10

Multiobjective Versions of Some $\mathcal{NP}$-Hard Problems

10.1 The Knapsack Problem and Branch and Bound

As for the assignment problem in the previous section, we consider only finding efficient solutions. And we also restrict ourselves to the bicriterion case. The bicriterion knapsack problem is the binary integer program

$$\max f_1(x) = \sum_{i=1}^{n} c_i^1 x_i \tag{10.1}$$

$$\max f_2(x) = \sum_{i=1}^{n} c_i^2 x_i \tag{10.2}$$

$$\text{subject to } \sum w_i x_i \leq W \tag{10.3}$$

$$x_i \in \{0,1\}; \quad j = 1, \ldots, n. \tag{10.4}$$

The problem is obviously $\mathcal{NP}$-hard, as a counterpart of an $\mathcal{NP}$-hard single objective problem (see Lemma 8.11). Whether the problem is $\#\mathcal{P}$-complete or intractable is yet unknown.

We will present a branch and bound algorithm. To avoid trivial solutions and to have a meaningful problem we make some basic assumptions on the parameters of the knapsack problem. We assume that all values c_i^k, all weights w_i as well as the capacity W are nonnegative. Furthermore, no single weight exceeds capacity, i.e. $w_i \leq W$ for all $i = 1, \ldots, n$, but the total weight of all items is bigger than W, $\sum_{i=1}^{n} w_i > W$.

For the solution of knapsack problems the value to weight ratios c_i^k / w_i are of essential importance. In the single objective linear knapsack problem (where $x_i \in \{0,1\}$ is replaced by $0 \leq x_i \leq 1$),

$$\begin{aligned} \max & \sum_{i=1}^{n} c_i x_i \\ \text{subject to } & \sum_{i=1}^{n} w_i x_i \leq W \\ & x_i \leq 1 \quad i = 1, \ldots, n \\ & x_i \geq 0 \quad i = 1, \ldots, n \end{aligned}$$

they are used to easily find an optimal solution.

Assume that items $1, \ldots, n$ are ordered such that

$$\frac{c_1}{w_1} \geq \frac{c_2}{w_2} \geq \ldots \geq \frac{c_n}{w_n}. \tag{10.5}$$

Let $i^* := \min\{i : \sum_{j=1}^{i} w_j > W\}$ be the smallest index such that the weight of items 1 to i exceeds the total capacity. Item i^* is called the *critical item*. The solution of the continuous knapsack problem is simply given by taking all items 1 to $i^* - 1$ and a fraction of the critical item, that is $x_i = 1$ for $i = 1, \ldots, i^* - 1$ and

$$x_{i^*} = \frac{\left(W - \sum_{i=1}^{i^*-1} w_i\right)}{w_{j^*}}.$$

Good algorithms for the single objective problem use this fact and focus on optimization of items around i^*, see e.g. Martello and Toth (1990); Pisinger (1997); Kellerer *et al.* (2004). Ideas of such algorithms have been adapted to the bicriterion case by Ulungu and Teghem (1997).

The two criteria induce two different sequences of value to weight ratios. Let $\mathcal{O}_k$ be the ordering (10.5) according to c_i^k/w_i, $k = 1, 2$. Let r_i^k be the rank or position of item i in order $\mathcal{O}_k$ and let $\mathcal{O}$ be the order according to increasing values of $(r_i^1 + r_i^2)/2$, the average rank of an item.

The branch and bound method will create partial solutions by assigning zeros and ones to subsets of variables denoted $\mathcal{B}_0$ and $\mathcal{B}_1$, respectively. These *partial solutions* constitute nodes of the search tree. Variables not assigned either zero or one are called *free variables* for a partial solution and define a set $\mathcal{F} \subseteq \{1, \ldots, n\}$ such that $\{1, \ldots, n\} = \mathcal{B}_1 \cup \mathcal{B}_0 \cup \mathcal{F}$. A solution formed by assigning all free variables a value is called *completion* of a partial solution. Variables of a partial solution will be assigned a value according to the order $\mathcal{O}$. It is convenient to number the items in that order so that we will have

$$\mathcal{B}_1 \cup \mathcal{B}_0 = \{1, \ldots, l-1\}, \mathcal{F} = \{l, \ldots, n\}$$

for some l. Furthermore we shall denote r_k the index of the first variable in $\mathcal{F}$ according to order $\mathcal{O}_k$, for $k = 1, 2$.

