots, 1)$.

Let t be a feasible tour of the ATSP, and $\mathcal{H}(i_1, i_{k+1})$ and $\mathcal{H}(j_1, j_{k+1})$ two hyperedges of length k such that $i_{k+1} \prec_t j_1$ and $\mathcal{H}(i_1, i_{k+1}) \cap \mathcal{H}(j_1, j_{k+1}) = \emptyset$ with respect to the nodes contained. It is obvious that the tour t can be described completely by the four hyperedges $\mathcal{H}(i_1, i_{k+1})$, $\mathcal{H}(i_{k+1}, j_1)$, $\mathcal{H}(j_1, j_{k+1})$, $\mathcal{H}(j_{k+1}, i_1)$. Burke et al. define a *k-hyperopt move* as being a new tour obtained after performing two steps: first, remove $\mathcal{H}(i_1, i_{k+1})$ and $\mathcal{H}(j_1, j_{k+1})$ from the tour t, then add arcs to $\mathcal{H}(i_{k+1}, j_1)$ and $\mathcal{H}(j_{k+1}, i_1)$ such that a new feasible tour is constructed. The set of all k-hyperopt moves is called a *k-hyperopt neighbourbood*.

Obviously, the size of the k-hyperopt neighbourhood increases exponentially with k. Due to the large size of the neighbourhood, instead of exploring the neighbourhood in a classical manner, Burke et al. propose an "optimal" construction of a k-hyperopt move by solving exactly a subproblem: the ATSP defined on the graph $G = (V', A')$, 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 A' the set of arcs in A that have both ends in V'. However, this approach is bound to be efficient only when k is relatively small. Otherwise, a large size ATSP would have to be solved as a subproblem.

In [6] Burke et al. provide numerical results for $k = 3$ and $k = 4$. They mention that they solve the subproblems to optimality using a dynamic programming algorithm, however they do not provide any further details. In [5] Burke et al. consider only the cases when $k = 2$ and $k = 3$ and use enumeration to solve the subproblems.

For a given hyperedge $\mathcal{H}(i_1, i_{k+1})$ the k-hyperopt move to be performed is determined in [5, 6] by evaluating the best k-hyperopt move over the set of *all* possible hyperopt moves. Therefore every hyperedge that does not intersect with $\mathcal{H}(i_1, i_{k+1})$ is considered, and for every such hyperedge a subproblem is solved. 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. The authors improve their method by using a tabu list that bans some hyperedges from being considered, based on the observation that only the hyperedges that have been affected by previous moves can provide improving hypermoves.

Burke et al. propose several methods that use the concept of k-hyperopt moves in [6]. They involve either a combination of hyperopt with 3-opt moves or the use of a local search based on hyperopt moves inside of a variable neighbourhood search

procedure [12]. A numerical comparison between the pure k-hyperopt approach and methods developed on top of it show the latter to be better. Although the k-hyperopt approach appears not to be fully competitive with other approaches for the ATSP [**?**], we believe that the k-hyperopt approach with further enhancements is a promising direction of research which deserves to be further investigated.

Very Large Scale Neighbourhoods. A similar idea was developed for the class of set partitioning problems. Thompson et al. [24, 25] defined the concept of a cyclic exchange for a local search approach, which transfers single elements between several subsets, in a "cyclic" manner. A two-exchange move can be seen as a cyclic exchange of length 2. Thompson et al. showed that for any current solution of a partitioning problem a new graph can be constructed. Costs are associated with its arcs and the set of nodes is split into subsets according to a partition induced by the current solution of the partitioning problem. Finding a cycle that uses at most one node for each subset in the new graph is equivalent to determining a cyclic exchange move for the original problem. Exact and heuristic methods [2, 8] that solve the problem of finding the most negative cost subset disjoint cycle (which corresponds to the best *improving* neighbour of the current solution) have been developed, however no work has been yet done on the integration of the exact algorithms within the local search framework.

Dynasearch. Dynasearch is another example where exponentially large neighbourhood are explored. The neighbourhood searched consists of all possible combinations of mutually independent simple search steps and one dynasearch move consists of a set of independent moves that are executed in parallel in a single local search iteration. Independence in the context of dynasearch means that the individual moves do not interfere with each other; this means that the gain incurred by a combined move must be the sum of the gains of the individual moves. In the case of independent moves, a dynamic programming algorithm is used to find the best combination of independent moves. So far, Dynasearch has only been applied to problems, where solutions are represented as permutations; current applications include the TSP [**?**], the single machine total weighted tardiness problem (SMTWTP) [**?**,**?**], and the linear ordering problem (LOP) [**?**]. Very good performance is reported when dynasearch is the local search routine inside an iterated local search (Section 4) or guided local search algorithm [27].

