ELSEVIER Discrete Applied Mathematics 123 (2002) 75–102

DISCRETE APPLIED MATHEMATICS

A survey of very large-scale neighborhood search techniques

Ravindra K. Ahuja[a,*], Özlem Ergun[b], James B. Orlin[c], Abraham P. Punnen[d]

[a]*Department of Industrial & Systems Engineering, University of Florida, P.O. Box 116595, Gainesville, FL 32611, USA*
[b]*School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA*
[c]*Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139, USA*
[d]*Department of Mathematics, Statistics and Computer Science, University of New Brunswick, Saint John, N.B., Canada E2L 4L5*

Received 9 September 1999; received in revised form 3 November 2000; accepted 8 January 2001

Abstract

Many optimization problems of practical interest are computationally intractable. Therefore, a practical approach for solving such problems is to employ heuristic (approximation) algorithms that can find nearly optimal solutions within a reasonable amount of computation time. An *improvement algorithm* is a heuristic algorithm that generally starts with a feasible solution and iteratively tries to obtain a better solution. Neighborhood search algorithms (alternatively called *local search algorithms*) are a wide class of improvement algorithms where at each iteration an improving solution is found by searching the "neighborhood" of the current solution. A critical issue in the design of a neighborhood search algorithm is the choice of the neighborhood structure, that is, the manner in which the neighborhood is defined. As a rule of thumb, the larger the neighborhood, the better is the quality of the locally optimal solutions, and the greater is the accuracy of the final solution that is obtained. At the same time, the larger the neighborhood, the longer it takes to search the neighborhood at each iteration. For this reason, a larger neighborhood does not necessarily produce a more effective heuristic unless one can search the larger neighborhood in a very efficient manner. This paper concentrates on neighborhood search algorithms where the size of the neighborhood is "*very large*" with respect to the size of the input data and in which the neighborhood is searched in an efficient manner. We survey three broad classes of very large-scale neighborhood search (VLSN) algorithms: (1) variable-depth methods in which large neighborhoods are searched heuristically, (2) large neighborhoods in which the neighborhoods are searched using network flow techniques or dynamic

*Corresponding author.
E-mail addresses: ahuja@ufl.edu (R.K. Ahuja), ozlem.ergun@isye.gatech.edu (Ö. Ergun), jorlin@mit.edu (J.B. Orlin), punnen@unbsj.ca (A.P. Punnen).

PII: S0166-218X(01)00338-9

programming, and (3) large neighborhoods induced by restrictions of the original problem that are solvable in polynomial time.

1. Introduction

Many optimization problems of practical interest are computationally intractable. Therefore, a practical approach for solving such problems is to employ heuristic (approximation) algorithms that can find nearly optimal solutions within a reasonable amount of computational time. The literature devoted to heuristic algorithms often distinguishes between two broad classes: constructive algorithms and improvement algorithms. A constructive algorithm builds a solution from scratch by assigning values to one or more decision variables at a time. An improvement algorithm generally starts with a feasible solution and iteratively tries to obtain a better solution. Neighborhood search algorithms (alternatively called *local search algorithms*) are a wide class of improvement algorithms where at each iteration an improving solution is found by searching the "neighborhood" of the current solution. This paper concentrates on neighborhood search algorithms where the size of the neighborhood is "*very large*" with respect to the size of the input data. For large problem instances, it is impractical to search these neighborhoods explicitly, and one must either search a small portion of the neighborhood or else develop efficient algorithms for searching the neighborhood implicitly.

A critical issue in the design of a neighborhood search approach is the choice of the neighborhood structure, that is, the manner in which the neighborhood is defined. This choice largely determines whether the neighborhood search will develop solutions that are highly accurate or whether they will develop solutions with very poor local optima. As a rule of thumb, the larger the neighborhood, the better is the quality of the locally optimal solutions, and the greater is the accuracy of the final solution that is obtained. At the same time, the larger the neighborhood, the longer it takes to search the neighborhood at each iteration. Since one generally performs many runs of a neighborhood search algorithm with different starting points, longer execution times per iteration lead to fewer runs per unit time. For this reason a larger neighborhood does not necessarily produce a more effective heuristic unless one can search the larger neighborhood in a very efficient manner.

Some very successful and widely used methods in operations research can be viewed as very large-scale neighborhood search techniques. For example, if the simplex algorithm for solving linear programs is viewed as a neighborhood search algorithm, then column generation is a very large-scale neighborhood search method. Also, the augmentation techniques used for solving many network flows problems can be categorized as very large-scale neighborhood search methods. The negative cost cycle canceling algorithm for solving the min cost flow problem and the augmenting path algorithm for solving matching problems are two such examples.

In this survey, we categorize very large-scale neighborhood methods into three possibly overlapping classes. The first category of neighborhood search algorithms we study are variable-depth methods. These algorithms focus on exponentially large neighborhoods and partially search these neighborhoods using heuristics. The second category contains network flow based improvement algorithms. These neighborhood search methods use network flow techniques to identify improving neighbors. Finally, in the third category we discuss neighborhoods for NP-hard problems induced by subclasses or restrictions of the problems that are solvable in polynomial time. Although we introduced the concept of large-scale neighborhood search by mentioning column generation techniques for linear programs and augmentation techniques for network flows, we will not address linear programs again. Rather, our survey will focus on applying very large-scale neighborhood search techniques to NP-hard optimization problems.

This paper is organized as follows. In Section 2, we give a brief overview of local search. We discuss variable-depth methods in Section 3. Very large-scale neighborhood search algorithms based on network flow techniques are considered in Section 4. In Section 5, efficiently solvable special cases of NP-hard combinatorial optimization problems and very large-scale neighborhoods based on these special cases are presented. We describe neighborhood metrics that might be a guide to the performance of local search algorithms with respect to the given neighborhoods in Section 6. Finally, in Section 7 we discuss the computational performance of some of the algorithms mentioned in the earlier sections.

2. Local search: an overview

We first formally introduce a combinatorial optimization problem and the concept of a neighborhood. There are alternative ways of representing combinatorial optimization problems, all relying on some method for representing the set of feasible solutions. Here, we will let the set of *feasible solutions* be represented as subsets of a finite set. We formalize this as follows:

Let $E=\{1,2,\dots,m\}$ be a finite set. In general, for a set S, we let $|S|$ denote its cardinality. Let $F\subseteq 2^E$, where 2^E denotes the set of all the subsets of E. The elements of F are called *feasible solutions*. Let $f : F\to\mathfrak{R}$. The function f is called the *objective function*. Then an instance of a *combinatorial optimization problem* (COP) is represented as follows:

$$\text{Minimize } \{f(S)\text{: } S\in F\}.$$

We assume that the family F is not given explicitly by listing all its elements; instead, it is represented in a compact form of size polynomial in m. An instance of a combinatorial optimization problem is denoted by the pair (F,f). For most of the problems we consider, the cost function is linear, that is, there is a vector $f_1,f_2,\dots,f_m$ such that for all feasible sets S, $f(S)=\sum_{i\in S} f_i$.

Suppose that (F,f) is an instance of a combinatorial optimization problem. A *neighborhood function* is a point to set map $N : F\to 2^E$. Under this function, each $S\in F$ has an associated subset $N(S)$ of E. The set $N(S)$ is called the *neighborhood* of the solution S, and we assume without loss of generality that $S \in N(S)$. A solution $S^* \in F$ is said to be *locally optimal* with respect to a neighborhood function N if $f(S^*) \leqslant f(S)$ for all $S \in N(S^*)$. The neighborhood $N(S)$ is said to be *exponential* if $|N(S)|$ grows exponentially in m as m increases. Throughout most of this survey, we will address exponential size neighborhoods, but we will also consider neighborhoods that are too large to search explicitly in practice. For example, a neighborhood with m^3 elements is too large to search in practice if m is large (say greater than a million).

We will refer to neighborhood search techniques using such neighborhoods as *very large-scale neighborhood search algorithms* or VLSN search algorithms.

For two solutions S and T, we let $S - T$ denote the set of elements that appear in S but not in T. We define the *distance* $d(S,T)$ as $|S - T| + |T - S|$, that is, the number of elements of E that appear in S or T but not both. Occasionally, we will permit neighborhoods to include infeasible solutions as well. For example, for the TSP we may permit the neighborhood of a tour to include each of the paths obtained by deleting an edge of the tour. To emphasize that the neighborhood contains more than tours, we normally give a combinatorial description of the non-feasible solutions permitted in the search. We refer to these non-feasible combinatorial structures as reference structures. For example, a Hamiltonian path may be a reference structure.

A neighborhood search algorithm (for a cost minimization problem) can be conceptualized as consisting of three parts:

(i) A neighborhood graph NG defined with respect to a specific problem instance, where NG is a directed graph with one node for each feasible solution (and/or instance of a non-feasible reference structure) created, and with an arc (S,T) whenever $T \in N(S)$.
(ii) A method for searching the neighborhood graph at each iteration.
(iii) A method for determining what is the next node of the neighborhood graph that the search in Step (ii) will choose. We will refer to this node as the *BaseSolution*.

The algorithm terminates when S is a locally optimal solution with respect to the given neighborhood. (See [1] for an extensive survey.)

We next define two neighborhoods based on the distance. The first neighborhood is $N_k(S) = \{T \in F\colon d(S,T) \leqslant k\}$. We will refer to this neighborhood as the *distance-k neighborhood.*

For some problem instances, any two feasible solutions have the same cardinality. This is true for the traveling salesman problem (TSP), where each feasible solution S represents a tour in a complete graph on n cities, and therefore has n arcs (see [46] for details on the TSP). In general, we say that T can be obtained by a single *exchange* from S if $|S-T|=|T-S|=1$; we say T can be obtained by a *k-exchange* if $|T-S|=|S-T|=k$. We define the *k-exchange neighborhood* of S to be $\{T\colon |S-T|=|T-S| \leqslant k\}$. If any two feasible solutions have the same cardinality, then the k-exchange neighborhood of S is equal to $N_{2k}(S)$. A standard example for the k-exchange neighborhood for the TSP is the 2-exchange neighborhood, also called the 2-opt neighborhood. Each node in the 2-opt neighborhood graph for the TSP is a tour, and two tours are neighbors if one can be obtained from the other by a 2-exchange. The method for searching the neighborhood is exhaustive (or some shortcut), and the next BaseSolution will be an improving solution.

Since $N_m(S)=F$, it follows that searching the distance-k neighborhood can be difficult as k grows large. It is typically the case that this neighborhood grows exponentially if k is not fixed, and that finding the best solution (or even an improved solution) in the neighborhood is NP-hard if the original problem is NP-hard.