Vector valued bounds will be used to fathom a node of the tree when no completion of the partial solution $(\mathcal{B}_1, \mathcal{B}_0)$ can possibly yield an efficient solution. A lower bound $(\underline{z}_1, \underline{z}_2)$ at a partial solution is simply given by the value of the variables which have already been assigned value 1,

$$(\underline{z}_1, \underline{z}_2) = \left(\sum_{i \in \mathcal{B}_1} c_i^1, \sum_{j \in \mathcal{B}_1} c_i^2 \right). \tag{10.6}$$

For the computation of upper bounds we define

$$\overline{W} := W - \sum_{i \in \mathcal{B}_1} w_i \geq 0,$$

the remaining free capacity of the knapsack after fixing variables in $\mathcal{B}_1$. Furthermore, we denote

$$s_k := \max \left\{ l_k \in \mathcal{F} : \sum_{j_k = r_k}^{l_k} w_{j_k} < \overline{W} \right\}$$

to be the last item that can be chosen to be added to a partial solution according to $\mathcal{O}_k$. Thus, $s_k + 1$ is in fact the critical item in order $\mathcal{O}_k$, taking the already fixed variables in $\mathcal{B}_0$ and $\mathcal{B}_1$ into account.

The upper bound for each objective value at a partial solution is computed according to the rule of Martello and Toth (1990).

$$\overline{z}_k = \underline{z}_k + \sum_{j_k = r_k}^{s_k} c_{j_k}^k + \max \left\{ \left[\overline{W}_k \frac{c_{s_k+2}^k}{w_{s_k+2}} \right], \left[c_{s_k+1}^k - (w_{s_k+1} - \overline{W}_k) \frac{c_{s_k}^k}{w_{s_k}} \right] \right\}, \tag{10.7}$$

where $\overline{W}_k = \overline{W} - \sum_{j_k = r_k}^{s_k} w_{j_k}$. The bound can be justified from the observation that $x_{s_{k+1}}$ cannot assume fractional values. Therefore it must be either zero or one. It is computed from the value of the already assigned variables $(\underline{z}_k)$, plus the value of those items that fit entirely, in order $\mathcal{O}_k$, plus the maximum term. The first term in the maximum in (10.7) comes from setting $x_{s_{k+1}} = 0$ and filling remaining capacity with $x_{s_{k+2}}$, while the second means setting $x_{s_{k+1}} = 1$ and removing part of x_{s_k} to satisfy the capacity of the knapsack. Another way of computing an upper bound using (10.7) is indicated in Exercise 10.1.

Given a partial solution, an assignment of zeros and ones to the free variables is sought, to find potentially efficient solutions of the whole problem. This problem related to the partial solution $(\mathcal{B}_1, \mathcal{B}_0)$ is again a bicriterion knapsack problem:

$$\begin{aligned} \max & \sum_{i\in\mathcal{F}} c_i^1 x_i + \sum_{i\in\mathcal{B}_1} c_i^1 \\ \max & \sum_{i\in\mathcal{F}} c_i^2 x_i + \sum_{i\in\mathcal{B}_1} c_i^2 \\ \text{subject to } & \sum_{i\in\mathcal{F}} w_i x_i \le \overline{W} \\ & x_i \in \{0,1\}. \end{aligned}$$

We now have all prerequisites to formulate the algorithm. The algorithm pursues a depth first strategy. That is, movement down a branch of the search tree, and therefore having more variables fixed in $\mathcal{B}_1$ or $\mathcal{B}_0$, is preferred to investigating partial solutions with fewer fixed variables. Successors of a node in a tree (branching) are distinguished by different sizes of $\mathcal{B}_1$ and $\mathcal{B}_0$. Actually, the tree can be drawn so that the sets $\mathcal{B}_1$ of successors of a node will be become smaller, as variables are moved from $\mathcal{B}_1$ to $\mathcal{B}_0$, see Figure 10.1. The idea is that by fixing many variables first according to their value to weight ratios, a good feasible solution is obtained fast, so that many branches of the tree can be fathomed early.