4 Iterated local search

A successful metaheuristic approach for solving difficult combinatorial problems is Iterated Local Search (ILS), which consists of running a local search procedure many times to perturbations of previously seen local optima. An outline of a simple ILS algorithm is given in Algorithm 1.

Algorithm 1 *Iterated Local Search*

Step 0: Let S be an initial solution.
Step 1: **while** `stopping_criterion` is not met **do**
(i) Let S' be the tour obtained from S after a perturbation.
(ii) Call `local_search`(S') to produce the tour S''.
(iii) **if** S'' is better than S **then** $S = S''$.

enddo

Step 2: Return S.

ILS can be applied in many different ways by, for example, changing the manner in which information about good solutions is collected (Step 1 (iii)), or the way the perturbation is performed. In the rest of this section we will show how different types of subproblems arise when these variations of the ILS are considered.

4.1 Collected Information

The first method focuses on the way the information is collected. An approximate method is run several times and the best solutions obtained are recorded. A subproblem is then constructed using the information stored and solved by an exact algorithm. The solution of the subproblem will be a solution of the original problem. We mention that this idea can be applied to a large number of problems. We illustrate this method by two examples.

Tour Merging. The paper we present next is the report of Applegate, Bixby, Chvàtal, and Cook [4], which is dedicated to the problem of finding near-optimal solutions of the TSP. Applegate et al. propose the use of an exact algorithm to solve a TSP subproblem defined by the outcome of multiple runs of a particular ILS algorithm applied to the original TSP. Applegate et al. propose an approach that makes use of the information obtained after running ILS several times. Their method is based on the observations that (i) high quality tours typically have many edges in common and, in fact, the number of shared edges increases with tour quality and (ii) that very high quality tours share many edges with optimal tours. Their method, called *tour merging*, is a two step procedure.

The first step consists of running an ILS like Chained Lin-Kernighan (CLK) [17, 4] or Iterated Helsgaun (IH) [**?**] a number of times, keeping the best solution obtained at each of the runs (both, CLK and IH are currently among the top performing algorithms for the TSP). In what follows we will denote the set of solutions recorded by $\mathcal{T}$. The second step consists of solving the TSP on the graph induced by this set of tours $\mathcal{T}$ 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 collected. 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 $A' = \{a \in A : \exists t \in \mathcal{T}, a \in t\}$.

It is hoped that this graph will be sparse with low *branch-width* [19] and also that it will be a graph that contains at least one (near-)optimal tour for the original TSP. Clearly, the TSP subproblem will be easier to solve than the original one. The quality of the tours obtained after running Chained Lin-Kernighan has a direct impact on the cardinality of the set of edges of the reduced graph. If the reduced graph is not very sparse, then the TSP subproblem may be computationally expensive to solve. Note that in tour merging, the subproblem generated is of the same type as the original problem. Here, the advantage of solving a subproblem comes from the reduced size of the subproblem, and therefore a natural direction of research is finding strategies that generate small subproblems of good quality.

In [4] two possibilities for the second step of the method are considered. The first is to use a general TSP optimisation code to find the optimal tour with respect to the graph G'. However, experimental results in [4] suggest that alternative methods could provide solutions much quicker. In particular, they suggest finding (heuristically) a branch-width decomposition of the graph G' and exploiting this decomposition by find optimal tours via a dynamic programming algorithm.

The tour merging approach was tested on a large number of TSPLIB benchmark problems [4]. The results obtained were very good, optimal solutions being identified for all problem tested. However the computational time was large.

Heuristic Concentration. A similar idea is put forward by Rosing et al. in [20] for the p-median problem. Given p, a given positive integer, and a graph that has weights associated with every node and distances associated with every arc, 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*. Rosing et al. propose repeatedly running a heuristic, with different random starts, while collecting the solutions obtained at each iteration. In a second step, they propose obtaining a solution for the p-median problem by solving a subproblem, in fact a restricted p-median problem. This method is called *Heuristic Concentration* (HC). The second step of HC starts with a subset of the best solutions found in the first phase. 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 into the concentration set, is solved.

Rosing et al. use CPLEX to solve the LP relaxation of the binary integer model of the reduced p-median problem. In most cases the solution obtained is integer; if not integer, it is easy to find using branch-and-bound. Numerical results are provided in both [20] and subsequent papers [21, 22].