3. Variable-depth methods

For $k=1$ or 2, the k-exchange (or similarly k-distance) neighborhoods can often be efficiently searched, but on average the resulting local optima may be poor. For larger values of k, the k-exchange neighborhoods yield better local optima but the effort spent to search the neighborhood might be too large. Variable-depth search methods are techniques that search the k-exchange neighborhood partially. The goal in this partial search is to find solutions that are close in objective function value to the global optima while dramatically reducing the time to search the neighborhood. Typically, they do not guarantee to be local optima. In VLSN search algorithms, we are interested in several types of algorithms for searching a portion of the k-exchange neighborhood. In this section, we describe the Lin–Kernighan [48] algorithm for the traveling salesman problem as well as other variable-depth heuristics for searching the k-exchange neighborhood for different combinatorial optimization problems. In the next section, we describe other approaches that in polynomial time implicitly search an exponential size subset of the k-exchange neighborhood when k is not fixed.

Before describing the Lin–Kernighan approach, we introduce some notation. Subsequently, we will show how to generalize the Lin–Kernighan approach to variable-depth methods (and ejection chains) for heuristically solving combinatorial optimization problems. Suppose that T and T' are both subsets of E, but not necessarily feasible. A *path* from T to T' is a sequence $T=T_1,\dots,T_K=T'$ such that $d(T_j,T_{j+1})=1$ for $j=1$ to $K-1$.

The variable-depth methods rely on a subroutine *Move* with the following features:

1. At each iteration, the subroutine Move creates a subset T_j and possibly also a feasible subset S_j from the input pair (S_{j-1},T_{j-1}) according to some search criteria. The subset T_j may or may not be feasible. We represent this operation as $\text{Move}(S_{j-1},T_{j-1})=(S_j,T_j)$
2. $d(T_j,T_{j+1})=1$ for all $j=1$ to $K-1$.
3. T_j typically satisfies additional properties, depending on the variable-depth approach.

Let T be the current TSP tour and assume without loss of generality that T visits the cities in order $1,2,3,\dots,n,1$. A *2-exchange neighborhood* for T can be defined as replacing two edges (i,j), and (k,l) by two other edges (i,k) and (j,l) or (i,l) and (j,k) to form another tour T'. Note that $d(T,T')=4$. The 2-exchange neighborhood may be described more formally by applying 4 Move operations where the first Move deletes the edge (i,j) from the tour, the second Move inserts the edge (i,k), the third Move deletes the edge (k,l) and the last Move inserts the edge (j,l).

Let $G=(N,A)$ be an undirected graph on n nodes. Let $P=v_1,\dots,v_n$ be an n node Hamiltonian path of G. A *stem and cycle* (this terminology is introduced by Glover

[26]) is an n-arc spanning subgraph that can be obtained by adding an arc (i, j) to a Hamiltonian path, where i is an end node of the path. Note that if node i is an end node of the path, and if j is the other end node of the path, then the stem and cycle structure is a Hamiltonian cycle, or equivalently is a tour. If T is a path or a stem and cycle structure, we let $f(T)$ denote its total length.

The Lin–Kernighan heuristic allows the replacement of as many as n edges in moving from a tour S to a tour T, that is, $d(S, T)$ is equal to some arbitrary $k \leqslant 2n$. The algorithm starts by deleting an edge from the original tour T_1 constructing a Hamiltonian path T_2. Henceforth one of the end points of T_2 is fixed and stays fixed until the end of the iteration. The other end point is selected to initiate the search. The even moves insert an edge into the Hamiltonian path T_{2j} incident to the end point that is not fixed to obtain a stem and cycle T_{2j+1}. The odd moves in the iteration delete an edge from the current stem and cycle T_{2j-1} to obtain a Hamiltonian path T_{2j}. From each Hamiltonian path T_{2j}, one implicitly constructs a feasible tour S_{2j} by joining the two end nodes. At the end of the Lin–Kernighan iteration we obtain the new BaseSolution tour S_i such that $f(S_i) \leqslant f(S_{2j})$ for all j.

Now we will describe the steps of the Lin–Kernighan algorithm in more detail. During an even move the edge to be added is the minimum length edge incident to the unfixed end point, and this is added to the Hamiltonian path T_{2j} if and only if $f(S) - f(T_{2j+1}) > 0$. Lin–Kernighan [48] also describe a look-ahead refinement to the way this edge is chosen. The choice for the edge to be added to the Hamiltonian path T_{2j} is made by trying to maximize $f(T_{2j}) - f(T_{2j+2})$. On the other hand, the edges to be deleted during the odd moves are uniquely determined by the stem and cycle structure, T_{2j-1} created in the previous move so that T_{2j} will be a Hamiltonian path. Additional restrictions may be considered when choosing the edges to be added. Researchers have considered different combinations of restrictions such as an edge previously deleted cannot be added again or an edge previously added cannot be deleted again in a later move.

Finally, the Lin–Kernighan algorithm terminates with a local optimum when no improving tour can be constructed after considering all nodes as the original fixed node.

We now illustrate the Lin–Kernighan's algorithm using a numerical example. Consider the tour on 10 nodes shown in Fig. 1(a). The algorithm first deletes the arc (1, 2) creating the Hamiltonian path shown in Fig. 1(b). Then the arc (2, 6) is added giving the stem and cycle illustrated in Fig. 1(c).

Deleting edge (6, 5) from this structure yields a Hamiltonian path and adding edge (5, 8), we get another stem and cycle. The edge insertion moves in the Lin–Kernighan heuristic are guided by a cost criterion based on 'cumulative gain' and the edge deletion moves are defined uniquely to generate the path structure. The Lin–Kernighan algorithm reaches a local optimum when after considering all nodes as the starting node no improving solutions can be produced.

There are several variations of the Lin–Kernighan algorithm that have produced high quality heuristic solutions. These algorithms use several enhancements such as 2-opt, 3-opt, and special 4-opt moves [29,39,40,48,49,50,62] to obtain a tour that cannot be constructed via the basic Lin–Kernighan moves. Also, efficient data structures are used to update the tours to achieve computational efficiency and solution quality [24,40].

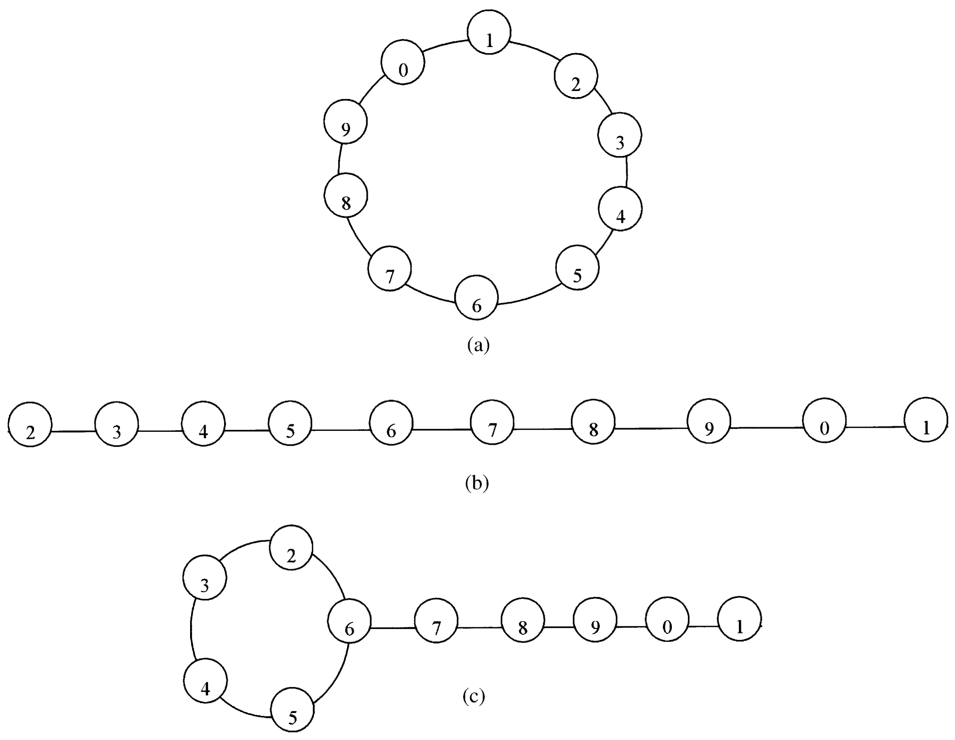

Fig. 1. Illustrating the Lin–Kernighan algorithm: (a) a tour on 10 nodes; (b) a Hamiltonian path; (c) a stem and cycle.

Papadimitriou [51] showed that the problem of determining a local optimum with respect to one version of the Lin–Kernighan algorithm is PLS-complete.

Now we can define the variable-depth methods for the TSP with the following procedure as a generalization of the Lin–Kernighan heuristic. This procedure takes as input a feasible tour S and then utilizes the function Move defined earlier. At each iteration the function Move creates the pair (T_j, S_j) where subset T_j is either a feasible solution or else an infeasible instance of a reference structure. The subset S_j is feasible. The function Move is called for r iterations where the value of r depends on an appropriate guidance rule. Finally, the procedure Variable-Depth-Search returns the feasible subset S_k that has the best objective function value found so far.

```
procedure Variable-Depth-Search($S$);
begin
  $S_1 := T_1 := S$;
  for $j := 2$ to $r$ do $(T_j, S_j) = \text{Move}(S_{j-1}, T_{j-1})$;
  select the set $S_k$ that minimizes $(f(S_j)$: $1 \leqslant j \leqslant r)$;
end;
```

This particular type of variable-depth search relies on a heuristic called “Move” that systematically creates a path of solutions starting from the initial solution. This framework is quite flexible, and there are a variety of ways of designing the procedure Move. The details of how Move is designed can be the difference between a successful heuristic and one that is not so successful.

In the procedure described above, we assumed that Move creates a single feasible solution at each stage. In fact, it is possible for move to create multiple feasible solutions at each stage [26,59] or no feasible solution at a stage [26,57].

Many variable-depth methods require the intermediate solutions T_j to satisfy certain topological (or structural) properties. For example, in the Lin–Kernighan algorithm we required that for each odd value of j, T_j is a stem and cycle. We also required that for each even value of j, T_j is a Hamiltonian path. We will also see examples in the next section where T_j satisfies additional properties that are not structural. For example, additional properties may depend on the ordering of the indices.

