PERGAMON

Computers & Operations Research 26 (1999) 409–425

computers & operations research

Solving the k-best traveling salesman problem

Edo S. van der Poort[a,1], Marek Libura[b,2], Gerard Sierksma[c,3], Jack A. A. van der Veen[d,4]

[a]*Department of Marketing & Logistics, Agrotechnological Research Institute, Wageningen, The Netherlands*

[b]*Systems Research Institute, Polish Academy of Sciences, Poland*

[c]*Department of Econometrics, University of Groningen, The Netherlands*

[d]*Centre for Supply Chain Management, Nijenrode University, The Netherlands*

Received June 1997; received in revised form March 1998

Scope and purpose

After having solved an optimization problem, it is often the case that one still must verify whether the optimal solution satisfies some additional restrictions that are not included in the original model. Such conditions are called "subtle conditions". In the case of subtle conditions, it is particularly of interest to have a set of k-best solutions in order to see which of the solutions in that set satisfy the extra conditions and how high the "price" is for incorporating such conditions.

*Corresponding author. Agrotechnological Research Institute, Department of Marketing and Logistics, PO Box 17, NL-6700AA Wageningen, The Netherlands. Tel.: 0031 317 475169; e-mail: E.S. van.der.Poort@ato.dlo.nl.

[1]Edo van der Poort is a technical researcher at the department of Management & Logistics of the Agrotechnological Research Institute (ATO-DLO). He holds a M.S. in Computing Science and a Ph.D. in Economics (combinatorial optimization). His research interests include combinatorial optimization, warehousing, and supply chain management.

[2]Marek Libura is Associate Professor at the Systems Research Institute of the Polish Academy of Sciences, Warsaw. He received a Ph.D. degree in Operations Research from the Institute of Applied Cybernetics, Polish Academy of Sciences. His primary research area is Combinatorial Optimization and its applications. His papers have been published in several journals including Discrete Applied Mathematics, Control and Cybernetics, OR Spectrum, Group Decision and Negotiation.

[3]Gerard Sierksma is an Associate Professor of Operations Research at the University of Groningen. He holds a M.S. and a Ph.D. in Mathematics (Combinatorial Convexity). His primary research area is quantitative logistics. In 1997 his book on Linear and Integer Programming; Theory and Practice (Marcel Dekker, Inc.) was published.

[4]Jack A.A. van der Veen is Associate professor in the area of Production & Logistics Management at Nijenrode University and co-ordinator of Nijenrode's Centre for Supply Chain Management. He obtained a PhD in Economics from the University of Groningen, The Netherlands. His research interests include Scheduling, Combinatorial optimization and Supply chain optimization. His papers have been published in several journals including the European Journal of Operational Research, Discrete Applied Mathematics and Mathematical Programming.

0305-0548/99/$ – see front matter © 1999 Elsevier Science Ltd. All rights reserved.
PII: S0305-0548(98)00070-7

Algorithms for finding sets of k-best solutions have mainly been studied in the literature for polynomially solvable combinatorial optimization problems. Little is known on algorithms for finding k-best solutions for $\mathcal{NP}$-hard combinatorial optimization problems. In this paper, we study algorithms for finding k-best solutions for one of the most notorious NP-hard problems, namely the Traveling Salesman Problem (TSP). Properties of the sets of k-best solutions for the TSP, such as the increase in tour length and the number of edges that the k-best solutions have in common or differ in, are discussed in detail.

Abstract

Although k-best solutions for polynomial solvable problems are extensively studied in the literature, not much is known for $\mathcal{NP}$-hard problems. In this paper we design algorithms for finding sets of k-best solutions to the Traveling Salesman Problem (TSP) for some positive integer k. It will be shown that a set of k-best Hamiltonian tours in a weighted graph can be determined by applying the so-called partitioning algorithms and by algorithms based on modifications of solution methods like branch-and-bound. In order to study the effectiveness of these algorithms, the time for determining a set of k-best solutions is investigated for a number of instances in Reinelt's TSPLIB library. It appears that the time required to find a set of k-best tours grows rather slowly in k. Furthermore, the results of numerical experiments show that the difference in length between a longest and a shortest tour in the set of k-best solutions grows only slowly in k and that also the 'structure' of the tours in the set of k-best tours is quite robust. © 1999 Elsevier Science Ltd. All rights reserved.

1. Introduction

In this paper we consider the k-best problem for the well-known *Traveling Salesman Problem* (TSP). The so-called *k-best TSP* can be formulated as follows. Given a weighted graph and some positive integer k, determine a *set of k-best tours*, i.e. a set of k tours where the length of any other tour is at least equal to the length of the longest tour in the set.

In the literature, the k-best problem for general discrete optimization problems is considered in [1] and [2]. For a number of specific combinatorial optimization problems the k-best problem is also studied in the literature. For the Assignment Problem see e.g. [3], for the Shortest Path Problem see [4–6] for the Minimum Spanning Tree Problem see [7, 8] and for the Perfect Matching Problem see [9]. Surprisingly, little is known on the k-best problem for $\mathcal{NP}$-hard combinatorial optimization problems.

In this paper two types of algorithms are considered for determining sets of k-best tours, namely partitioning algorithms and algorithms based on a modification of solution methods like branch-and-bound. We will show how the partitioning algorithms for determining k-best solutions to general discrete optimization problems can be modified and applied to the TSP. An example of such a partitioning algorithm can be found in [1]. This algorithm is a generalization of the algorithms developed by Murty [3] and Yen [5] for solving the k-best problem for the Assignment and Shortest Path Problems, respectively. Another partitioning algorithm is formulated in [2]. This algorithm is a generalization of an algorithm by Gabow [7] for generating weighted spanning trees in order. In the modification of solution methods like branch-and-bound we follow the ideas of Piper and Zoltners [10] and Wilson and Jain [11] who showed how to modify solution methods in order to determine a set of k-best solutions for Zero–One Programming and Zero–One Goal Programming, respectively.

In order to investigate the effectiveness of both types of algorithms, we evaluate the running times of applying the algorithms to a number of instances of the TSPLIB library (see [12]). For these instances, we will also consider the difference in length between a longest and a shortest tour and the structure of the tours in the set of k-best tours as a function of k.

2. Definitions and basic results

For $n \geqslant 3$ and $1 \leqslant m \leqslant \binom{n}{2}$, consider the graph $G = (V, E)$ with vertex set $V = \{1, \dots, n\}$ and edge set $E = \{e_1, \dots, e_m\} \subseteq \{\{i, j\}: i, j \in V, i \neq j\}$. The length of edge e is a real number and is denoted by $d(e)$. The vector $d = [d(e_1), \dots, d(e_m)]^T \in \mathbb{R}^m$ is called the *vector of edge lengths* of the graph G. The pair (G, d) will be called a *weighted graph*. The length of an edge set S with respect to d is given by $L_d(S) := \sum_{e \in S} d(e)$. A *Hamiltonian tour* (a *tour* for short) in the graph G is a subset of E that forms a cycle containing each vertex in V exactly once. By $\mathscr{H}$ we denote the set of all tours in G. Throughout, it will be assumed that G is Hamiltonian, i.e. that $\mathscr{H} \neq \emptyset$. Using this notation, the TSP is the problem of finding a tour in $\operatorname{argmin}\{L_d(H): H \in \mathscr{H}\}$, with $\operatorname{argmin}\{\cdot\}$ denoting the set of optimal solutions for a minimization problem.

Let $1 \leqslant k \leqslant |\mathscr{H}|$ and define $\mathscr{H}(0) := \emptyset$. Any set $\mathscr{H}(k) = \{H_{(1)}, \dots, H_{(k)}\} \subseteq \mathscr{H}$ satisfying

$$L_d(H_{(1)}) \leqslant L_d(H_{(2)}) \leqslant \cdots \leqslant L_d(H_{(k)}) \leqslant L_d(H) \text{ for all } H \in \mathscr{H} \backslash \mathscr{H}(k)$$

is called a *set of k-best tours*. The *k-best TSP* is defined as the problem of finding a set $\mathscr{H}(k)$ in (G, d). Throughout this paper we assume, without loss of generality, that G contains at least k tours. This is no restriction as we may assume that all edges not in E have a sufficiently large length. Obviously, $\mathscr{H}(k)$ is in general not uniquely determined. In the extreme case, the so-called *constant TSP* (see e.g. [13]), all tours have the same length, so that any subset of $\mathscr{H}$ of cardinality k is a set of k-best tours. The difference in length between a longest and a shortest Hamiltonian tour in $\mathscr{H}(k)$ is denoted by $L_{\mathscr{H}(k)}$, i.e. $L_{\mathscr{H}(k)} := L_d(H_{(k)}) - L_d(H_{(1)})$. Note that, unlike $\mathscr{H}(k)$, $L_{\mathscr{H}(k)}$ is uniquely determined by (G, d).