We have seen that the two methods presented so far in this section act on the way information is collected at each iteration of the ILS. Then, *after* running the ILS, an exact algorithm is used to solve a subproblem determined by that information. Therefore an exact algorithm is applied only once, to exactly one problem. Next, we will talk about a paper that proposes several methods that generate many subproblems.

4.2 Modifying the Perturbation Step

The role of the perturbation in an ILS approach is to make large changes in the current solution and therefore allow the local search to leave local optima, while still conserving the good characteristics of a current solution. A good perturbation step is problem dependent, and should therefore be object of investigation. An idea of obtaining good perturbations for the ILS is defining a subproblem, usually of similar type with the original problem, and solve it using an exact algorithm or a good heuristic.

An example of determining a perturbation by using exact algorithms that solve a subproblem is the paper of Lourenço [15] given in the context of the job-shop scheduling problem.

The job-shop scheduling problem is defined for machines and jobs. Each job consists of a sequence of operations, each with a given processing time, that have to be

performed in a given order on different machines. The goal is to minimise the completion time of the last job subject to precedence constraints on the operations and additional capacity constraints saying that no machine can work on two operations at the same time. The job-shop scheduling problem can be modelled using the well-known disjunctive graph model [23].

Lourenço describes several variations of the ILS and provides numerical results for all of them. Firstly, she tests several methods of finding an initial feasible schedule. She then proposes the use of local improvement and simulated annealing as local search procedures. Finally, for the perturbation step, several perturbation ideas are put forward and tested for each of the two local search procedures. Since this is where the subproblems we are interested in are generated, we will discuss in some detail how the perturbation is performed.

The first perturbation procedure proposed by Lourenço involves the modification of the disjunctive 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 in the disjunctive graph. Then Carlier's algorithm [7] (a branch-and-bound method) is applied to one of the machines and the one-machine scheduling problem is solved. 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. The edges corresponding to that machine are then oriented according to the optimal solution obtained. The same treatment is then applied to the second machine. Lourenço mentions that this perturbation idea can create cycles in the disjunctive graph, and suggests a way of obtaining a feasible schedule from the graph with cycles (see [15] 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 [14] as an exact method. In order to do that preemption is allowed and two one-machine with lags on a chain are solved. Lourenço also gives a simple technique for eliminating cycles. We note that in this case too the subproblems solved are of a different type compared to the original problem.

The paper we described in this section is an example of how the perturbation can be determined such that a significant modification of the current solution is performed, while the main characteristics of the current solution are conserved. This is accomplished by modifying only part of the current solution by solving to optimality a subproblem of a similar type with the original problem.

5 Lower bounds

Another combination of exact and approximate methods is the usage of lower or upper bounds obtained from the application of an exact algorithm to a subproblem. Obtaining good bounds is important, since the bounds are used by the search algorithm to determine next moves or eliminate possible moves. The example we chose to illustrate this technique is the Ant Colony Optimisation.

Ant Colony. Ant Colony Optimisation (ACO) [**?**] is a recent metaheuristic approach for solving hard combinatorial optimisation problems, which is loosely inspired by the

pheromone trail laying and following behavior of real ants. Artificial ants in ACO are stochastic solution construction procedures that probabilistically build a solution by iteratively adding solution components to partial solutions by taking into account (i) heuristic information on the problem instance being solved, if available, and (ii) (artificial) pheromone trails which change dynamically at run-time to reflect the agents' acquired search experience. Of the available ACO algorithms (see [**?**] for an overview), the **A**pproximate **N**ondeterministic **T**ree **S**earch (ANTS) algorithm [16] is of particular interest for us here. Since ANTS was first applied to the quadratic assignment problem (QAP), we present the essential part of the algorithm using this example application.