Throughout, we keep a list $\mathcal{L}$ of potentially nondominated points identified so far (note that due to maximization y^1 dominating y^2 means $y^1 > y^2$ here), and a list of nodes $\mathcal{N}$ still to be processed.

The list $\mathcal{N}$ is maintained as a last-in-first-out queue. Nodes are fathomed if the bounds show that they can only lead to dominated solutions, if they have been completely investigated (they represent a complete solution), or if no further succeeding node can be constructed.

When a node is fathomed, the algorithm backtracks and creates a new node by moving the last item of $\mathcal{B}_1$ to $\mathcal{B}_0$, removing all items after this new item from $\mathcal{B}_0$. If, however, the last item in $\mathcal{B}_1$ was n then the algorithm chooses the smallest v such that all items $\{v,\ldots,n\}$ were in $\mathcal{B}_1$, removes them all, and defines $\mathcal{B}_0$ to be all previous elements of $\mathcal{B}_0$ up to $v-1$ and to include v.

When a node is not fathomed, the algorithm proceeds deeper down the tree in that it creates a new successor node. This may again be done in two ways. If $\mathcal{B}_1$ allows addition of the first item l of $\mathcal{F}$, as many items as possible are included in $\mathcal{B}_1$, according to order $\mathcal{O}$, i.e. as they appear in $\mathcal{F}$. If, on the other hand, the remaining capacity $\overline{W}$ does not allow item l to be added to $\mathcal{B}_1$, the first possible item r of $\mathcal{F}$, which can be added to $\mathcal{B}_1$ is sought and item r is added to $\mathcal{B}_1$. Of course all items $i,\ldots,r-1$ must be added to $\mathcal{B}_0$. Since the current node has not been fathomed such an r must exist.

Algorithm 10.1 (Branch and bound for the knapsack problem.)

Input: *Values c_i^1 and c_i^2, weights w_i for $i = 1, \ldots, n$ and capacity W.*
Initialization: *Create root node N_0 as follows.*
Set $\mathcal{B}_1 := \emptyset$, $\mathcal{B}_0 := \emptyset$, $\mathcal{F} := \{1, \ldots, n\}$,
Set $\underline{z} := \binom{0}{0}, \overline{z} := \binom{\infty}{\infty}$, $\mathcal{L} := \emptyset$, $\mathcal{N} := \{N_0\}$.
While $\mathcal{N} \neq \emptyset$
Choose the last node $N \in \mathcal{N}$.
Compute $\overline{W}$ *and* $\underline{z}$.
Add $\underline{z}$ *to* $\mathcal{L}$ *if it is not dominated.*
Compute $\overline{z}$.
If $\{i \in \mathcal{F} : w_i \leq \overline{W}\} = \emptyset$ *or* $\overline{z}$ *is dominated by some* $y \in \mathcal{L}$
Fathom node N. $\mathcal{N} := \mathcal{N} \setminus \{N\}$.
Create a new node N' *as follows.*
Let $t := \max\{i : i \in \mathcal{B}_1\}$.
If $t < n$ *do*
$\mathcal{B}_1 := \mathcal{B}_1 \setminus \{t\}$, $\mathcal{B}_0 := (\mathcal{B}_0 \cap \{1, \ldots, t-1\}) \cup \{t\}$, $\mathcal{F} := \{t+1, \ldots, n\}$
End if.
If $t = n$ *do*
Let u *be* $\min\{j : \{j, j+1, \ldots, t-1, t\} \subset \mathcal{B}_1\}$.
Let v *be* $\max\{j : | \in \mathcal{B}_1 \setminus \{u, \ldots, t\}\}$.
$\mathcal{B}_1 := \mathcal{B}_1 \setminus \{v, u, u+1, \ldots, t-1, t\}$, $\mathcal{B}_0 := (\mathcal{B}_0 \cap \{1, \ldots, v-1\}) \cup \{v\}$,
$\mathcal{F} := \{v+1, \ldots, n\}$
End if. $\mathcal{N} := \mathcal{N} \cup \{N\}$
If set $\mathcal{B}_1$ *of* N' *is smaller than* $\mathcal{B}_1$ *of predecessor nodes of* N, *which are not predecessors of* N' *then fathom these nodes.*
If no new node can be created ($\mathcal{B}_1 = \emptyset$), *STOP.*
Otherwise
Create a new node N' *as follows.*
Let $s := \max\left\{t \in \mathcal{F} : \sum_{j=1}^{t} w_j < \overline{W}\right\}$ *according to order* $\mathcal{O}$.
If $w_l > \overline{W}$ *let* $s := l - 1$.
If $s \geq l$ *do*
$\mathcal{B}_1 := \mathcal{B}_1 \cup \{i, \ldots, s\}$, $\mathcal{B}_0 := \mathcal{B}_0$, $\mathcal{F} := \mathcal{F} \setminus \{i, \ldots, s\}$.
End if
If $s = i - 1$ *do*
Let $r := \min\{j : j \in \mathcal{F}, w_j < \overline{W}\}$ *according to order* $\mathcal{O}$.
$B_1 := B_1 \cup \{r\}$, $\mathcal{B}_0 := \mathcal{B}_0 \cup \{i, \ldots, r-1\}$, $\mathcal{F} := \mathcal{F} \setminus \{i, \ldots, r\}$