Throughout this paper we will use the following example.

Example. Fig. 1 shows a weighted graph with all its Hamiltonian tours. The set of 5-best Hamiltonian tours is in this case unique and given by $\mathscr{H}(5) = \{H_{(1)}, \dots, H_{(5)}\}$.

Now consider the computational complexity of the k-best TSP. We distinguish the cases of variable and fixed k. In the case that k is variable, i.e. k is included in the input, the k-best TSP is $\mathscr{NP}$-hard, because the TSP itself is contained in the k-best TSP. Now consider the case that k is fixed, i.e. k is not part of the input. Since a shortest tour can be easily determined from a set of k-best tours, it follows that the k-best TSP is also $\mathscr{NP}$-hard for fixed k. So, we have the following result.

Theorem 1. *For each $k \geqslant 1$, the k-best TSP is $\mathscr{NP}$-hard for both fixed and variable k.*

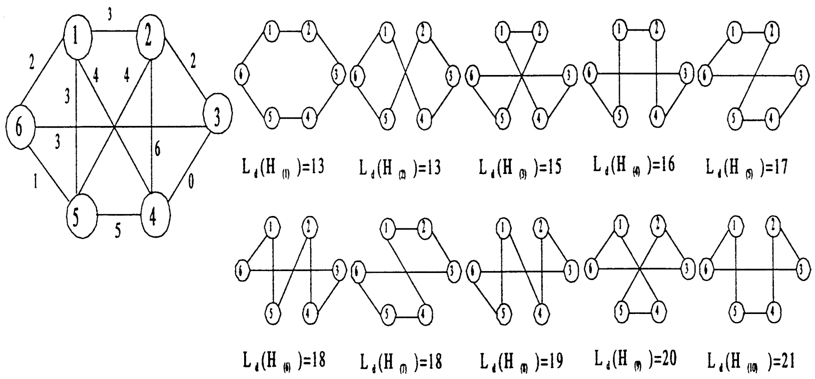

Fig. 1. (G, d) with all its Hamiltonian tours.

For any I, $O \subseteq E$, the set $\{H \in \mathscr{H} : I \subseteq H \subseteq E \backslash O\}$ is called a *restricted set of tours* in $\mathscr{H}$. For combinatorial optimization problems, it is shown in [2] how to solve an instance with a restricted set of feasible solutions as a "regular" (nonrestricted) instance. In the following theorem this result is applied to the TSP by showing how to modify the vector of edge lengths in order to solve an instance of the TSP with a restricted set of tours.

Theorem 2. *Let* $I, O \subseteq E$ *and* $E > 2n \cdot \max\{|d(e)| : e \in E\}$. *For each* $e \in E$, *define* $d'(e) := \infty$ *if* $e \in O, d'(e) := -M$ *if* $e \in I$, *and* $d'(e) := d(e)$ *otherwise. If* $\{H \in \mathscr{H} : I \subseteq H \subseteq E \backslash O\} \neq \emptyset$ *then* $\operatorname{argmin}\{L_d(H) : H \in \mathscr{H}, I \subseteq H \subseteq E \backslash O\} = \operatorname{argmin}(L_{d'}(H) : H \in \mathscr{H}\}$.

Proof. See [2]. Note that $M > 2m \cdot \max\{|d(e)| : e \in E\}$ in [2]. Since each tour contains n edges, it is sufficient for the TSP that $M > 2n \cdot \max\{|d(e)| : e \in E\}$. □

3. Determining *k*-best tours by using partitioning algorithms

In this section we consider algorithms for determining a set of k-best tours that exploit the fact that algorithms are available for solving instances of the TSP. We apply the following basic idea. Assume that all feasible solutions (tours) are partitioned in pairwise disjoint sets such that each set is a "restricted set of tours". Recall that according to Theorem 2 an optimal tour in each set can be determined by applying an algorithm for the (ordinary) TSP. From the optimal solutions of all sets, the k-best solutions can be determined.

In the literature two algorithms are given for partitioning all feasible solutions in pairwise disjoint sets in order to determine a set of k-best solutions to general discrete optimization problems, namely the algorithm of Lawler [2] and the algorithm of Hamacher and Queyranne [2]. In this paper the algorithms of Lawler and Hamacher and Queyranne are called *partitioning algorithms*. The two algorithms differ in the way they partition the set of all feasible solutions. We discuss the application of both algorithms to the TSP.

Due to the fact that the k-best TSP is $\mathcal{NP}$-hard, it is not straightforward to give a good performance measure for both partitioning algorithms. We will compare the two algorithms by looking at the number of calls in the algorithm to solve a (restricted) TSP.

We first describe how the Lawler algorithm can be used for determining a set of k-best tours. Before presenting the formal description of the algorithm, we consider a single iteration in detail. Assume that at the beginning of iteration i a set $\mathscr{H}(i-1)$ of $(i-1)$-best tours has been determined and that all feasible solutions $\mathscr{H} \backslash \mathscr{H}(i-1)$ have been partitioned in a collection $\mathscr{L}^{[i]}$ of restricted sets of tours. A tour $H^{[i]}$ is determined from the shortest tours in each of the sets $\mathscr{S}$ in $\mathscr{L}^{[i]}$ by taking any

$$\mathscr{S}^{|i|} \in \operatorname{argmin}\{\min\{L_d(H): H \in \mathscr{S}\}: S \in \mathscr{L}^{[i]}\} \text{ and } H^{[i]} \in \operatorname{argmin}\{L_d(H): H \in \mathscr{S}^{[i]}\}.$$

Then, $\mathscr{H}(i-1) \cup \{H^{[i]}\}$ is a set of i-best tours. Next, assume that the edges are labelled is such a way that $H^{[i]} = \{e_1, \dots, e_n\}$. Then, the feasible solution $\mathscr{S}^{[i]} \backslash \{H^{[i]}\}$ are partitioned in sets

$$\{H \in \mathscr{S}^{[i]}: \{e_1, \dots, e_{j-1}\} \subseteq H \subseteq E \backslash \{e_j\}\} \text{ for } j = 1, \dots, n,$$

with by definition $\{e_1, \dots, e_{j-1}\} = \emptyset$ for $j = 1$. At the end of iteration i, a set $\mathscr{H}(i)$ of i-best tours is determined and all feasible solutions $\mathscr{H} \backslash \mathscr{H}(i)$ are partitioned in a collection $\mathscr{L}^{[i+1]}$ of restricted sets of tours. This completes the detailed description of iteration i.

The formal description of the Lawler algorithm for the k-best TSP is given below.

```
Algorithm 3.1: Lawler algorithm for determining k-best tours.
input: $(G, d)$ with the set of tours $\mathscr{H}$ and some integer $1 \leqslant k \leqslant |\mathscr{H}|$.
Output: A set $\mathscr{H}(k)$ of $k$-best tours in $(G, d)$.
$\mathscr{L}^{[1]} := \{\mathscr{H}\}$; $\mathscr{H}(0) := \emptyset$;
for $i := 1$ to $k$ do
begin
    'Take any $\mathscr{S}^{[i]} \in \operatorname{argmin}\{\min\{L_d(H): H \in \mathscr{S}\}: \mathscr{S} \in \mathscr{L}^{[i]}\}$ and
    $H^{[i]} \in \operatorname{argmin}\{L_d(H): H \in \mathscr{S}^{[i]}\}$';
    $\mathscr{H}(i) := \mathscr{H}(i-1) \cup \{H^{[i]}\}$;
    'Label the edges in such a way that $H^{[i]} = \{e_1, \dots, e_n\}$';
    $\mathscr{L}^{[i+1]} := (\mathscr{L}^{[i]} \backslash \{\mathscr{S}^{[i]}\}) \cup \{\{H \in \mathscr{S}^{[i]}: \{e_1, \dots, e_{j-1}\} \subseteq E \backslash \{e_j\}\}: j = 1, \dots, n\}$
end
```

The following theorem establishes the correctness of Algorithm 3.1.