Glover [26] considered a structured class of variable-depth methods called *ejection chains* based on classical alternating path methods, extending and generalizing the ideas of Lin–Kernighan. Glover writes “In rough overview, an ejection chain is initiated by selecting a set of elements to undergo a change of state (e.g., to occupy new positions and or receive new values). The result of this change leads to identifying a collection of other sets, with the property that the elements of at least one must be “ejected from” their current state. State change steps and ejection steps typically alternate, and the options for each depend on the cumulative effect of previous steps (usually, but not necessarily, being influenced by the step immediately preceding). In some cases, a cascading sequence of operations may be representing a domino effect. *The ejection chain terminology is intended to be suggestive rather than restrictive*, *providing a unifying thread that links a collection of useful procedures for exploiting structure*, *without establishing a narrow membership that excludes other forms of classification*”.

In this paper, we will use the following more restrictive definition of ejection chains. We refer to the variable-depth method as an *ejection chain* if

(i) $|T_1| = |T_3| = |T_5| = \cdots = n$, and

(ii) $|T_2| = |T_4| = |T_6| = \cdots = n + 1$ (or $n - 1$).

For each even value of j, if $|T_j| = |S| - 1$, then T_j was obtained from T_{j-1} by ejecting an element. Otherwise, $|T_j| = |S| + 1$, and T_{j+1} is obtained from T_j by ejecting an element. Many of the variable-depth methods developed in the literature may be viewed as ejection chains. Typically these methods involve the construction of different reference structures along with a set of rules to obtain several different feasible solutions from them. To our knowledge, all the variable-depth methods for the traveling salesman problem considered in the literature may be viewed as ejection chains.

We can envision the nodes of the neighborhood graph with respect to the Lin–Kernighan neighborhood to consist of paths and stem-and-cycles. (Recall that tours are also instances of stem-and-cycles.) The endpoints of each edge of the neighborhood graph would link a path to a stem and cycle. The search technique would be the one proposed by Lin and Kernighan [42], and the selection procedure would be to select the best of the tours discovered along the way. Thus the reference structures described in ejection chains would be the nodes of the neighborhood graph in the ejection chain techniques. A technique in which the next BaseSolution is much more than a distance one from the current BaseSolution is a "variable-depth method". An ejection chain is a variable-depth method in which neighbors have the property that one is a subset of the other (and thus an element has been ejected in going from the larger to the smaller).

In the method described above, variable-depth methods rely on the function Move. One can also create exponential size subsets of N_k that are searched using network flows. In these neighborhoods, any neighbor can also be reached by a sequence of Moves in an appropriately defined neighborhood graph. Note that for the Lin–Kernighan algorithm the neighborhood size is polynomial, and it is the search that leads to finding solutions that are much different than the BaseSolution. We describe these network flows based techniques in the next section. Some of these techniques can also be viewed as ejection chain techniques if there is a natural way of associating an ejection chain (an alternating sequence of additions and deletions) with elements of the neighborhood [26,27,57,22].

Variable-depth and ejection chain based algorithms have been successfully applied in getting good solutions for a variety of combinatorial optimization problems. Glover [26], Rego [59], Zachariasen and Dum [81], Johnson and McGeoch [40], Mak and Morton [49], Pesch and Glover [53] considered such algorithms for the TSP. The vehicle routing problem is studied by Rego and Roucairol [61] and Rego [60]. Clustering algorithms using ejection chains are suggested by Dondorf and Pesch [13]. Variable-depth methods for the generalized assignment problem have been considered by Yagiura et al. [77] and ejection chain variations are considered by Yagiura et al. [76]. In addition, short ejection chain algorithms are applied to the multilevel generalized assignment problem by Laguna et al. [45]. These techniques are also applied to the uniform graph partitioning problem [16,20,42,52], categorized assignment problem [3], channel assignment problem [17], and nurse scheduling [14]. Sourd [68] apply a very general class of large neighborhood improvement procedures where the distance between two neighbors is variable to scheduling tasks on unrelated machines. These neighborhoods are developed by generating partial but still large enumeration trees based on the current solutions and searched heuristically.

4. Network flows based improvement algorithms

In this section, we study local improvement algorithms where the neighborhoods are searched using network flow based algorithms. The network flow techniques used to identify improving neighbors can be grouped into three categories: (i) minimum cost cycle finding methods; (ii) shortest path or dynamic programming based methods; and (iii) methods based on finding minimum cost assignments and matchings. The neighborhoods defined by cycles may be viewed as generalizations of 2-exchange neighborhoods. Neighborhoods based on assignments may be viewed as generalizations of insertion-based neighborhoods. In the following three subsections, we give general definitions of these exponential neighborhoods and describe the network flow algorithms used for finding an improving neighbor. For many problems, one determines an improving neighbor by applying a network flow algorithm to a related graph, which we refer to as an *improvement graph.*

4.1. Neighborhoods defined by cycles

In this subsection, we first define a generic partitioning problem. We then define the 2-exchange neighborhood and the cyclic exchange neighborhood.

Let $A=\{a_1,a_2,a_3,\dots,a_n\}$ be a set of n elements. The collection $\{S_1,S_2,S_3,\dots,S_K\}$ defines *a K-partition of A* if each set S_j is non-empty, the sets are pairwise disjoint, and their union is A. For any subset S of A, let $d[S]$ denote the cost of S. Then the set partitioning problem is to find a partition of A into at most K subsets so as to minimize $\Sigma_k d[S_k]$.

Let $\{S_1,S_2,S_3,\dots,S_K\}$ be any feasible partition. We say that $\{T_1,T_2,T_3,\dots,T_K\}$ is a *2-neighbor* of $\{S_1,S_2,S_3,\dots,S_K\}$ if it can be obtained by swapping two elements that are in different subsets. The 2-exchange neighborhood of $\{S_1,S_2,S_3,\dots,S_K\}$ consists of all 2-neighbors of $\{S_1,S_2,S_3,\dots,S_K\}$. We say that $\{T_1,T_2,T_3,\dots,T_K\}$ is a *cyclic-neighbor* of $\{S_1,S_2,S_3,\dots,S_K\}$ if it can be obtained by transferring single elements among a sequence of $k \leqslant K$ subsets in S. Let $(S_h^1,S_m^2,S_n^3,\dots,S_p^k)$ be such a sequence of k subsets, then we also require that $h=p$, that is the last subset of the sequence is identical to S_h^1. We refer to the transferring of elements as a *cyclic exchange.* We illustrate a cyclic exchange using Fig. 2. In this example, node 9 is transferred from subset S_1 to subset S_2. Node 2 is transferred from subset S_2 to subset S_3. Node 3 is transferred from subset S_3 to S_3. Finally, the cycle exchange is completed by transferring node 14 from subset S_3 to subset S_4. Finally, the exchange is completed by transferring node 14 from subset S_4 to subset S_1. One may also define a *path neighbor* in an analogous way. From a mathematical perspective, it is easy to transform a path exchange into a cyclic exchange by adding appropriate dummy nodes.

In general, the number of cyclic neighbors is substantially greater than the number of 2-neighbors. Whereas there are $O(n^2)$ 2-neighbors, for fixed value of K, there are $O(n^K)$ cyclic neighbors. If K is allowed to vary with n, there may be an exponential number of cyclic neighbors.

Thompson [73], Thompson and Orlin [74], and Thompson and Psaraftis [75] show how to find an improving neighbor in the cyclic exchange neighborhood by finding a negative cost “subset-disjoint” cycle in an improvement graph. Here we will describe how to construct the improvement graph. Let $A=\{a_1,a_2,\dots,a_n\}$ be the set of elements for the original set partitioning problem and let $S[i]$ denote the subset containing element a_i. The improvement graph is a graph $G=(V,E)$ where $V=\{1,2,\dots,n\}$ is a set of nodes corresponding to the indices of the elements of A of the original problem. Let $E=\{(i,j)\colon S[i]\neq S[j]\}$, where an arc (i,j) corresponds to the transfer of node i from $S[i]$ to $S[j]$ and the removal of j from $S[j]$. For each arc $(i,j)\in E$, we let $c[i,j]=d[\{i\}\cup S[j]\setminus\{j\}]-d[S[j]]$, that is, the increase in the cost of $S[j]$ when i is added to the set and j is deleted. We say that a cycle W in G is *subset-disjoint* if for every pair i and j of nodes of W, $S[i]\neq S[j]$, that is, the elements of A corresponding to the nodes of W are all in different subsets. There is a one-to-one cost-preserving correspondence between cyclic exchanges for the partitioning problem and subset-disjoint

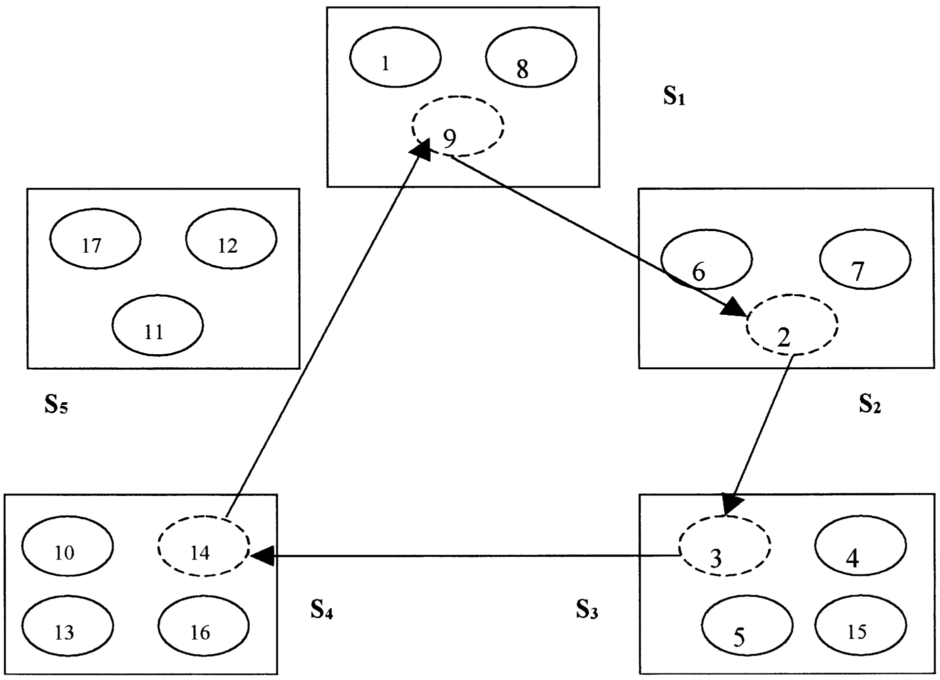

Fig. 2. Illustrating a cyclic exchange.

cycles in the improvement graph. In particular, for every negative cost cyclic exchange, there is a negative cost subset-disjoint cycle in the improvement graph. Unfortunately, the problem of determining whether there is a subset-disjoint cycle in the improvement graph is NP-complete, and the problem of finding a negative cost subset-disjoint cycle is NP-hard. (See, for example, Thompson [73], Thompson and Orlin [74], and Thompson and Psaraftis [75].)