End if
$\mathcal{N} := \mathcal{N} \cup \{N\}$
End otherwise.
End while.
Output: All efficient solutions.

We illustrate Algorithm 10.1 with an example also used in Ulungu and Teghem (1997). Following the iterations along the branch and bound tree of Figure 10.1 will make clear how the algorithm works.

Example 10.1 (Ulungu and Teghem (1997)). We consider the problem

$$\begin{array}{lrrrrrrrrr} \max & 11x_1 & + & 5x_2 & + & 7x_3 & + & 13x_4 & + & 3x_5 \\ \max & 9x_1 & + & 2x_2 & + & 16x_3 & + & 5x_4 & + & 4x_5 \\ \text{subject to} & 4x_1 & + & 2x_2 & + & 8x_3 & + & 7x_4 & + & 5x_5 \leq 16 \\ & & & & & & & & & x_j \in \{0,1\}, j = 1,\ldots,5 \end{array}$$

The orders are

$$\begin{aligned} \mathcal{O}_1 &= \{x_1, x_2, x_4, x_3, x_5\} \\ \mathcal{O}_2 &= \{x_1, x_3, x_2, x_5, x_4\} \\ \mathcal{O} &= \{x_1, x_2, x_3, x_4, x_5\}. \end{aligned}$$

First, node N_0 is created with $\mathcal{B}_1 = \emptyset$, $\mathcal{B}_0 = \emptyset$, $\mathcal{F} = \{1,2,3,4,5\}$ and the bounds are initialized as $\underline{z} = \binom{0}{0}$, $\overline{z} = \binom{\infty}{\infty}$ and $\mathcal{L} := \emptyset$, $\mathcal{N} = \{N_0\}$