Theorem 3. *Consider the weighted graph (G, d) with the set of tours $\mathscr{H}$. Let k be an integer such that $1 \leqslant k \leqslant |\mathscr{H}|$. Then, for each $i = 1, \dots, k$ the following assertions hold*:

(1) *$\mathscr{L}^{[i]}$ is a partitioning of $\mathscr{H} \backslash \{H^{[1]}, \dots, H^{[i-1]}\}$.*
(2) *$\{H^{[1]}, \dots, H^{[i]}\}$ is a set of i-best tours.*
(3) *Each element in $\mathscr{L}^{[i]}$ is a restricted set of tours.*

Proof. The proofs of (1)–(3) are by induction. The results are obvious for $i = 1$. Let $i \in \{1, \dots, k-1\}$, and assume that (1)–(3) hold for all $l = 1, \dots, i$. From the assumption that $k \leqslant |\mathscr{H}|$ and the induction hypothesis $\cup \mathscr{L}^{[i]} = \mathscr{H} \backslash \{H^{[1]}, \dots, H^{[i-1]}\}$, it follows that $\cup \mathscr{L}^{[i]} \neq \emptyset$. Hence, both $\operatorname{argmin}\{\min\{L_d(H): H \in \mathscr{S}\}: \mathscr{S} \in \mathscr{L}^{[i]}\} \neq \emptyset$ and $\operatorname{argmin}\{L_d(H): H \in \mathscr{S}^{[i]}\} \neq \emptyset$.

(1) We first prove the following claim:

Claim: For each $\mathscr{S} \subseteq \mathscr{H}$ and $T = \{e_1, \dots, e_n\} \in \mathscr{S}$, it holds that

$$\mathscr{S} \backslash \{T\} = \cup \{\{H \in \mathscr{S}: \{e_1, \dots, e_{j-1}\} \subseteq H \subseteq E \backslash \{e_j\}\}: j = 1, \dots, n\}.$$

Proof of the claim. For each $j = 1, \dots, n$, it holds that $T \notin \{H \in \mathscr{S}: \{e_1, \dots, e_{j-1}\} \subseteq H \subseteq E \backslash \{e_j\}\}$, so that $\cup \{\{H \in \mathscr{S}: \{e_1, \dots, e_{j-1}\} \subseteq H \subseteq E \backslash \{e_j\}\}: j = 1, \dots, n\} \subseteq \mathscr{S} \backslash \{T\}$. On the other hand, if $H \in \mathscr{S} \backslash \{T\}$, and $j \in \{1, \dots, n\}$ is the smallest index such that $e_j \notin H$, then $H \in \{H \in \mathscr{S}: \{e_1, \dots, e_{j-1}\} \subseteq H \subseteq E \backslash \{e_j\}\}$. This proves the claim.

In order to prove that $\mathscr{L}^{[i]}$ is a partitioning of $\mathscr{H} \backslash \{H^{[1]}, \dots, H^{[i-1]}\}$, we first show that the elements of $\mathscr{L}^{[i]}$ are pairwise disjoint. The reader can easily check that in all cases: (a) $\mathscr{S}, \mathscr{S}' \in \mathscr{L}^{[i]}$, (b) $\mathscr{S} \in \mathscr{L}^{[i]} \backslash \{\mathscr{S}^{[i]}\}$ and $\mathscr{S}' = \{H \in \mathscr{S}^{[i]}: \{e_1, \dots, e_j\} \subseteq E \backslash \{e_j\}\} \subseteq \mathscr{S}^{[i]}$ for some $j \in \{1, \dots, n\}$, and (c) $\mathscr{S} = \{H \in \mathscr{S}^{[i]}: \{e_1, \dots, e_j\} \subseteq H \subseteq E \backslash \{e_j\}\}$ and $\mathscr{S}' = \{H \in \mathscr{S}^{[i]}: \{e_1, \dots, e_{j'}\} \subseteq H \subseteq E \backslash \{e_{j'}\}\}$ for some $j, j' \in \{1, \dots, n\}$ with $j \neq j'$ the assertion holds.

Now we show that $\cup \mathscr{L}^{[i]} = \mathscr{H} \backslash \{H^{[1]}, \dots, H^{[i-1]}\}$. By definition,

$$\cup \mathscr{L}^{[i+1]} = \cup (\mathscr{L}^{[i]} \backslash \{\mathscr{S}^{[i]}\}) \cup \bigcup_{j=1}^{n} \{H \in \mathscr{S}^{[i]}: \{e_1, \dots, e_{j-1}\} \subseteq H \subseteq E \backslash \{e_j\}\},$$

which can be rewritten, by applying the Claim, as

$$(\cup \mathscr{L}^{[i]}) \backslash \{\mathscr{S}^{[i]}\} \cup (\mathscr{S}^{[i]} \backslash \{H^{[i]}\}) = (\cup \mathscr{L}^{[i]}) \backslash \{H^{[i]}\}.$$

Finally, the induction hypothesis gives that

$$(\cup \mathscr{L}^{[i]}) \backslash \{H^{[i]}\} = (\mathscr{H} \backslash \{H^{[1]}, \dots, H^{[i-1]}\}) \backslash \{H^{[i]}\} = \mathscr{H} \backslash \{H^{[1]}, \dots, H^{[i]}\}.$$

This proves that $\mathscr{L}^{[i]}$ is a partitioning of $\mathscr{H} \backslash \{H^{[1]}, \dots, H^{[i-1]}\}$.

(2) By the induction hypothesis, $\{H^{[1]}, \dots, H^{[i]}\}$ is a set of i-best tours. In order to show that $\{H^{[1]}, \dots, H^{[i+1]}\}$ is a set of $(i+1)$-best tours, we first show that $L_d(H^{[i+1]}) \leqslant L_d(H)$ for all $H \in \mathscr{H} \backslash \{H^{[1]}, \dots, H^{[i]}\}$. By definition, $L_d(H^{[i+1]}) = \min\{L_d(H): H \in \mathscr{S}^{[i+1]}\} = \min\{\min\{L_d(H): H \in \mathscr{S}\}: \mathscr{S} \in \mathscr{L}^{(i+1]}\} = \min\{L_d(H): H \in \cup \mathscr{L}^{[i+1]}\}$, which, by (1), is equal to $\min\{L_d(H): H \in \mathscr{H} \backslash \{H^{[1]}, \dots, H^{[i]}\}\}$. This proves that $L_d(H^{[i+1]}) \leqslant L_d(H)$ for all $H \in \mathscr{H} \backslash \{H^{[1]}, \dots, H^{[i]}\}$. Since, also $L_d(H^{[i]}) \leqslant L_d(H^{[i+1]})$, it follows that $\{H^{[1]}, \dots, H^{[i+1]}\}$ is a set of $(i+1)$-best tours.

(3) Again, the reader can easily check that in the cases: (a) $\mathscr{S} \in \mathscr{L}^{[i]}$ and (b) $\mathscr{S} = \{H \in \mathscr{S}^{[i]}: \{e_1, \dots, e_j\} \subseteq H \subseteq E \backslash \{e_j\}\} \subseteq \mathscr{S}^{[i]}$ for some $j \in \{1, \dots, n\}$, $\mathscr{S}$ is a restricted set of tours. $\square$

In the following theorem a maximum is given for the number of TSP instances that needs to be solved in Algorithm 3.1 in order to find a set of k-best tours. Note that this number possibly includes instances that correspond to empty restricted sets of tours.

Theorem 4. *At most* $1 + (k-1)(n-2)$ *instances of the TSP need to be solved in Algorithm* 3.1 *in order to determine* $\mathscr{H}(k)$.