Even though the problem of determining a negative cost subset-disjoint cycle in the improvement graph is NP-hard, there are effective heuristics for searching the graph. (See, for example, Thompson and Psaraftis [75] and Ahuja et al. [2].)

The cyclic exchange neighborhood search is successfully applied to several combinatorial optimization problems that can be characterized as specific partitioning problems. Thompson and Psaraftis [75], Gendreau et al. [25], and Fahrion and Wrede [19] solve the vehicle routing problem with the cyclic exchange neighborhood search. Frangioni et al. [23] apply cyclic exchanges to minimum makespan machine scheduling. Thompson and Psaraftis [75] also demonstrate its application to some scheduling problems. Ahuja et al. [2] developed the best available solutions for a widely used set of benchmark instances for the capacitated minimum spanning tree problems using cyclic exchanges.

The idea of finding improving solutions by determining negative cost cycles in improvement graphs has been used in several other contexts. Talluri [72] identifies cost saving exchanges of equipment type between flight legs for the daily airline fleet assignment problem by finding negative cost cycles in a related network. The fleet assignment problem can be modeled as an integer multicommodity flow problem subject to side constraints where each commodity refers to a fleet type. Talluri considers a given solution as restricted to two fleet types only, and looks for improvements that can be obtained by swapping a number of flights between the two fleet types. He develops an associated improvement graph and shows that improving neighbors correspond to negative cost cycles in the improvement graph. Schneur and Orlin [66] and Rockafellar [63] solve the linear multicommodity flow problem by iteratively detecting and sending flows around negative cost cycles. Their technique readily extends to a cycle-based improvement heuristic for the integer multicommodity flow problem. Wayne in [79] gives a cycle canceling algorithm for solving the generalized minimum cost flow problem. Firla et al. [21] introduce an improvement graph for the intersection of any two integer programs. Paths and cycles in this network correspond to candidates for improving feasible solutions. Furthermore, this network gives rise to an algorithmic characterization of the weighted bipartite b-matching problem. The algorithms discussed by Glover and Punnen [30] and Yeo [78] construct traveling salesman tours that are better than an exponential number of tours. These algorithms can also be viewed as computing a minimum cost cycle in an implicitly considered special layered network. We will discuss this heuristic in more detail in Section 5.

4.2. Neighborhoods defined by paths (or dynamic programming)

We will discuss three different types of neighborhood search algorithms based on shortest paths or dynamic programming. We discuss these approaches in the context of the traveling salesman problem. We can view these neighborhood search approaches when applied to the TSP as: (i) adding and deleting edges sequentially, (ii) accepting in parallel multiple swaps where a swap is defined by interchanging the current order of two cities in a tour, and (iii) cyclic shifts on the current tour. We next discuss these neighborhoods in more detail.

4.2.1. Creating a new neighbor by adding and deleting arcs sequentially

We first discuss a class of shortest path based methods that consider neighbors obtained by alternately adding and deleting edges from the current tour. These methods exhaustively search a subset of the ejection chain neighborhood of Section 3 with additional restrictions on the edges to be added. We assume for simplicity that the tour S visits the cities in the order $1,2,3,\dots,n,1$. Utilizing the terminology introduced in Section 3, let tour T be a k-exchange neighbor of S and let the path from S to T be the sequence $S=T_1,\dots,T_K=T$. These neighborhoods correspond to trial solutions created by odd and even paths described in Punnen and Glover [57] among several other neighborhoods and trial solutions constructed from different kinds of path structures. The trial solution generated by odd paths was developed independently by Firla et al. [22]. To illustrate, this type of trial solution generated by either odd or even paths

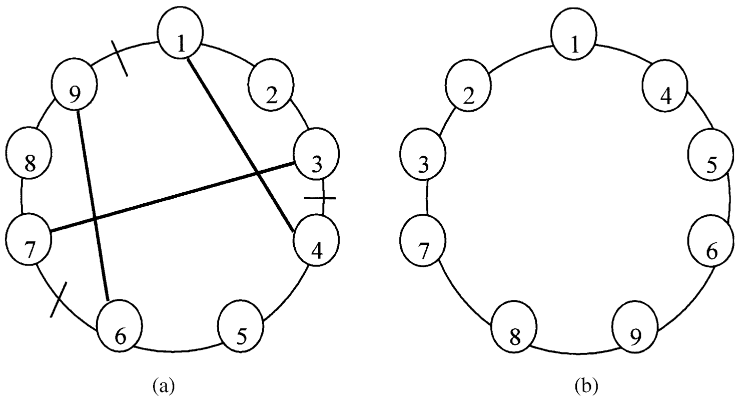

Fig. 3. Illustrating the alternate path exchange.

consider the following algorithm:

(i) Drop the edge $(n, 1)$ to obtain a Hamiltonian path T_2 and add an edge $(1, i)$ from node 1 to node i (where $i > 2$) to obtain a stem and cycle structure T_3.

(ii) The current terminal node of the path is node i. Drop edge $(i, i-1)$ and add edge $(i-1, j)$ for $i < j < n$, creating a Hamiltonian path first and then a stem and cycle structure.

(iii) Check if a termination criterion is reached. If yes, go to Step (iv). If no, let $i = j$ and go to step (ii).

(iv) Drop edge $(j, j-1)$ and add the final edge $(j-1, n)$ to complete the tour.

Fig. 3 illustrates this process on a 9 node tour $S = (1, 2, \ldots, 9, 1)$. The path exchange procedure initiates by deleting edge $(n, 1)$ and adding edge $(1, 4)$. Then edge $(3, 4)$ is dropped and edge $(3, 7)$ is added. Finally, the new tour T is created by deleting edge $(6, 7)$ and adding edge $(6, 9)$. In Fig. 3(a), we represent the edges that are added by bold lines and the edges that are dropped by a dash on the edge. Fig. 3(b) illustrates the new tour obtained after the path exchange.

Firla et al. [22], Glover [26] and Punnen and Glover [57] show that an improving solution in this neighborhood can be found in $O(n^2)$ time by finding an odd or even length shortest path in an improvement graph. Here, we describe an improvement graph that can be constructed to identify the best neighbor of a tour by finding a shortest path with even or odd number of nodes. Recall that $S = (1, 2, 3, \ldots, n, 1)$ is the current n node traveling salesman tour. The improvement graph is a graph $G = (V, E)$ where $V = \{1, 2, \ldots, n\}$ corresponding to nodes of the original problem, and $E = \{(i, j)\colon 1 = i < j - 1 < n\}$ is a set of directed arcs. Arcs $(1, j) \in E$ (such that $2 < j = n$) correspond to the deletion of edge $(n, 1)$ and the addition of edge $(1, j)$, and arcs $(i, j) \in E$ (such that $1 < i < j - 1 < n$) correspond to the deletion of edge $(i-1, i)$ and the addition of edge $(i-1, j)$ on the original tour S. If $d[i, j]$ is the cost of going from city i to j then we associate a cost $c[1, j] = -d[n, 1] + d[1, j]$ for each arc $(1, j) \in E$ such that $2 < j = n$ and a cost $c[i, j] = -d[i-1, i] + d[i-1, j]$ for each arc $(i, j) \in E$ such that

$1 < i < j-1 < n$. Finally, finding a negative cost path in G from node 1 to n identifies a profitable k-exchange.

In addition, trial solutions created from even and odd paths, new path structures such as broken paths and reverse paths leading to different trial solutions and reference structures are studied in [57]. The size of the neighborhood generated by even and odd paths alone is $\Omega(n2^n)$. Speed up techniques for searching neighborhoods is important even when the size of the neighborhood is not exponential. For example, using a shortest path algorithm on a directed acyclic improvement graph, Glover [27] obtained a class of best 4-opt moves in $O(n^2)$ time.

4.2.2. Creating a new neighbor by compounded swaps

The second class of local search algorithms defined by path exchanges is a generalization of the swap neighborhood. Given an n node traveling salesman tour $T = (1,2,3,\dots,n,1)$, the swap neighborhood generates solutions by interchanging the positions of nodes i and j for $1 \leqslant i < j \leqslant n$. For example letting $i = 3$ and $j = 6$, $T' = (1,2,6,4,5,3,7,\dots,n,1)$ is a neighbor of T under the swap operation. Two swap operations switching node i with j, and node k with l are said to be *independent* if $\max\{i,j\} < \min\{k,l\}$, or $\min\{i,j\} > \max\{k,l\}$. Then a large-scale neighborhood on the tour T can be defined by compounding (taking the union of) an arbitrary number of independent swap operations.

Congram et al. [9] and Potts and van de Velde [55] applied this compounded swap neighborhood to the single machine total weighted tardiness scheduling problem and the TSP, respectively. They refer to this approach as *dynasearch*. In their paper, Congram et al. [9] show that the size of the neighborhood is $O(2^{n-1})$ and give a dynamic programming recursion that finds the best neighbor in $O(n^3)$ time. Hurink [38] apply a special case of the compounded swap neighborhood where only adjacent pairs are allowed to switch in the context of one machine batching problems and show that an improving neighbor can be obtained in $O(n^2)$ time by finding a shortest path in the appropriate improvement graph.

Now we describe an improvement graph to aid in searching the compounded swap neighborhood. Let $T = (1,2,3,\dots,n,1)$ be an n node traveling salesman tour. The improvement graph is a graph $G = (V,E)$, where (i) $V = \{1,2,\dots,n,1',2',\dots,n'\}$ is a set of nodes corresponding to the nodes of the original problem and a copy of them, and (ii) E is a set of directed arcs $(i,j') \cup (j',k)$, where an arc (i,j') corresponds to the swap of the nodes i and j, and an arc (j',k) indicates that node k will be the first node of the next swap. For example, a path of three arcs $(i,j'),(j',k),(k,l')$ in G represents two swap operations switching node i with j, and node k with l. To construct the arc set E, every pair (i,j') and (j',k) of nodes in V is considered, and arc (i,j') is added to E if and only if $j > i > 1$. Arc (j',k) is added to E if and only if $j = 1$ and $k > j$ or $j > 1$ and $k > j+1$. For each arc $(i,j') \in E$, we associate a cost $c[i,j']$ that is equal to the net increase in the optimal cost of the TSP tour after deleting the edges $(i-1,i)$, $(i,i+1)$, $(j-1,j)$ and $(j,j+1)$ and adding the edges $(i-1,j)$, $(j,i+1)$, $(j-1,i)$ and $(i,j+1)$. In other words, if $d[i,j]$ is the cost of going from node i to node j in the original problem and $d[n,n+1] = d[n,1]$, then

$$c[i,j'] = (-d[i-1,i] - d[i,j] - d[j,j+1])$$

$$+(d[i-1,j] + d[j,i] + d[i,j+1])$$

$$\text{for } j' = i+1,$$

and

$$c[i,j'] = (-d[i-1,i] - d[i,i+1] - d[j-1,j] - d[j,j+1])$$
$$+ (d[i-1,j] + d[j,i+1] + d[j-1,i] + d[i,j+1]) \quad \text{for } j' > i+1.$$

The cost $c[j',k]$ of all edges (j',k) are set equal to 0.