The QAP can best be described as the problem of assigning a set of objects to a set of locations with given distances between the locations and given flows between the objects. The objective 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, two $n \times n$ matrices $A = [a_{ij}]$ and $B = [b_{rs}]$, where a_{ij} is the distance between locations i and j and b_{rs} is 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. Then the problem can be formulated as:

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

subject to the standard assignment constraints

$$\sum_{i=1}^{n} x_{ij} = 1,\ j = 1, .., n; \quad \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, .., n; \quad x_{ij} \in \{0, 1\} \quad i, j = 1, .., n$$

When applied to the QAP, in ANTS each ant constructs a solution by iteratively assigning objects to a free location. Given a location j, an ant assigns a still unassigned object i to this location with a probability that is proportional to $\alpha \cdot \tau_{ij}(t) + (1 - \alpha) \cdot \eta_{ij}$, where $\tau_{ij}(t)$ is the pheromone trail associated to the assignment of object i to a location j (pheromone trails give the "learned" desirability of choosing an assignment), η_{ij} is the heuristic desirability of this assignment, and α is a weighting factor between pheromone and heuristic. Lower bound computations are exploited at various places in ANTS. Before starting the actual solution process, ANTS first computes the Gilmore-Lawler lower bound (GLB) [10, 13], which amounts to solving a linear assignment problem by solving its linear programming relaxation. Along with the lower bound computation one gets the values of the dual variables $u_i, i = 1, \ldots, n$ and $v_i, i = 1, \ldots, n$ corresponding to the assignment constraints. 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 is assumed to be the location's impact on the QAP solution cost and, hence, the earlier it is tried to assign an object to that location. The main idea of ANTS is to use at each construction step lower bound computations to define the heuristic information of the attractiveness of a adding a specific assignment of object i to location j. This is achieved by tentatively adding the specific assignment (i, j) to the current partial solution and by estimating the cost of a complete solution containing that assignment by means of a lower bound. This estimate is used as the heuristic information η_{ij} during

the solution construction: the lower the estimate the more attractive is the addition of a specific assignment. Using lower bounds computations also presents several additional advantages like the elimination of possible moves if the cost estimation is larger than the so far best found solution. Additionally, tight bounds give a strong indication of how good a move is. However, the lower bound is to be computed at each construction step; hence, the lower bounds should be efficiently computable. Therefore, Maniezzo did not use GLB during the ants' construction steps, but exploits the weaker LBD lower bound, which can be computed in $\mathcal{O}(n)$. For details on the lower bound computation we refer to [16]. Experimental results have shown that ANTS is currently one of the best available algorithms for the QAP. The good performance of ANTS algorithm has also been confirmed in a variety of further applications.

6 Conclusions

The main conclusion of our paper is that there are many research opportunities to develop algorithms that integrate local search and exact techniques and that not much has been done so far in this area. We have presented a number of approaches that use both exact and local search methods in a rather complex way, and that, in our oppinion, can be further improved and extended to a number of different applications than the ones for which they have originally been developed.

Acknowledgments 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-2001-01419. 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, 1997.
2. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Inc. Englewood Cliffs, NJ, 1993.
3. D. Applegate, R. Bixby, V. Chvátal, and W. Cook. On the solution of traveling salesman problem. *Documenta Mathematica*, Extra Volume ICM III:645–656, 1998.
4. 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.
5. 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.
6. 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.
7. J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11:42–47, 1982.

8. I. Dumitrescu. *Constrained Shortest Path and Cycle Problems*. PhD thesis, The University of Melbourne, 2002. http://www.intellektik.informatik.tu-darmstadt.de/ irina.
9. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco, CA, 1979.
10. P. C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the SIAM*, 10:305–313, 1962.
11. F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, MA, 1997.
12. 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.
13. E. L Lawler. The quadratic assignment problem. *Management Science*, 9:586–599, 1963.
14. 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.
15. 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.
16. V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, 11(4):358–369, 1999.
17. O. Martin, S.W. Otto, and E.W. Felten. Combining simulated annealing with local search heuristics. *Annals of Operations Research*, 63:57–75, 1996.
18. G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.
19. N. Robertson and P.D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory*, 52:153–190, 1991.
20. K.E. Rosing and C.S. ReVelle. Heuristic concentration: Two stage solution construction. *European Journal of Operational Research*, pages 955–961, 1997.
21. 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.
22. K.E. Rossing. Heuristic concentration: a study of stage one. *ENVIRON PLANN B*, 27(1):137–150, 2000.
23. B. Roy and B. Sussmann. Les problemes d'ordonnancement avec constraintes disjonctives. Notes DS no. 9 bis, SEMA.
24. P.M. Thompson and J.B. Orlin. The theory of cycle transfers. Working Paper No. OR 200-89, 1989.
25. P.M. Thompson and H.N. Psaraftis. Cyclic transfer algorithm for multivehicle routing and scheduling problems. *Operations Research*, 41:935–946, 1993.
26. 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.
27. C. Voudouris. *Guided Local Search for Combinatorial Optimization Problems*. PhD thesis, Department of Computer Science, University of Essex, Colchester, UK, 1997.