Table 1
Algorithm 3.1 applied to (G, d) in Fig. 1.

i	$\mathscr{L}^{[i]}$
1	$\{\mathscr{H}\} = \{\{H_{(1)}, \ldots, H_{(10)}\}\}$
2	$\{H \in \mathscr{H} : H \subseteq E \backslash \{\{1,2\}\}\} = \{H_{(2)}, H_{(6)}, H_{(8)}, H_{(9)}, H_{(10)}\}$
	$\{H \in \mathscr{H} : \{\{1,2\}\} \subseteq H \subseteq E \backslash \{\{2,3\}\}\} = \{H_{(3)}, H_{(4)}, H_{(5)}\}$
	$\{H \in \mathscr{H} : \{\{1,2\},\{2,3\}\} \subseteq H \subseteq E \backslash \{\{3,4\}\}\} = \{H_{(7)}\}$
3	$\{H \in \mathscr{H} : H \subseteq E \backslash \{\{1,2\},\{1,4\}\}\} = \{H_{(6)}, H_{(10)}\}$
	$\{H \in \mathscr{H} : \{\{1,4\}\} \subseteq H \subseteq E \backslash \{\{1,2\},(3,4\}\}\} = \{H_{(8)}, H_{(9)}\}$
	$\{H \in \mathscr{H} : \{\{1,2\}\} \subseteq H \subseteq E \backslash \{\{2,3\}\}\} = \{H_{(3)}, H_{(4)}, H_{(5)}\}$
	$\{H \in \mathscr{H} : \{\{1,2\},\{2,3\}\} \subseteq H \subseteq E \backslash \{\{3,4\}\}\} = \{H_{(7)}\}$
4	$\{H \in \mathscr{H} : \{\{1,2\}\} \subseteq H \subseteq E \backslash \{\{2,3\},\{2,5\}\}\} = \{H_{(4)}\}$
	$\{H \in \mathscr{H} : \{\{1,2\},\{2,5\}\} \subseteq H \subseteq E \backslash \{\{2,3\},\{5,6\}\}\} = \{H_{(5)}\}$
	$\{H \in \mathscr{H} : H \subseteq E \backslash \{\{1,2\},\{1,4\}\}\} = \{H_{(6)}, H_{(10)}\}$
	$\{H \in \mathscr{H} : \{\{1,4\}\} \subseteq H \subseteq E \backslash \{\{1,2\},\{3,4\}\}\} = \{H_{(8)}, H_{(9)}\}$
	$\{H \in \mathscr{H} : \{\{1,2\},\{2,3\}\} \subseteq H \subseteq E \backslash \{\{3,4\}\}\} = \{H_{(7)}\}$
5	$\{H \in \mathscr{H} : \{\{1,2\},\{2,5\}\} \subseteq H \subseteq E \backslash \{\{2,3\},\{5,6\}\}\} = \{H_{(5)}\}$
	$\{H \in \mathscr{H} : H \subseteq E \backslash \{\{1,2\},\{1,4\}\}\} = \{H_{(6)}, H_{(10)}\}$
	$\{H \in \mathscr{H} : \{\{1,4\}\} \subseteq H \subseteq E \backslash \{\{1,2\},\{3,4\}\}\} = \{H_{(8)}, H_{(9)}\}$
	$\{H \in \mathscr{H} : \{\{1,2\},\{2,3\}\} \subseteq H \subseteq E \backslash \{\{3,4\}\}\} = \{H_{(7)}\}$

Proof. $H_{(1)}$ is determined by solving one instance of the TSP. In order to determine $H_{(i)}$ for $i = 2, \ldots, k$, Algorithm 3.1 creates n new instances of the TSP in each iteration. However, by labelling the edges in $H_{(i)} := \{e_1, \ldots, e_n\}$ in such a way that e_j and $e_{(j \bmod n)+1}$ are adjacent for $j = 1, \ldots, n$, it follows that $\{H \in \mathscr{S}^* : \{e_1, \ldots, e_{j-1}\} \subseteq H \subseteq E \backslash \{e_j\}\} = \emptyset$ for $j = n-1, n$. Hence, $\mathscr{H}(k)$ can be determined by solving at most $1 + (k-1)(n-2)$ instances of the TSP. $\square$

Example (*continued*). We apply Algorithm 3.1 to (G, d) in Fig. 1 in order to determine a set of 5-best tours. The results are presented in Table 1. Note that all empty restricted sets are omitted in the table. Consider for instance $i = 2$. Note that $\mathscr{L}^{[1]} = \{\mathscr{H}\}$ and let $H_{(1)} = \{e_1, \ldots, e_n\} = \{\{1,2\}, \{2,3\}, \{3,4\}, \{4,5\}, \{5,6\}, \{6,1\}\}$. Then, $\{H \in \mathscr{H} : H \subseteq E \backslash \{\{1,2\}\}\} = \{H_{(2)}, H_{(6)}, H_{(8)}, H_{(9)}, H_{(10)}\}$, $\{H \in \mathscr{H} : \{\{1,2\}\} \subseteq H \subseteq E \backslash \{\{2,3\}\}\} = \{H_{(3)}, H_{(4)}, H_{(5)}\}$, and $\{H \in \mathscr{H} : \{\{1,2\},\{2,3\}\} \subseteq H \subseteq E \backslash \{\{3,4\}\}\} = \{H_{(7)}\}$; all other sets are empty. Note that $1 + 4 \times 4 = 17$ instances of the TSP are solved in order to determine $\mathscr{H}(5)$, including 9 instances that correspond to empty restricted sets of tours.

We now describe how the Hamacher and Queyranne algorithm can be used for determining a set of k-best tours. Before presenting the formal description of the algorithm, we again first consider a single iteration in detail. Assume that at the beginning of iteration i a set $\mathscr{H}(i-1)$ of $(i-1)$-best tours has been determined and that all tours in $\mathscr{H}$ have been partitioned a collection $\mathscr{L}^{[i]}$ of $(i-1)$

restricted sets of tours. Let

$$\mathscr{L}^{[i]} := \{\mathscr{S}^{[i]}_{(1)}, \dots, \mathscr{S}^{[i]}_{(i-1)}\},$$

with $\mathscr{S}^{[i]}_{(j)}$, for $j = 1, \dots, i-1$, being a restricted set of tours and $H^{[j]}$ a shortest tour in $\mathscr{S}^{[i]}_{(j)}$. A tour $H^{[i]}$ is determined from the shortest tours in $\mathscr{S}^{[i]}_{(j)} \backslash \{H^{[j]}\}$ by taking any

$$j^* \in \operatorname{argmin}\{\min\{L_d(H) : H \in \mathscr{S}^{[i]}_{(j)} \backslash H^{[j]}\} : j = 1, \dots, i-1\}$$

and

$$H^{[i]} \in \operatorname{argmin}\{L_d(H) : H \in \mathscr{S}^{[i]}_{(j^*)} \backslash \{H^{[j^*]}\}\}.$$

Note that a shortest tour in $\mathscr{S}^{[i]}_{(j)} \backslash \{H^{[j]}\}$ can be determined by finding a second best tour in $\mathscr{S}^{[i]}_{(j)}$. Now recall from [14] that the problem whether a graph with a given Hamiltonian cycle contains a second Hamiltonian cycle is $\mathscr{N}\mathscr{P}$-complete. Hence, it easily follows that the problem of determining a second best tour in $\mathscr{S}^{[i]}_{(j)}$ given a shortest tour in $\mathscr{S}^{[i]}_{(j)}$ is $\mathscr{N}\mathscr{P}$-hard. However, a second best tour in $\mathscr{S}^{[i]}_{(j)}$ can be determined by solving a 2-best TSP by applying Algorithm 3.1 to the TSP instance corresponding to $\mathscr{S}^{[i]}_{(j)}$. Now take any $e \in H^{[j^*]} \backslash H^{[i]}$. The set of tours $\mathscr{S}^{[i]}_{(j^*)}$ is partitioned into sets $\mathscr{S}^{[i+1]}_{(j^*)}$ and $\mathscr{S}^{[i+1]}_{(i)}$ consisting of, respectively, all tours containing e and all tours not containing e. At the end of iteration i, a set $\mathscr{H}(i)$ of i-best tours is determined and all feasible solutions $\mathscr{H}$ are partitioned in the collection $\mathscr{L}^{[i+1]}$ of i restricted sets of tours. This completes the detailed description of iteration i.

The formal description of the Hamacher and Queyranne algorithm for the k-best TSP is given below.