Now finding the best neighbor of a TSP tour for the compounded swap neighborhood is equivalent to finding a shortest path on this improvement graph, and hence takes $O(n^2)$ time. Note that since TSP is a cyclic problem, one of the nodes is held fixed during the exchange. In the above construction of the improvement graph, without loss of generality, we assumed that node 1 is not allowed to move, and hence the neighborhood is searched by finding a shortest path from node $1'$ to either node n or n'. The dynamic programming recursion given in Congram et al. [9] for searching the neighborhood will also take $O(n^2)$ time when applied to the TSP. The shortest path algorithm given above takes $O(n^3)$ time when applied to the total weighted tardiness scheduling problem because it takes $O(n^3)$ time to compute the arc costs.

4.2.3. *Creating a new neighbor by a cyclical shift*

The final class of local search algorithms in this section is based on a kind of cyclic shift of pyramidal tours [8]. A tour is called *pyramidal* if it starts in city 1, then visits cities in increasing order until it reaches city n, and finally returns through the remaining cities in decreasing order back to city 1. Let $T(i)$ represent the city in the ith position of tour T. A tour T' is a *pyramidal neighbor* of a tour T if there exists an integer p such that:

(i) $0 \leqslant p \leqslant n$,
(ii) $T'(1) = T(i_1), T'(2) = T(i_2), \ldots, T'(p) = T(i_p)$ with $i_1 < i_2 < \cdots < i_p$ and
(iii) $T'(p+1) = T(j_1), T'(p+2) = T(j_2), \ldots, T'(n) = T(j_{n-p})$ with $j_1 > j_2 > \cdots > j_{n-p}$.

For example, if tour $T = (1,2,3,4,5,1)$ then tour $T' = (1,3,5,4,2,1)$ is a pyramidal neighbor. Note that a drawback of this neighborhood is that edges (1,2) and $(1,n)$ belong to all tours. To avoid this Carlier and Villon [8] consider the n rotations associated with a given tour. The size of this neighborhood is $\theta(n2^{n-1})$ and it can be searched in $O(n^3)$ time using n iterations of a shortest path algorithm in the improvement graph.

Now, we describe an improvement graph where for the TSP a best pyramidal neighbor can be found by solving for a shortest path on this graph. Let $T = (1,2,3,\ldots,n,1)$ be an n node traveling salesman tour. The improvement graph is a graph $G = (V,E)$ where (i) $V = \{1,2,\ldots,n,1',2',\ldots,n'\}$ corresponding to the nodes of the original problem and a copy of them, and (ii) E is a set of directed arcs $(i,j') \cup (j',k)$, where an arc (i,j') corresponds to having the nodes i to j in a consecutive order, and an arc

(j',k) corresponds to skipping the nodes $j+1$ to $k-1$ and appending them in reverse order to the end of the tour. To construct the arc set E, every pair (i,j') and (j',k) of nodes in V is considered. Arc (i,j') is added to E if and only if $i \leqslant j$, and arc (j',k) is added to E if and only if $j < k+1$. For each arc $(i,j') \in E$, we associate a cost $c[i,j']$ that is equal to the net increase in the cost of the pyramidal neighbor of the TSP tour after adding the edge $(j+1, i-1,)$ when the tour is visiting the previously skipped cities in reverse order. For each arc $(j',k) \in E$, we associate a cost $c[j',k]$ that is equal to the net increase in the optimal cost of the TSP tour after adding the edge (j,k) and deleting the edges $(j,j+1)$ and $(k-1,k)$. In other words, if $d[i,j]$ is the cost of going from city i to j in the original problem, then for $i<j$ and $j<n-1$

$$c[i,j'] = d[j+1, i-1],$$

$$c[j',k] = -d[j,j+1] - d[k-l,k] + d[k-1,k].$$

Note that some extra care must be taken when calculating the cost of the edges that have 1, $1'$, n, and n' as one of the end points. Now the neighborhood can be searched by finding a shortest path from node 1 to either node n or n'. Carlier and Villon [8] also show that if a tour is a local optimum for the above neighborhood, then it is a local optimum for the 2-exchange neighborhood.

In addition to these three classes of neighborhoods, dynamic programming has been used to determine optimal solutions for some special cases of the traveling salesman problem. Simonetti and Balas [67] solve the TSP with time windows under certain kinds of precedence constraints with a dynamic programming approach. Burkard et al. [7] prove that over a set of special structured tours that can be represented via PQ-trees, the TSP can be solved in polynomial time using a dynamic programming approach. They also show that the set of pyramidal tours can be represented by PQ-trees and there is an $O(n^2)$ algorithm for computing the shortest pyramidal tour. These results will be discussed in more detail in Section 5.

4.3. Neighborhoods defined by assignments and matchings

In this section, we discuss an exponential neighborhood structure defined by finding minimum cost assignments in an improvement graph. We illustrate this neighborhood in the context of the traveling salesman problem. Also, we show that the assignment neighborhood can be generalized to a neighborhood defined by finding a minimum cost matching on a non-bipartite improvement graph. We demonstrate this generalization on the set partitioning problem.

The assignment neighborhood for the traveling salesman problem can be viewed as a generalization of the simple neighborhood defined by ejecting a node from the tour and reinserting it optimally. Given an n node tour $T=(1,2,3,\ldots,n,1)$, if the cost of going from city i to j is $d[i,j]$, then the first step in searching the assignment neighborhood is to create a bipartite improvement graph as follows:

(i) For some $k \leqslant \lfloor n/2 \rfloor$, choose and eject k nodes from the current tour T. Let the set of these ejected nodes be $V=\{v_1,v_2,\ldots,v_k\}$ and the set of remaining nodes be $U=\{u_1,u_2,\ldots,u_{n-k}\}$.

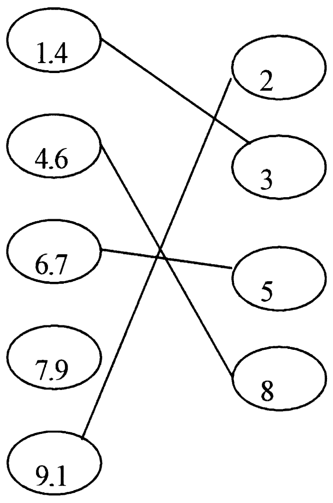

Fig. 4. Illustrating the matching neighborhood.

(ii) Construct a sub-tour $T' = (u_1, u_2, \ldots, u_{n-k}, u_1)$. Let q_i denote the edge for every (u_i, u_{i+1}) for $i = 1$ to $n - k - 1$, and let q_{n-k} denote the edge (u_{n-k}, u_1).

(iii) Now construct a complete bipartite graph $G = (N, N', E)$ such that $N = \{q_i\colon i = 1$ to $n - k\}$, $N' = V$, and the weight on each edge (q_i, v_j) is $c[q_i, v_j] = d[u_i, v_j] + d[v_j, u_{i+1}] - d[u_i, u_{i+1}]$.

A neighbor of T corresponds to a tour T^* obtained by inserting the nodes of V into the subtour T' with at most one node inserted between adjacent nodes of T'. The minimum cost assignment of k arcs corresponds to the minimum cost neighbor of T.

Using Fig. 4, we will demonstrate the assignment neighborhood on the 9 node tour $T = (1, 2, 3, 4, 5, 6, 7, 8, 9, 1)$. Let $V = \{2, 3, 5, 8\}$, then we can construct the subtour on the nodes in U as $T' = (1, 4, 6, 7, 9, 1)$. Fig. 4 illustrates the bipartite graph G with only the edges of the matching for simplicity. The new tour obtained is $T'' = (1, 3, 4, 8, 6, 5, 7, 9, 2, 1)$. Note that when $k = \lfloor n/2 \rfloor$, the size of the assignment neighborhood is equal to $\Omega(\lfloor n/2 \rfloor !)$.

The assignment neighborhood was first introduced in the context of the TSP by Sarvanov and Doroshko [64] for the case $k = n/2$ and n even. Gutin [31] gives a theoretical comparison of the assignment neighborhood search algorithm with the local steepest descent algorithms for $k = n/2$. Punnen [56] considered the general assignment neighborhood for arbitrary k and n. An extension of the neighborhood where paths instead of nodes are ejected and reinserted optimally by solving a minimum weight matching problem is also given in [56]. Gutin [32] shows that for certain values of k the neighborhood size can be maximized along with some low complexity algorithms searching related neighborhoods. Gutin and Yeo [35] constructed a neighborhood based on the assignment neighborhood and showed that moving from any tour T to another tour

T' using this neighborhood takes at most 4 steps. Deineko and Woeginger [12] study several exponential neighborhoods for the traveling salesman problem and the quadratic assignment problem based on assignments and matchings in bipartite graphs as well as neighborhoods based on partial orders, trees and other combinatorial structures. The matching based neighborhood heuristic is also applied to the inventory routing problem by Dror and Levy [15].

Another class of matching based neighborhoods can be obtained by packing subtours, where the subtours are generated by solving a bipartite minimum weight matching problem [41,58]. It is NP-hard to find the best such tour. Efficient heuristics can be used to search this neighborhood [28,41,58]. Neighborhood search algorithms using this neighborhood can be developed by utilizing cost modifications or other means to control the matching generated.

We next consider a neighborhood structure based on non-bipartite matchings. We will discuss this neighborhood in the context of the general set partitioning problem considered in Section 3. Let $S = \{S_1, S_2, S_3, \ldots, S_K\}$ be a partition of the set $A = \{a_1, a_2, a_3, \ldots, a_n\}$. Then construct a complete graph $G = (N, E)$ such that every node i for $1 \leqslant i \leqslant K$ represents the subset S_i in S. Now the weights $c[i,j]$ on the edge (i,j) of G can be constructed separately with respect to a variety of rules. One such rule can be as follows:

(i) Let the cost contribution of subset S_i to the partitioning problem be $d[S_i]$.
(ii) For each edge (i,j) in E, combine the elements in S_i and S_j, and repartition it into two subsets optimally. Let the new subsets be S_i' and S_j'.
(iii) Then $c[i,j] = (d[S_i'] + d[S_j']) - (d[S_i] + d[S_j])$.