Algorithm 3.2: Hamacher and Queyranne algorithm for determining k-best tours.

```
input: $(G, d)$ with the set of tours $\mathscr{H}$ and some integer $1 \leqslant k \leqslant |\mathscr{H}|$.
Output: A set $\mathscr{H}(k)$ of $k$-best tours in $(G, d)$.
'Determine a shortest tour $H^{[1]}$ in $(G, d)$';
$\mathscr{S}^{[2]}_{(1)} := \mathscr{H}$; $\mathscr{H}(1) := \{H^{[1]}\}$
for $i := 2$ to $k$ do
begin
  'Take any $j^* \in \operatorname{argmin}\{\min\{L_d(H) : H \in \mathscr{S}^{[i]}_{(j)} \backslash H^{[j]}\} : j = 1, \dots, i-1\}$,
  $H^{[i]} \in \operatorname{argmin}\{L_d(H) : H \in \mathscr{S}^{[i]}_{(j^*)} \backslash \{H^{[j^*]}\}\}$ and $e \in H^{[j^*]} \backslash H^{[i]}$';
  $\mathscr{H}(i) := \mathscr{H}(i-1) \cup \{H^{[i]}\}$;
for all $j \in \{1, \dots, i-1\} \backslash \{j^*\}$ do $\mathscr{S}^{[i+1]}_{(j)} := \mathscr{S}^{[i]}_{(j)}$;
$\mathscr{S}^{[i+1]}_{(j^*)} := \{H \in \mathscr{S}^{[i]}_{(j^*)} : e \in H\}$; $\mathscr{S}^{[i+1]}_{(i)} := \{H \in \mathscr{S}^{[i+1]}_{(j^*)} : e \notin H\}$;
end
```

The following theorem establishes the correctness of Algorithm 3.2.

Theorem 5. *Consider the weighted graph (G, d) with the set of tours $\mathscr{H}$. Let k be an integer such that $1 \leqslant k \leqslant |\mathscr{H}|$. Then, for each $i = 2, \dots, k$ and $j = 1, \dots, i-1$ the following assertions hold.*

(1) *$\mathscr{L}^{[i]}$ is a partitioning of $\mathscr{H}$.*
(2) *$H^{[j]}$ is a shortest tour in $\mathscr{S}^{[i]}_{(j)}$.*
(3) *$\{H^{[1]}, \dots, H^{[i]}\}$ is a set of i-best tours.*
(4) *$\mathscr{S}^{[i]}_{(j)}$ is a restricted set of tours.*

Proof. The proofs of (1)–(4) are by induction. The results are obvious for $i = 2$. Let $i \in \{2, \dots, k-1\}$, and assume that (1)–(4) hold for all $l = 2, \dots, i$. From the assumption that $k \leqslant |\mathscr{H}|$ and the induction hypothesis $\cup\{\mathscr{S}^{[i]}_{(j)}: j = 1, \dots, i-1\} = \mathscr{H}$, it follows that both argmin $\{\min\{L_d(H): H \in S^{[i]}_{(j)} \backslash \{H^{[j]}\}\}: j = 1, \dots, \mathrm{i}-1\} \neq \emptyset$ and argmin$\{L_d(H): H \in \mathscr{S}^{[i]}_{(j^*)} \backslash \{H^{[j^*]}\}\} \neq \emptyset$.

(1) We first show for $j, j' = 1, \dots, i-1, j \neq j'$ that $\mathscr{S}^{[i]}_{(j)} \cap \mathscr{S}^{[i]}_{(j')} = \emptyset$. The reader can easily check that in all cases: (a) $j, j' \in \{1, \dots, i-1\} \backslash \{j^*\}$, (b) $j = j^*$ and $j' \in \{1, \dots, i-1\} \backslash \{j^*\}$, (c) $j = i$ and $j' \in \{1, \dots, i-1\} \backslash \{j^*\}$, and (d) $j = j^*$ and $j' = i$ the assertion holds. Now we show that also $\cup\{\mathscr{S}^{[i]}_{(j)}: j = 1, \dots, i-1\} = \mathscr{H}$. Since $\mathscr{S}^{[i+1]}_{(i)} \cup \mathscr{S}^{[i+1]}_{(j^*)} = \mathscr{S}^{[i]}_{(j^*)}$ and $\mathscr{S}^{[i+1]}_{(j)} = \mathscr{S}^{[i]}_{(j)}$ for $j \in \{1, \dots, i-1\} \backslash \{j^*\}$, it follows that $\cup \mathscr{S}^{[i+1]}_{(j)}: j = 1, \dots, i\} = \cup \mathscr{S}^{[i]}_{(j)}: j = 1, \dots, i-1\} = \mathscr{H}$ which proves that $\mathscr{L}^{[i]}$ is a partitioning of $\mathscr{H}$.

(2) The reader can easily check that in all cases: (a) $j \in \{1, \dots, i-1\} \backslash \{j^*\}$, (b) $j = j^*$, and (c) $j = i$, $H_{(j)}$ is a shortest tour in $\mathscr{S}^{[i+1]}_{(j)}$.

(3) By the induction hypothesis, $\{H^{[1]}, \dots, H^{[i]}\}$ is a set of i-best tours. In order to show that $\{H^{[1]}, \dots, H^{[i]}\}$ is a set of i-best tours we first show that $L_d(H^{[i+1]} \leqslant L_d(H)$ for all $H \in \mathscr{H} \backslash \{H^{[1]}, \dots, H^{[i]}\}$. By definition, $L_d(H^{[i+1]}) = \min\{L_d(H): H \in \mathscr{S}^{[i+1]}_{(j^*)} \backslash \{H^{[j^*]}\}\} = \min\{\min\{L_d(H): H \in \mathscr{S}^{[i+1]}_{(j)} \backslash \{H^{[j]}\}\}: j = 1, \dots, i\}$, which can be written as $\min\{L_d(H)$: $H \in (\cup \mathscr{S}^{[i]}) \backslash \{H^{[1]}, \dots, H^{[i]}\}\}$ and by (1) this is equal to $\min\{L_d(H): H \in \mathscr{H} \backslash \{H^{[1]}, \dots, H^{[i]}\}\}$. This proves that $L_d(H^{[i+1]} \leqslant L_d(H)$ for all $H \in \mathscr{H} \backslash \{H^{[1]}, \dots, H^{[i]}\}$. Since, also $L_d(H^{[i]}) \leqslant L_d(H^{[i+1]})$, it follows that $\{H^{[1]}, \dots, H^{[i+1]}\}$ is a set of $(i+1)$-best tours.

(4) Again, the reader can easily check that $\mathscr{S}^{[i+1]}_{(j)}$ is a restricted set of tours in the cases (a) $j \in \{1, \dots, i-1\} \backslash \{j^*\}$, (b) $j = j^*$, and (c) $j = 1$. □

In the following theorem a maximum is given for the number of TSP instances that needs to be solved in Algorithm 3.2 in order to find a set of k-best tours. This maximum is based on determining a second best tour in $\mathscr{S}^{[i]}_{(j)}$, for $j = 1, \dots, i-1$, by applying Algorithm 3.1.

Theorem 6. *At most* $1 + (2k-3)(n-2)$ *instances of the TSP need to be solved in Algorithm* 3.2 *in order to determine* $\mathscr{H}(k)$.