Note that if the edges with non-negative weights are eliminated from G, then any negative cost matching on this graph will define a cost improving neighbor of S. Tailard apply these types of ideas to a general class of clustering problems in [70], and to the vehicle routing problem in [69].

5. Solvable special cases and related neighborhoods

There is a vast literature on efficiently solvable special cases of NP-hard combinatorial optimization problems. Of particular interest for our purposes are those special cases that can be obtained from the original NP-hard problem by restricting the problem topology, or by adding constraints to the original problem, or by a combination of these two factors. By basing neighborhoods on these efficiently solvable special cases, one can often develop exponential sized neighborhoods that may be searched in polynomial time. We point out that most of these techniques have not been tested experimentally and might yield poor local optima. Note that the cyclical shift neighborhood discussed in Section 4 is based on an $O(n^2)$ algorithm for finding the minimum cost pyramidal tour.

Our next illustration deals with Halin graphs. A *Halin graph* is a graph that may be obtained by embedding a tree that has no nodes of degree 2 in the plane, and joining the leaf nodes by a cycle so that the resulting graph is planar. Cornuejols et al. [10] gave an $O(n)$ algorithm for solving the traveling salesman problem on a Halin graph.

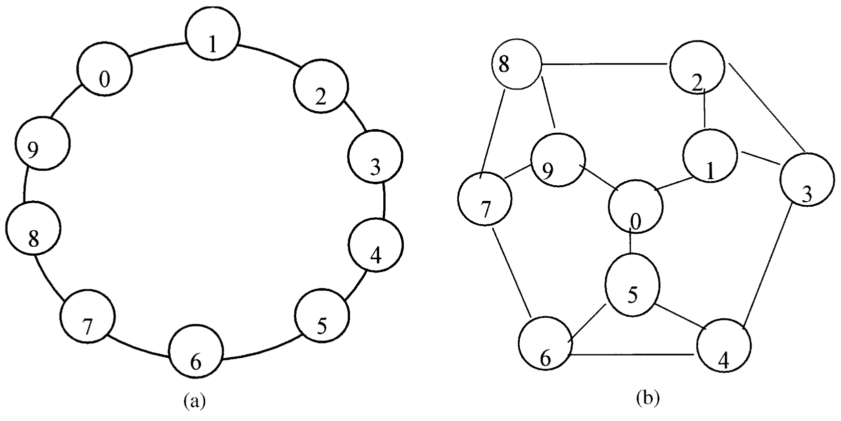

Fig. 5. A Halin graph: (a) a tour on 10 nodes; (b) a Halin extension.

Note that Halin graphs may have an exponential number of TSP tours, such as in the example of Fig. 3 (taken from [10]).

We now show how Halin graphs can be used to construct a very large neighborhood for the traveling salesman problem. Suppose that T is a tour. We say that H is a *Halin extension* of T if H is a Halin graph and if T is a subgraph of H. Fig. 5 illustrates a Halin extension for tour $T=(0,1,2,\ldots,9,0)$. Suppose that there is an efficient procedure HalinExtend(T), that creates some Halin extension of T. To create the neighborhood $N(T)$, one would let $H(T)=$ HalinExtend(T), and then let $N(T)=\{T'\colon T'$ is a tour in $H(T)\}$. To find the best tour in this neighborhood, one would find the best tour in $H(T)$. In principle, one could define a much larger neighborhood: $N(T)=\{T'\colon$ there exists a Halin extension of T containing in $T'\}$. Unfortunately, this neighborhood may be too difficult to search efficiently since it would involve simultaneously optimizing over all Halin extensions of T. Similar Halin graph based schemes for the bottleneck traveling salesman problem and the steiner tree problem could be developed using the linear time algorithm of Philips et al. [54], and Winter [80], respectively.

In our previous examples, we considered pyramidal tours, which may be viewed as the traveling salesman problem with additional constraints. We also considered the TSP as restricted to Halin graphs. In the next example, we consider (cf. [30]) a neighborhood that simultaneously relies on a restricted class of graphs plus additional side constraints. Glover and Punnen [30] identified the following class of tours among which the best member can be identified in linear time. Let $C_1, C_2, \ldots, C_k$ be k vertex-disjoint cycles each with at least three nodes, and such that each node is in one of the cycles. A *single edge ejection tour* is a tour T with the following properties:

1. $|T \cap C_i| = |C_i| - 1$ for $i=1$ to k, that is, T has $|C_i|-1$ arcs in common with C_i.
2. There is one arc of T directed from C_i to C_{i+1} for $i=1$ to $k-1$ and from C_k to C_1.

The number of ways of deleting arcs from the cycles is at least $\prod_i |C_i|$, which may be exponentially large, and so the number of single edge ejection tours is exponentially large. To find the optimum single edge ejection tour, one can solve a related shortest path problem on an improvement graph. The running time is linear in the number of arcs. Given a tour, one can delete $k+1$ arcs of the tour, creating k paths, and then transform these k paths into the union of k cycles described above. In principle, the above neighborhood search approach is easy to implement; however, the quality of the search technique is likely to be sensitive to the details of the implementation. Glover and Punnen [30] consider broader neighborhoods as well, including something that they refer to as "double ejection tours". They also provide an efficient algorithm for optimizing over this class of tours.

Yeo [78] considered another neighborhood for the asymmetric version of the TSP. His neighborhood is related to that of Glover and Punnen [30], but with dramatically increased size. He showed that the search time for this neighborhood is $O(n^3)$. Burkard and Deineko [6] identified another class of exponential neighborhood such that the best member can be identified in quadratic time. Each of these algorithms can be used to develop VLSN search algorithms. However, to the best of our knowledge, no such algorithms have been implemented yet.

We summarize the results of this section with a general method for turning a solution method for a restricted problem into a very large-scale neighborhood search technique. Let X be a class of NP-hard combinatorial optimization problem. Suppose that X' is a restriction of X that is solvable in polynomial time. Further suppose that for a particular instance (F,f) of X, and for every feasible subset S in F, there is a subroutine "CreateNeighborhood(S)" that creates a well-structured instance (F',f) of X' such that

1. S is an element of F'.
2. F' is a subset of F.
3. (F',f) is an instance of X'.

We refer to F' as the *X'-induced neighborhood* of S. The neighborhood search approach consists of calling CreateNeighborhood(S) at each iteration, and then optimizing over (F',f) using the polynomial time algorithm. Then S is replaced with the optimum of (F',f), and the algorithm is iterated. Of particular interest is the case when the subroutine CreateNeighborhood runs in polynomial time, and the size of F' is exponential. The neighborhood would not be created explicitly. While we believe that this approach has enormous potential in the context of neighborhood search, this potential is largely unrealized. Moreover, it is not clear in many situations how to realize this potential. For example, many NP-hard combinatorial optimization problems are solvable in polynomial time when restricted to series-parallel graphs. Examples include network reliability problem [65], optimum communication spanning tree problem [18], vertex cover problem [5,71], the feedback vertex set problem [5,71], etc. Similar results are obtained for job shop scheduling problems with specific precedence restrictions [44]. It is an interesting open question how to best exploit these efficient algorithms for series-parallel graphs in the context of neighborhood search.

6. Neighborhood metrics

In this section we describe neighborhood metrics that might be a guide as to the performance of local search algorithms with respect to the given neighborhoods. As mentioned earlier, a critical issue in the design of neighborhood search heuristics is the balance between the size of the neighborhood and the time required to search it. Hence an important neighborhood metric is the *size of the neighborhood*. In terms of the neighborhood graph, the size of the neighborhood for a given solution S can be viewed as the number of directed arcs leaving S, or equivalently the outdegree of S. For the variable-depth methods the neighborhood size is not necessarily exponential, and it is the search that leads to finding solutions that are much different than the BaseSolution. On the other hand, for network based approaches and methods induced by solvable cases the neighborhood size is typically exponential. (See Table 1)

Table 1 summarizes the size of the neighborhoods discussed previously for the TSP (this table is mostly taken from [7]):

Another neighborhood metric that is studied in the literature is the *diameter of the neighborhood graph*. The *distance* from a node S to a node T in the neighborhood graph is the length of a shortest path from S to T. The *diameter* of the neighborhood graph NG is the least positive integer d such that $d(S,T)=d$ for all nodes S and T of NG. Gutin and Yeo [35] construct polynomially searchable neighborhoods of exponential size for the TSP where the corresponding neighborhood graphs has diameter four; that is, for any pair of tours T_1 and T_5, there exists tours T_2, T_3, and T_4 such that $T_i \in N(T_{i-1})$ for all $i=2,3,4,5$. The neighborhood graph for the cyclical-shift based neighborhood considered by Carlier and Villon [8] has diameter $\theta(\log n)$.

For a given neighborhood graph, we say that $P{=}i_1, i_2, \ldots, i_K$ is *monotone* if the objective value $f(i_j) < f(i_{j-1})$ for $j{=}2$ to K. We let $d_m(S)$ denote the length of the shortest monotone path from S to a local optimum solution. If $d_m(S)$ is exponentially large for

Table 1

Neighborhood	Size	Log (size)	Time to search	Reference
2-Opt	$\Omega(n^2)$	$\Theta(\log n)$	$O(n^2)$	Croes [11]
k-Opt	$\Omega(n^k)$	$\Theta(\log n)$	$O(n^k)$	Lin [47]
Pyramidal	$\Omega(2^n)$	$\Theta(n)$	$O(n^2)$	Klyaus [43]
Cyclical shift	$\Omega(n2^n)$	$\Theta(n)$	$O(n^3)$	Carlier and Villion [8]
Edge ejection	$\Omega((12^n)^{1/3})$	$\Theta(n)$	$O(n)$	Glover and Punnen [30]
Shortest path based edge ejection	$\Omega(n2^n)$	$\Omega(n)$	$O(n^2)$	Punnen and Glover [57], Glover [26]
Cyclic-exchange (fixed-k subsets)	$\Theta(n^k)$	$\Theta(\log n)$	$O(n^2)$	Ahuja et al. [2]
Compound swaps	$\Theta(2^{n-1})$	$\Theta(n)$	$O(n^2)$	Potts and van de Velde [55]
Matching based[a]	$\Theta(n!/2)$	$\Theta(n \log n)$	$O(n^3)$	Sarvanov and Doroshko [64]
Halin graphs	$\Omega(2^n)$	$\Theta(n)$	$O(n)$	Cornuejols et al. [10]
PQ-Trees	$2^{\Theta(n \log \log n)}$	$\Theta(n \log \log n)$	$O(n^3)$	Burkard et al. [7]

[a]This assumes that $k = \lfloor n/2 \rfloor$, nodes are deleted. Better time bounds are available if fewer nodes are deleted, and the size of the neighborhood accordingly is decreased.

any S, this ensures that a neighborhood search approach starting at S will be exponentially long. In the converse direction, let $d_m^p(S,T)$ denote the longest monotone path from S to T. If $d_m^p(S,T)$ is guaranteed to be polynomial, than any neighborhood search technique based on this neighborhood will have a polynomial number of iterations.