Proof. A set $\mathscr{H}(2)$ can be determined by applying Algorithm 3.1 at the cost of solving at most $1 + (n-2)$ instances of the TSP. A tour $H^{[i]}$, for $i = 3, \dots, k$, can be determined by creating two new instances of the 2-best TSP in each iteration of Algorithm 3.2. In both instances a shortest tour is already known, so that in each instance a set of 2-best tours can be determined, using Algorithm 3.1, at the cost of solving at most $(n-2)$ instances of the TSP. Hence, a set k-best tours in (G, d) can be determined by solving at most $1 + (n-2) + (k-2)2(n-2) = 1 + (2k-3)(n-2)$ instances of the TSP. □

Table 2
Algorithm 3.2 applied to (G, d) in Fig. 1.

i	$\mathscr{L}^{[i]} = \{\mathscr{S}^{[i]}_{(1)}, \ldots, \mathscr{S}^{[i]}_{(i-1)}\}$
2	$\mathscr{S}^{[i]}_{(1)} = \{H_{(1)}, \ldots, H_{(10)}\}$
3	$\mathscr{S}^{[i]}_{(1)} = \{H \in \mathscr{H} : \{\{1,2\}\} \subseteq H\} = \{H_{(1)}, H_{(3)}, H_{(4)}, H_{(5)}, H_{(7)}\}$
	$\mathscr{S}^{[i]}_{(2)} = \{H \in \mathscr{H} : H \subseteq E \backslash \{\{1,2\}\}\} = \{H_{(2)}, H_{(6)}, H_{(8)}, H_{(9)}, H_{(10)}\}$
4	$\mathscr{S}^{[i]}_{(1)} = \{H \in \mathscr{H} : \{\{1,2\},\{2,3\}\} \subseteq H\} = \{H_{(1)}, H_{(7)}\}$
	$\mathscr{S}^{[i]}_{(2)} = \{H \in \mathscr{H} : H \subseteq E \backslash \{\{1,2\}\}\} = \{H_{(2)}, H_{(6)}, H_{(8)}, H_{(9)}, H_{(10)}\}$
	$\mathscr{S}^{[i]}_{(3)} = H \in \mathscr{H} : \{\{1,2\}\} \subseteq H \subseteq \{\{2,3\}\}\} = \{H_{(3)}, H_{(4)}, H_{(5)}\}$
5	$\mathscr{S}^{[i]}_{(1)} = \{H \in \mathscr{H} : \{\{1,2\},\{2,3\}\} \subseteq H\} = \{H_{(1)}, H_{(7)}\}$
	$\mathscr{S}^{[i]}_{(2)} = \{H \in \mathscr{H} : H \subseteq E \backslash \{\{1,2\}\}\} = \{H_{(2)}, H_{(6)}, H_{(8)}, H_{(9)}, H_{(10)}\}$
	$\mathscr{S}^{[i]}_{(3)} = \{H \in \mathscr{H} : \{\{1,2\},\{2,3\}\} \subseteq H \subseteq \{\{2,3\}\}\} = \{H_{(3)}, H_{(5)}\}$
	$\mathscr{S}^{[i]}_{(4)} = \{H \in \mathscr{H} : \{\{1,2\}\} \subseteq H \subseteq \{\{2,3\},\{2,5\}\}\} = \{H_{(4)}\}$

Example (*continued*). We apply Algorithm 3.2 to (G, d) in Fig. 1 in order to determine a set of 5-best tours. The results are presented in Table 2. Consider the instance, $i = 2$. Note that $\mathscr{S}^{[2]}_{(1)} = \{\mathscr{H}\}$ and $H_{(2)}$ is a shortest tour is $\mathscr{S}^{[2]}_{(1)} \backslash \{H_{(1)}\}$. Let $e = \{1,2\} \in H_{(1)} \backslash H_{(2)}$. Then, $\mathscr{S}^{[3]}_{(1)} := \{H \in \mathscr{H} : \{\{1,2\}\} \subseteq H\} = \{H_{(1)}, H_{(3)}, H_{(4)}, H_{(5)}, H_{(7)}\}$ and $\mathscr{S}^{[3]}_{(2)} := \{H \in \mathscr{H} : H \subseteq E \backslash \{\{1,2\}\}\} = \{H_{(2)}, H_{(6)}, H_{(8)}, H_{(9)}, H_{(10)}\}$. Note that $1 + 7 \times 4 = 29$ instances of the TSP are solved in order to determine $\mathscr{H}(5)$, including 17 instances corresponding to empty restricted sets of tours. These instances appeared while solving instances of the 2-best TSP using Algorithm 3.1.

On the basis of Theorems 4 and 6, we conclude that, for solving the k-best TSP, the algorithm of Lawler is "better" than the algorithm of Hamacher and Queyranne in the sense that if $k \geqslant 2$ a fewer number of instances of the TSP is to be solved. The Lawler algorithm requires $(k-2)(n-2)$ less calls of the TSP algorithm.

We conclude this section by presenting, as an illustration, the running time at Algorithm 3.1 for a number of instances in the TSPLIB library (see [12]). For this purpose, we implemented Algorithm 3.1 with the branch-and-bound algorithm of Volgenant and Jonker [15] and solved the 50-best TSP for instances with the number of cities varying between 17 and 137. Fig. 2 shows the running time of Algorithm 3.1 for solving the k-best TSP as a fraction of the running time for solving the 1-best TSP. It can be observed that the increase in running time of Algorithm 3.1 is not as pessimistic as might be expected on the basis of the number of TSP instances.

4. A branch-and-bound method for determining *k*-best tours

Since solution methods such as *branch-and-bound* and *branch-and-cut* are capable of solving large instances of the TSP (see e.g. [16]), it is of interest to consider modifications of these methods in order to solve the k-best TSP. In this section we modify the branch-and-bound method in such a way that a set of k-best tours is determined.

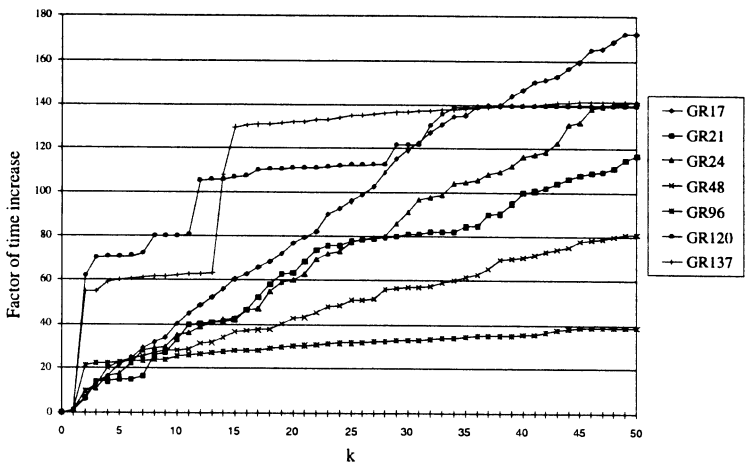

Fig. 2. Running time of Algorithm 3.1 for solving the k-best TSP as a fraction of the running time for solving the 1-best TSP.

The main ingredient of branch-and-bound methods is the dynamically generated *Branch-and-Bound Tree* (BBT). Any node is this tree corresponds to an instance of the TSP with a restricted set of tours. Initially, the BBT contains only one node, the so-called *root*, which corresponds to the set $\mathscr{H}$. The BBT is expanded by *branching* on edges in E. Consider, for instance, the expansion of a node in the BBT with the nonempty set of tours $\mathscr{S}$ by branching on some edge e. To that end a "left node" with the set of tours in $\mathscr{S}$ containing e is introduced and a "right node" with the set of tours in $\mathscr{S}$ not containing e. During the expansion of the BBT a set $\mathscr{C}$ of at most k tours (called the incumbent) is stored being the best tours found so far. Initially, $\mathscr{C}$ is empty and when the algorithm is completed then $\mathscr{C}$ contains a set of k-best tours. The set $\mathscr{C}$ will be used to give an upper bound U on the length of the longest tour in $\mathscr{H}(k)$ in the following way. If $\mathscr{C}$ contains k tours then U is equal to the length of the longest tour in $\mathscr{C}$, and otherwise U is defined equal to infinity.

We now consider the exploration of a node in the BBT with the restricted set of tours $\mathscr{S} := \{H \in \mathscr{H} : I \subseteq H \subseteq E \backslash O\}$, for some $I, O \subseteq E$, in more detail. The branch-and-bound algorithm starts by determining a lower bound, denoted by $LB(I, O)$, such that $L_d(H) \geqslant LB(I, O)$ for all tours H in $\mathscr{S}$. Clearly, if $LB(I, O) \geqslant U$ then no tour in $\mathscr{S}$ should be considered for $\mathscr{C}$. Such a node is called *fathomed*. The exploration of the BBT is stopped once all nodes are fathomed and at that moment $\mathscr{C}$ is equal to $\mathscr{H}(k)$. In the exploration of a node with $LB(I, O) < U$ two cases have to be considered. In the case that $\mathscr{S}$ consists of a single tour then both $\mathscr{C}$ and U are updated as follows. If $\mathscr{C}$ contains k tours then first a longest tour in $\mathscr{C}$ is removed. The tour in $\mathscr{S}$ is added to $\mathscr{C}$ and the length of the longest tour in the renewed $\mathscr{C}$ is determined. If $\mathscr{C}$ contains less than k tours then the tour in $\mathscr{S}$ is simply added to $\mathscr{C}$ and the length of the longest tour in $\mathscr{C}$ is recomputed. In the case that $\mathscr{S}$ consists of more than one tour then two new nodes are created by branching to an edge in E.

The formal description of a modified branch-and-bound algorithm is given in Algorithm 4.1 below. The algorithm uses the recursive procedure EXPLORE to evaluate the nodes in the BBT.

This procedure has three parameters, namely the sets I, O, and $\mathscr{C}$. Note that the set I corresponds to a partial tour. The modified branch-and-bound algorithm is started by calling the procedure EXPLORE for the empty sets I, O, and $\mathscr{C}$. Note that the procedure EXPLORE explores the BBT in a depth-first fashion, i.e. the procedure first considers the case that the branching edge is added to (the partial tour) I and thereafter the case that e is added to O.

For details on branching strategies and lower bounding, we refer to [17, 18]. Note that our modification of the branch-and-bound method leaves some possibilities for improving the efficiency. Especially, initializing the set $\mathscr{C}$ and the upper bound U by applying heuristics may drastically reduce the number of nodes in the BBT.

Algorithm 4.1: Branch-and-bound algorithm for finding k-best tours.
input: (G, d) with the set of tours $\mathscr{H}$ and some integer $1 \leqslant k \leqslant |\mathscr{H}|$.
Output: A set $\mathscr{H}(k)$ of k-best tours in (G, d).

```
procedure EXPLORE($I, O, \mathscr{C}$)
begin
   'determine $LB(I, O)$ with respect to $(G, d)$';
   if $LB(I, O) < U$
   then if $|I| = n - 1$
      then begin
            'Let $H$ denote the completion of the partial tour $I$';
            if $|\mathscr{C}| = k$ then 'remove a longest tour in $\mathscr{C}$';
            $\mathscr{C} := \mathscr{C} \cup \{H\}$;
            if $|\mathscr{C}| = k$ then 'let $U$ be the length of a longest tour in $\mathscr{C}$'
            end
      else begin
            'determine a branching edge $e$';
            EXPLORE($I \cup \{e\}, O, \mathscr{C}$);
            'determine $LB(I, O \cup \{e\})$ with respect to $(G, d)$';
            if $LB(I, O \cup \{e\}) < U$ then EXPLORE($I, O \cup \{e\}, \mathscr{C}$)
            end
end;
begin
   $U := \infty$;
   $\mathscr{H}(k) := \emptyset$;
   EXPLORE($\emptyset, \emptyset, \mathscr{H}(k)$);
end
```

Example (*continued*). Fig. 3 shows the BBT for the case that Algorithm 4.1 is applied to (G, d) in Fig. 1 in order to determine a set of 5-best tours. The (nonleaf) nodes in the tree are labelled from 0 (the root) through 14 such as to indicate the sequence in which the nodes are evaluated. The lower bounds and branching edges are indicated next to the nodes and the left branches, respectively. For instance, the root has a lower bound of 8, node 1 corresponds to the tours in $\mathscr{H}$ that contain the edge $\{3, 4\}$, and node 13 corresponds to the tours in $\mathscr{H}$ that do not contain the edge $\{3, 4\}$. The

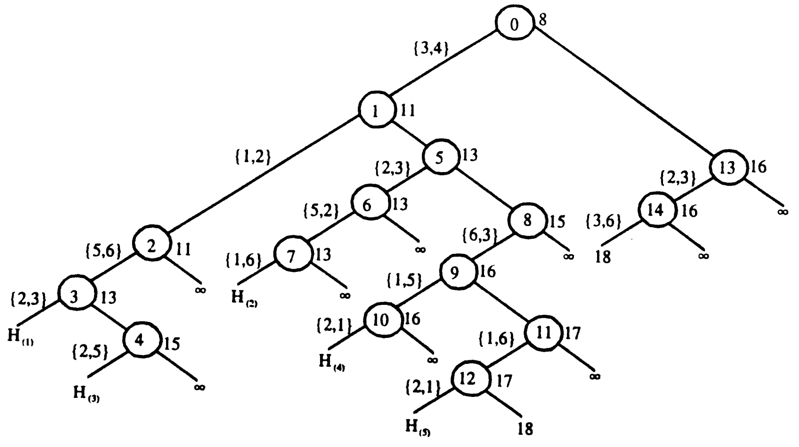

Fig. 3. Branch-and-bound tree for determining $\mathscr{H}(5)$ in (G, d) of Fig. 1.

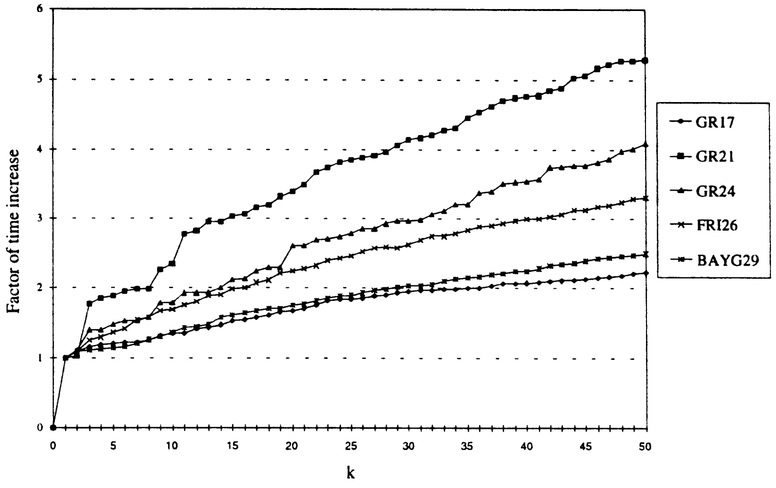

Fig. 4. The running time of Algorithm 4.1 for solving the k-best TSP as a fraction of the running time for solving the 1-best TSP.

lower bounds in Fig. 3 are obtained by subtracting from each row and successively each column in the current distance matrix the smallest element in the row (column), and adding the sum of these values to the length $L_d(I)$ of the partial tour. For instance, the lower bound of the root is obtained from the distance matrix $D = [d(i,j)]$ by subtracting 2 from the rows 1 and 2, subtracting 1 from the rows 5 and 6, and subtracting 1 from the columns 1 and 2. For each of the leafs, a tour is indicated when the tour is part of the set of 5-best tours and otherwise a lower bound is given. After exploring the left subtree (the nodes 0 through 12) the upper bound U is equal to 17, so that also the right subtree (the nodes 13 and 14) with lower bound 16 still needs to be explored.

We conclude this section by presenting, as an illustration, the running time of Algorithm 4.1 for a number of instances from the TSPLIB library. Unfortunately, we were not able to adapt the branch-and-bound code of Volgenant and Jonker [15] for solving the k-best TSP. We therefore implemented Algorithm 4.1 for the branch-and-bound algorithm of Syslo et al. [19] and solved the 50-best TSP for instances with the number of cities varying between 17 and 30. Fig. 4 shows the running time of Algorithm 4.1 for solving the k-best TSP as a fraction of the running time for solving the 1-best TSP. It can be observed that the running time of Algorithm 4.1 grows very "slowly" as a function of k. In comparison with the increase in running time of the Lawler algorithm for the TSP (see Fig. 2) the increase in running time of the modified branch-and-bound algorithm grows much slower.

5. The results of numerical experiments

In this section we investigate a number of characteristics of the sets of k-best tours for a selection of the instances in the TSPLIB library. We therefore consider again the results of solving the 50-best TSP for instances in the TSPLIB library with the number of cities varying between 17 and 137.

Firstly, we are interested in the relationship between k and the difference in length between a longest and a shortest tour in the set of k-best tours, i.e. $L_{\mathscr{H}(k)}$. Fig. 5 shows for a selection of instances from TSPLIB the percentage of increase in length as fraction of the length of the shortest tour for k ranging between 1 and 50. It can be observed that the percentage of increase in length is very small, i.e. for all instances less than 5%.