Finally, we consider *domination analysis* of neighborhood search algorithms. The domination analysis of a heuristic analyzes the number of solutions that are 'dominated' by the solution produced. Let α be a heuristic algorithm for a combinatorial optimization problem which produces a solution S^* in F. The domination number of α, denoted by $\text{dom}(\alpha)$, is the cardinality of the set $F(S^*)$, where $F(S^*)=\{S\in F: f(S)\geqslant f(S^*)\}$. If $\text{dom}(\alpha)=|F|$, then S^* is an optimal solution. Domination analysis of various algorithms for the TSP has been studied in [28–30,33,34,36,58,56]

7. Computational performance of VLSN search algorithms

In this section, we briefly discuss the computational performance of some of the VLSN algorithms mentioned in the earlier sections. We first consider the traveling salesman problem. The Lin–Kernighan algorithm and its variants are widely believed to be the best heuristics for the TSP. In an extensive computational study, Johnson and McGeoch [40] substantiate this belief by providing a detailed comparative performance analysis. Rego [59] implemented a class of ejection chain algorithms for the TSP with very good experimental results indicating superiority of his method over the original Lin–Kernighan algorithm. Punnen and Glover [57] implemented a shortest path based ejection chain algorithm. The implementations of Rego [59] and Punnen and Glover [57] are relatively straightforward and use simple data structures.

Recently, Helsgaun [37] reported very impressive computational results based on a complex implementation of the Lin–Kernighan algorithm. Although this algorithm uses the core Lin–Kernighan variable-depth search, it differs from previous implementations in several key aspects. The superior computational result is achieved by efficient data handling, special 5-opt moves, new non-sequential moves, effective candidate lists, cost computations, effective use of upper bounds on element costs, information from the Held and Karp 1-tree algorithm and sensitivity analysis, among others. Helsgaun reports that his algorithm produced optimal solutions for all test problems for which an optimal solution is known, including the 7397 city problem and the 13,509 city problem considered by Applegate et al. [4]. Helsgaun estimated the average running time for his algorithm as $O(n^{2.2})$. To put this achievement in perspective, it may be noted that the 13,509 city problem solved to optimality by an exact branch and cut algorithm by Applegate et al. [4] used a cluster of three Digital Alpha 4100 servers (with 12 processors) and a cluster of 32 Pentium-II PCs and consumed three months of computation time. Helsgaun, on the other hand, used a 300 MHz G3 Power Macintosh. For the 85,900 city problem, pla85900 of the TSPLIB, Helsgaun obtained an improved solution using two weeks of CPU time. (We also note that the vast majority of time spent by Applegate et al. [4] is in proving that the optimal solution is indeed optimal.)

Table 2
Small TSP Problems: best % deviation from the optimum

Problem	REGO	HLK	MLK	SPG
bier127	0.12	0.00	0.42	0.10
u159	0.00	0.00	0.00	—
ch130	—	0.00	—	0.23
ch150	—	0.00	—	0.40
d198	0.30	0.00	0.53	0.33
d493	0.97	0.00	—	2.65
ei1101	0.00	0.00	0.00	0.79
fl417	0.78	0.00	—	0.41
gi1262	0.13	0.00	1.30	1.81
kroA150	0.00	0.00	0.00	0.00
kroA200	0.27	0.00	0.41	0.68
kroB150	0.02	0.00	0.01	0.07
kroB200	0.11	0.00	0.87	0.42
kroC100	0.00	0.00	0.00	0.00
kroD100	0.00	0.00	0.00	0.00
kroE100	0.00	0.00	0.21	0.02
lin105	0.00	0.00	0.00	0.00
lin318	0.00	0.00	0.57	1.03
pcb442	0.22	0.00	1.06	2.41
pr107	0.05	0.00	0.00	0.00
pr124	0.10	0.00	0.08	0.00
pr136	0.15	0.00	0.15	0.00
pr144	0.00	0.00	0.39	0.00
pr152	0.90	0.00	4.73	0.00
pr226	0.22	0.00	0.09	0.11
pr264	0.00	0.00	0.59	0.20
pr299	0.22	0.00	0.44	1.30
pr439	0.55	0.00	0.54	1.29
rd100	0.00	0.00	—	0.00
rd400	0.29	0.00	—	2.45
ts225	0.25	0.00	—	0.00
gr137	0.20	0.00	0.00	—
gr202	1.02	0.00	0.81	—
gr229	0.23	0.00	0.20	—
gr431	0.91	0.00	1.41	—

In Table 2, we summarize the performance of Rego's algorithm (REGO) [59], Helsgaun–Lin–Kernighan algorithm (HLK) [37], the modified Lin–Kernighan algorithm of Mak and Morton [49] and the shortest path algorithm of Punnen and Glover (SPG) [57] on small instances of the TSP problem. In the table we use the best case for each of these algorithms.

In Table 3, we summarize the performance of Rego's algorithm (REGO) [59], Helsgaun–Lin–Kernighan algorithm (HLK) [37], and the Lin–Kernighan implementation (JM-LK) of Johnson and McGeoch [40]. Reference source not found for large

Table 3
Large TSP: average % deviation over several runs

Problem	Rego	JM-LK	HLK
dsj1000	1.10	3.08	0.035
pr1002	0.86	2.61	0.00
pr2392	0.79	2.85	0.00
pcb3038	0.97	2.04	0.00
fl3795	7.16	8.41	—
fl4461	1.06	1.66	0.001
pla7397	1.57	2.19	0.001

instances of the TSP problem. In the table we use the average % deviation for each of these algorithms.

Next, we consider the capacitated minimum spanning tree problem. This is a special case of the partitioning problem discussed in Section 4. Exploiting the problem structure, Ahuja et al. [2] developed a VLSN search algorithm based on cyclic exchange neighborhood. This algorithm is highly efficient and obtained improved solutions for many benchmark problems. It currently has the best available solution for every instance listed in the set of benchmarks, accessible at http://www.ms.ic.ac.uk/info.html.

As our last example, we consider VLSN search algorithms for the generalized assignment problems (GAP). Yagiura et al. [77] developed an ejection chain based tabu search algorithm for the GAP. They report that in reasonable amount of computation time they obtained solutions that are superior or comparable with that of existing algorithms. Based on computational experiments and comparisons, it is reported that on benchmark instances the solutions produced by their algorithm are within 16% optimal.

Many of the references cited throughout this paper also report computational results based on their algorithms. For details we refer to these papers. Several neighborhoods of very large size discussed in the paper have not been tested experimentally within a VLSN search framework. Effective implementations of these and related neighborhoods are topics for further investigation.

Acknowledgements

The research of the first author was supported by the NSF Grant DMI-9900087. The second and third authors were in part supported by the NSF Grants DMI-9810359 and DMI-9820998. The fourth author was supported by NSERC Grant OPG0170381.

References

[1] E. Aarts, J.K. Lenstra, Local Search in Combinatorial Optimization, Wiley, New York, 1997.

[2] R.K. Ahuja, J.B. Orlin, D. Sharma, New neighborhood search structures for the capacitated minimum spanning tree problem, Research Report 99-2, Department of Industrial & Systems Engineering, University of Florida, 1999.
[3] V. Aggarwal, V.G. Tikekar, Lie-Fer Hsu, Bottleneck assignment problem under categorization, Comput. Oper. Res. 13 (1986) 11–26.
[4] D. Applegate, R. Bixby, V. Chvatal, W. Cook, On the solution of traveling salesman problems, Documenta Math. ICM (1998) 645–656.
[5] S. Arnborg, A. Proskurowski, Linear time algorithms for NP-hard problems restricted to partial k-trees, Discrete Appl. Math. 23 (1989) 11–24.
[6] R.E. Burkard, V.G. Deineko, Polynomially solvable cases of the traveling salesman problem and a new exponential neighborhood, Computing 54 (1995) 191–211.
[7] R.E. Burkard, V.G. Deineko, G.J. Woeginger, The travelling salesman problem and the PQ-tree, Mathematics of Operations Reserach 23 (1998) 613–623.
[8] J. Carlier, P. Villon, A new heuristic for the traveling salesman problem, RAIRO Oper. Res. 24 (1990) 245–253.
[9] R.K. Congram, C.N. Potts, S.L. van de Velde, An iterated dynasearch algorithm for the single machine total weighted tardiness scheduling problem, 1998, paper in preparation.
[10] G. Cornuejols, D. Naddef, W.R. Pulleyblank, Halin graphs and the traveling salesman problem, Math. Programming 26 (1983) 287–294.
[11] G.A. Croes, A method for solving traveling-salesman problems, Oper. Res. 6 (1958) 791–812.
[12] V. Deineko, G.J. Woeginger, A study of exponential neighborhoods for the traveling salesman problem and the quadratic assignment problem, Report Woe-05, Technical University Graz, 1997.
[13] U. Dorndorf, E. Pesch, Fast clustering algorithms, ORSA J. Comput. 6 (1994) 141–153.
[14] K.A. Dowsland, Nurse scheduling with tabu search and strategic oscillation, European J. Oper. Res. 106 (1998) 393–407.
[15] M. Dror, L. Levy, A vehicle routing improvement algorithm comparison of a “greedy” and a “matching” implementation for inventory routing, Comput. Oper. Res. 13 (1986) 33–45.
[16] A.E. Dunlop, B.W. Kernighan, A procedure for placement of standard cell VLSI circuits, IEEE Trans. Comput.-Aided Design 4 (1985) 92–98.
[17] M. Duque-Anton, Constructing efficient simulated annealing algorithms, Discrete Appl. Math. 77 (1997) 139–159.
[18] E.S. El-Mallah, C.J. Colbourn, Optimum communication spanning trees in series parallel networks, SIAM J. Comput. 14 (1985) 915–925.
[19] R. Fahrion, M. Wrede, On a principle of chain exchange for vehicle routing problems (I-VRP), J. Oper. Res. Soc. (1990) 821–827.
[20] C.M. Fiduccia, R.M. Mattheyses, A linear time heuristic for improving network partitions, in: ACM IEEE Nineteenth Design Automation Conference Proceedings, IEEE Computer Society, Los Alamitos, CA, 1982, pp. 175–181.
[21] R.T. Firla, B. Spille, R. Weismantel, A primal analogue of cutting plane algorithms, Department of Mathematics, Otto-von-Guericke-University Magdeburg, 1999.
[22] R.T. Firla, B. Spille, R. Weismantel, personal communication.
[23] A. Frangioni, E. Necciari, M.G. Scutella, Multi-exchange algorithms for the minimum makespan machine scheduling problem, Dipartimento di Informatica, University of Pisa, 2000, paper in preparation.
[24] M.L. Fredman, D.S. Johnson, L.A. McGeoch, Data structures for traveling salesman, J. Algorithms 16 (1995) 432–479.
[25] M. Gendreau, F. Guertin, J.Y. Potvin, R. Seguin, Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries, CRT-98-10, 1998.
[26] F. Glover, Ejection chains, reference structures, and alternating path algorithms for the traveling salesman problem, Research report, University of Colorado-Boulder, Graduate School of Business, 1992. {A short version appeared in Discrete Appl. Math. 65 (1996) 223–253.}
[27] F. Glover, Finding the best traveling salesman 4-opt move in the same time as a best 2-opt move, J. Heuristics 2 (1996) 169–179.
[28] F. Glover, G.M. Gutin, A. Yeo, Zverovich, Construction heuristics and domination analysis for the asymmetric TSP, Research Report, Brunel University, 1999.