A second point of interest is the relationship between k and the structure of the tours in the set of k-best tours. To that end we consider the set of edges that are in *all* tours and the set of edges that are in *at least one* tour in $\{H_{(1)}, \ldots, H_{(k)}\}$. Figs. 6 and 7 show the cardinality of, respectively, the intersection and the union of the set of k-best tours for a selection of instances in TSPLIB. Observe

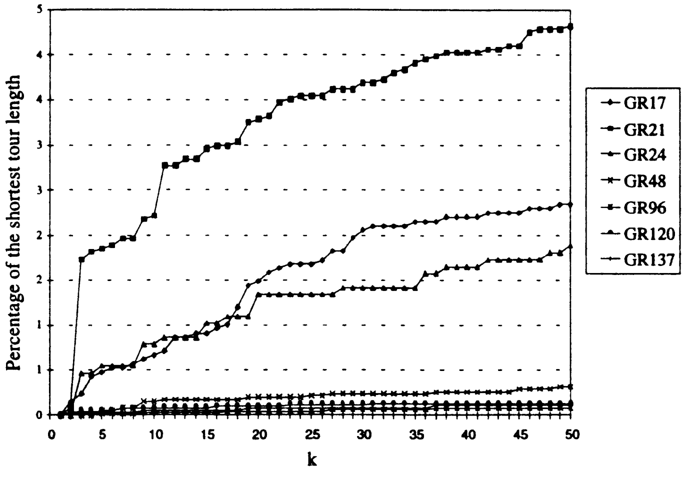

Fig. 5. Difference in length between a longest and a shortest tour in $\mathscr{H}(k)$.

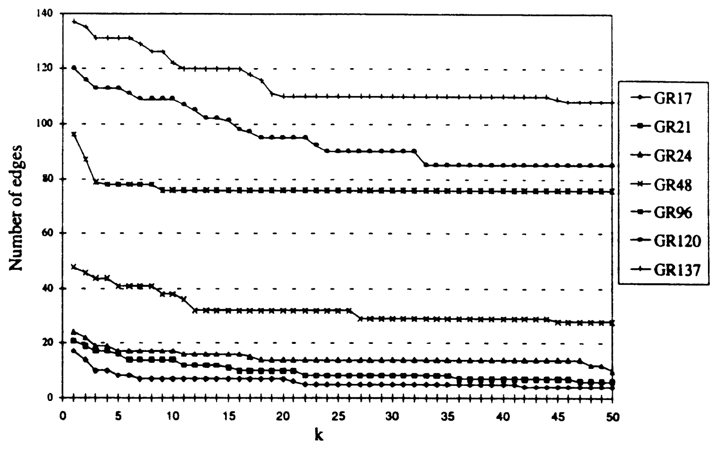

Fig. 6. Cardinality of the intersection of the set of k-best tours.

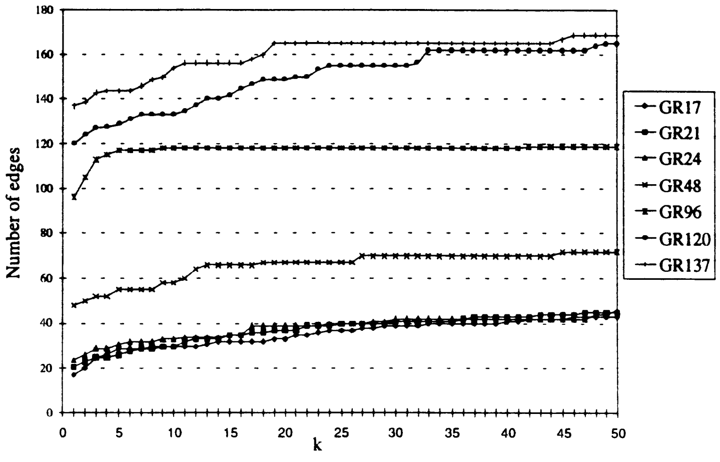

Fig. 7. Cardinality of the union of the set of k-best tours.

that the cardinality of the intersection decreases slowly and that the cardinality of the union increases slowly for increasing values of k. Hence, the structure of the tours in the set of k-best tours changes only slowly.

6. Conclusions

In this paper we considered two types of algorithms for determining a set of k-best solutions for the TSP, namely the partitioning algorithms of Lawler and Hamacher & Queyranne and a modification of the branch-and-bound method. The numerical results are reported for a number of instances from Reinelt's TSPLIB library.

The Lawler algorithm, using the Volgenant and Jonker branch-and-bound code, is able to solve up to medium-sized instances of the TSP with an increase in running time that grows slowly in k. The modification of the branch-and-bound algorithm of Syslo et al. is only able to solve small instances of the TSP, but the running time grows remarkably slow in k.

The experiments show that the difference in length between a longest and a shortest tour in the set of k-best tours grows only slowly in k and that also the intersection and union of the tours in the set of k-best tours are not changing fast.

Acknowledgements

The authors like to thank Joris Keizers and Reynold Zuil for implementing the algorithms and giving useful comments.

References

[1] Lawler EL. A procedure for computing the k best solutions to discrete optimization problems and its applications to the shortest path problem. Management Science 1972;18:401–7.
[2] Hamacher HW, Queyranne M. K best solutions to combinatorial optimization problems. Annls of Operations Research 1985;4:123–43.
[3] Murthy KG. An algorithm for ranking all assignments in order of increasing cost. Operations Research 1967;16:682–7.
[4] Azevedo JA, Santos Costa MEO, Silvestre Madeira JJER, Vieira Martins EQ. An algorithm for the ranking of shortest paths. European Journal of Operational Research 1993;69:97–106.
[5] Yen JY. Finding the k shortest loopless paths in a network. Management Science 1971;17:712–6.
[6] Shier DR. Iterative methods for determining the k shortest paths in a network. Networks 1976;6:205–29.
[7] Gabow HN. Two algorithms for generating weighted spanning trees in order. SIAM Journal of Computing 1977;6:139–50.
[8] Katoh, N, Ibaraki T, Mine H. An algorithm for finding k minimum spanning trees. SIAM Journal of Computing 1981;10:247–55.
[9] Chegireddy CR, Hamacher HW. Algorithms for finding k-best perfect matchings. Discrete Applied Mathematics 1987;18:155–65.
[10] Piper CJ, Zoltners AA. Some easy postoptimality analysis for zero–one programming. Management Science 1976;22:759–65.
[11] Wilson GR, Jain HK. An approach to postoptimality and sensitivity analysis of zero–one goal programs. Naval Research Logistics 1988;35:73–84.
[12] Reinelt G. TSPLIB, A traveling salesman problem library. ORSA Journal on Computing 1991;3:376–84.
[13] Gilmore PC, Lawler EL, Shmoys DB. Well solved special cases. In: Lawler EL, Lenstra JK, Rinnooy Kan AHG, Shmoys DB editors. The traveling salesman problem: a guided tour of combinatorial optimization. Ch. 4. Chichester, New York: Wiley, 1985.
[14] Johnson DS, Papadimitriou CH. Computational complexity. In: Lawler EL, Lenstra JK, Rinnooy Kan AHG, Shmoys DB editors. The traveling salesman problem: a guided tour of combinatorial optimization. Ch. 3. Chichester, New York: Wiley, 1985.
[15] Volgenant A, Jonker R. A branch and bound algorithm for the symmetric Traveling salesman problem based on the 1-tree relaxation. European Journal of Operational Research 1982;9:83–9.
[16] Padberg M, Rinaldi G. A branch-and-cut algorithm for the resolution of large scale symmetric traveling salesman problems. SIAM Review 1991;33:60–100.

[17] Balas E, Toth P. Branch and bound methods. In: Lawler EL, Lenstra JK, Rinnooy Kan AHG, Shmoys DB editors. The traveling salesman problem: a guided tour of combinatorial optimization. Ch. 10. Chichester, New York: Wiley, 1985.

[18] Ibaraki T. Enumerative approaches to combinatorial optimization (part I). Annals of Operations Research 1987;10.

[19] Syslo MM, Deo N, Kowalik JS. Discrete Optimization Algorithms with Pascal Programs. Englewood Cliffs, New Jersey: Prentice-Hall, 1983.

[20] Lawler EL, Lenstra JK, Rinnooy Kan AHG, Shmoys DB editors. The traveling salesman problem: a guided tour of combinatorial optimization. Wiley, Chichester, New York: 1985.