[29] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Dordrecht, 1997.

[30] F. Glover, A.P. Punnen, The traveling salesman problem: new solvable cases and linkages with the development of approximation algorithms, J. Oper. Res. Soc. 48 (1997) 502–510.

[31] G.M. Gutin, On the efficiency of a local algorithm for solving the traveling salesman problem, Automat. Remote Control 11 (part 2) (1988) 1514–1519.

[32] G.M. Gutin, Exponential neighborhood local search for the traveling salesman problem, Comput. Oper. Res. 26 (1999) 313–320.

[33] G.M. Gutin, A. Yeo, Polynomial algorithms for the TSP and the QAP with a factorial domination number, Manuscript, Brunel University, UK, 1998.

[34] G.M. Gutin, A. Yeo, TSP heuristics with large domination number, Report 12/98, Department of Mathematics and Statistics, Brunel University, UK, 1998.

[35] G.M. Gutin, A. Yeo, Small diameter neighborhood graphs for the traveling salesman problem, Comput. Oper. Res. 26 (1999) 321–327.

[36] G.M. Gutin, A. Yeo, TSP tour domination and Hamiltonian cycle decomposition of regular digraphs, Manuscript, Brunel University, UK, 1999.

[37] K. Helsgaun, An effective implementation of the Lin–Kernighan traveling salesman heuristic, Manuscript, Roskilde University, Denmark, 1999.

[38] J. Hurink, An exponential neighborhood for a one machine batching problem, OR Spektrum 21 (1999) 461–476.

[39] D.S. Johnson, Local search and the traveling salesman problem, in: Proceedings of 17th International Colloquium on Automata Languages and Programming, Lecture Notes in Computer Science, Springer, Berlin, 1990, pp. 443–460.

[40] D.S. Johnson, L.A. McGeoch, The travelling salesman problem: a case study in local optimization, in: E.H.L. Aarts, J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization, Wiley, New York, 1997, pp. 215–310.

[41] R.M. Karp, A patching algorithm for the non-symmetric traveling salesman problem, SIAM J. Comput. 8 (1979) 561–573.

[42] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell System Tech. J. 49 (1970) 291–307.

[43] P.S. Klyaus, The structure of the optimal solution of certain classes of the traveling salesman problems, Vestsi Akad. Nauk BSSR, Phys. Math. Sci., Minsk, (1976) 95–98 (in Russian).

[44] S. Knust, Optimality conditions and exact neighborhoods for sequencing problems, Universitat Osnabruck, Fachbereich Mathematik/Informatik, Osnabruck, Germany, 1997.

[45] M Laguna, J. Kelly, J.L. Gonzales-Velarde, F. Glover, Tabu search for multilevel generalized assignment problem, European J. Oper. Res. 82 (1995) 176–189.

[46] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys, The Traveling Salesman Problem, Wiley, New York, 1985.

[47] S. Lin, Computer solutions to the traveling salesman problem, Bell System Tech. J. 44 (1965) 2245–2269.

[48] S. Lin, B. Kernighan, An effective heuristic algorithm for the traveling salesman problem, Oper. Res. 21 (1973) 498–516.

[49] K. Mak, A. Morton, A modified Lin–Kernighan traveling salesman heuristic, ORSA J. Comput. 13 (1992) 127–132.

[50] I.I. Melamed, S.I. Sergeev, I.K. Sigal, The traveling salesman problem: approximation algorithms, Avtomati Telemekh 11 (1989) 3–26.

[51] C.H. Papadimitriou, The complexity of Lin–Kernighan algorithm, SIAM J. Comput. (1992) 450–465.

[52] C.H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[53] E. Pesch, F. Glover, TSP ejection chains, Discrete Appl. Math. 76 (1997) 165–181.

[54] J.M. Phillips, A.P. Punnen, S.N. Kabadi, A linear time algorithm for the bottleneck traveling salesman problem on a Halin graph, Inform. Process. Lett. 67 (1998) 105–110.

[55] C.N. Potts, S.L. van de Velde, Dynasearch—Iterative local improvement by dynamic programming. part I. The traveling salesman problem, Technical Report, University of Twente, The Netherlands, 1995.

[56] A.P. Punnen, The traveling salesman problem: new polynomial approximation algorithms and domination analysis, J. Inform. Optim. Sci., to appear.
[57] A.P. Punnen, F. Glover, Ejection chains with combinatorial leverage for the TSP, Research Report, University of Colorado-Boulder, 1996.
[58] A.P. Punnen, S.N. Kabadi, Domination analysis of heuristics for the asymmetric traveling salesman problem, Manuscript, University of New Brunswick, 1998.
[59] C. Rego, Relaxed tours and path ejections for the traveling salesman problem, European J. Oper. Res. 106 (1998a) 522–538.
[60] C. Rego, A subpath ejection method for the vehicle routing problem, Management Sci. 44 (1998b) 1447–1459.
[61] C. Rego, C. Roucairol, A parallel tabu search algorithm using ejection chains for the vehicle routing problem, in: I.H. Osman, J.P. Kelly (Eds.), Meta-Heuristics: Theory and Applications, Kluwer Academic Publishers, Dordrecht, 1996.
[62] G. Reinelt, The traveling salesman computational solutions for TSP application, Lecture Notes in Computer Science, Vol. 840, Springer, Berlin, 1994.
[63] R.T. Rockafellar, Network Flows and Monotropic Optimization, Wiley, New York, 1984.
[64] V.I. Sarvanov, N.N. Doroshko, Approximate solution of the traveling salesman problem by a local algorithm with scanning neighborhoods of factorial cardinality in cubic time, Software: Algorithms and Programs, Vol. 31, Mathematics Institute of the Belorussia Academy of Science, Minsk, 1981, pp. 11–13 (in Russian).
[65] A. Satyanarayana, R.K. Wood, A linear time algorithm for computing k-terminal reliability in series parallel networks, SIAM J. Comput. 14 (1985) 818–832.
[66] R.R. Schneur, J.B. Orlin, A scaling algorithm for multicommodity flow problems, Oper. Res. 46 (1998).
[67] N. Simonetti, E. Balas, Implementation of a linear time algorithm for certain generalized traveling salesman problems, in: Proceedings of the IPCO V, Lecture Notes in Computer Science, Vol. 1084, Springer, Berlin, 1996, pp. 316–329.
[68] F. Sourd, Scheduling tasks on unrelated machines: large neighborhood improvement procedures, J. Heuristics, submitted for publication.
[69] E.D. Taillard, Parallel iterative search methods for vehicle routing problems, Network 23 (1993) 661–673.
[70] E.D. Taillard, Heuristic methods for large centroid clustering problems, Technical Report IDSIA-96-96, Lugano, 1996.
[71] T. Takamizawa, T. Nishizeki, N. Sato, Linear time computability of combinatorial problems on series-parallel graphs, J. ACM 29 (1982) 623–641.
[72] K.T. Talluri, Swapping applications in a daily airline fleet assignment, Transportation Sci. 30 (1996) 237–248.
[73] P.M. Thompson, Local search algorithms for vehicle routing and other combinatorial problems, Ph.D. Thesis, Operations Research Center, MIT, Cambridge, MA, 1988.
[74] P.M. Thompson, J.B. Orlin, The theory of cyclic transfers, Operations Research Center Working Paper, MIT, Cambridge MA, August 1989.
[75] P.M. Thompson, H.N. Psaraftis, Cyclic transfer algorithms for multivehicle routing and scheduling problems, Oper. Res. 41 (1993).
[76] M. Yagiura, T. Ibaraki, F. Glover, An ejection chain approach for the generalized assignment problem, Technical Report #99013, Department of Applied Mathematics and Physics, Kyoto University, 1999.
[77] M. Yagiura, T. Yamaguchi, T. Ibaraki, A variable-depth search algorithm for the generalized assignment problem, in: S. Voss, S. Martello, I.H. Osman (Eds.), Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, Boston, 1999, pp. 459–471.
[78] A. Yeo, Large exponential neighborhoods for the traveling salesman problem, Preprint no. 47, Department of Mathematics and Computer Science, Odense University, 1997.
[79] K. Wayne, A polynomial combinatorial algorithm for generalized minimum cost flow, STOC 1999.
[80] P. Winter, Steiner problem in Halin networks, Discrete Appl. Math. 17 (1987) 281–294.

[81] M. Zachariasen, M. Dam, Tabu search on the geometric traveling salesman problem, in: I.H. Osman, J.P. Kelly (Eds.), Meta-Heuristics: Theory and Applications, Kluwer Academic Publishers, Dordrecht, 1996.

For further reading

P.C. Gilmore, E.L. Lawler, D.B. Shmoys, Well-solved special cases, in: E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (Eds.), The Traveling Salesman Problem, Wiley, New York, 1985, pp. 87–143.

G.M. Gutin, On approach to solving the traveling salesman problem, in: Theory, Methodology, and Practice of System Research, Mathematical Methods of Systems Analysis, VNIIST, Moscow, 1984, pp. 184–186.