1. Node N_0 is selected and a new node N_1 is created with $\mathcal{B}_1 = \{1,2,3\}$, $\mathcal{B}_0 = \emptyset$, $\mathcal{F} = \{4,5\}$. Thus $\mathcal{N} = \{N_0, N_1\}$.
2. Node N_1 is selected. We compute $\overline{W} = 2$, $\underline{z} = \overline{z} = \binom{23}{27}$ and add this to $\mathcal{L}$. $\mathcal{L} = \{\binom{23}{27}\}$.
   Since $\{j \in \mathcal{F} : w_j < \overline{W}\} = \emptyset$, node N_1 is fathomed and we create node N_2. Since$t = 3$ we get $\mathcal{B}_1 = \{1,2\}$, $\mathcal{B}_0 = \{3\}$, $\mathcal{F} = \{4,5\}$.
   $N = \{N_0, N_2\}$.
3. Node N_2 is selected. We compute $\overline{W} = 10$, $\underline{z} = \binom{16}{11}$, check that $\{j \in F : w_j < \overline{W}\} = \{4,5\} \neq \emptyset$. The upper bound is $\overline{z} = \binom{29}{18}$.
   Node N_3 is created with $s = 4$ and $\mathcal{B}_1 = \{1,2,4\}$, $\mathcal{B}_0 = \{3\}$, $\mathcal{F} = \{5\}$.
   $\mathcal{N} = \{N_0, N_2, N_3\}$
4. N_3 is selected. $\overline{W} = 3$ and $\underline{z} = \overline{z} = \binom{29}{16}$ is added to $\mathcal{L}$ so that $\mathcal{L} = \{\binom{23}{27}, \binom{29}{16}\}$. Since $\{j \in F : w_j < \overline{W}\} = \emptyset$ node N_3 is fathomed.
   Node N_4 is created with $t = 4$, $\mathcal{B}_1 = \{1,2\}$, $\mathcal{B}_0 = \{3,4\}$, and $\mathcal{F} = \{5\}$.
   $\mathcal{N} = \{N_0, N_2, N_4\}$.

5. We select N_4 and compute $\overline{W} = 10$, $\underline{z} = \binom{16}{11}$ and $\overline{z} = \binom{16+3}{11+4} = \binom{19}{15}$. $\overline{z}$ is dominated, so node N_4 fathomed.
   Node N_5 is created with $t = 2$, $\mathcal{B}_1 = \{1\}$, $\mathcal{B}_0 = \{2\}$, and $\mathcal{F} = \{3, 4, 5\}$.
   $\mathcal{B}_1$ at N_5 is smaller than $\mathcal{B}_1$ at node N_2 and N_2 is fathomed.
   $\mathcal{N} = \{N_0, N_5\}$.
6. Node N_5 is selected. At this node $\overline{W} = 12$, $\underline{z} = 119$. Since $\{j \in F : w_j < \overline{W}\} \neq \emptyset$, $\overline{z} = \binom{27}{27}$ is not dominated.
   Node N_6 is created with $s = 3$, $\mathcal{B}_1 = \{1, 3\}$, $\mathcal{B}_0 = \{2\}$, and $\mathcal{F} = \{4, 5\}$.
   $\mathcal{N} = \{N_0, N_5, N_6\}$.
7. Select N_6. $\overline{W} = 4$ and $\underline{z} = \overline{z} = \binom{18}{25}$ Because $\{j \in \mathcal{F} : w_j < \overline{W}\} = \emptyset$ node N_6 is fathomed.
   We create N_7 with $t = 3$, $\mathcal{B}_1 = \{1\}$, $\mathcal{B}_0 = \{2, 3\}$, $\mathcal{F} = \{4, 5\}$.
   $\mathcal{N} = \{N_0, N_5, N_7\}$.
8. Select N_7. $\overline{W} = 12$, $\underline{z} = \binom{11}{9}$. $\{j \in \mathcal{F} : w_j < \overline{W}\} \neq \emptyset$. Upper bound $\overline{z} = \binom{27}{18}$ is not dominated.
   We create N_8 with $s = 5$, $\mathcal{B}_1 = \{1, 4, 5\}$, $\mathcal{B}_0 = \{2, 3\}$, and $\mathcal{F} = \emptyset$.
   $\mathcal{N} = \{N_0, N_5, N_7, N_8\}$
9. Node N_8 is selected. $\overline{W} = 0$ and $\underline{z} = \overline{z} = \binom{27}{18}$ is added to $\mathcal{L}$ to give $\mathcal{L} = \{\binom{23}{27}, \binom{29}{16}, \binom{27}{18}\}$. Since obviously $\{j \in F : w_j < \overline{W}\} = \emptyset$ node N_8 fathomed.
   Node N_9 is created with $t = 5$, $u = 4$, $v = 1$, and $\mathcal{B}_1 = \emptyset$, $\mathcal{B}_0 = \{1\}$, $\mathcal{F} = \{2, 3, 4, 5\}$.
   $\mathcal{B}_1$ at N_9 is smaller than $\mathcal{B}_1$ at N_7 and N_5 so that N_5, N_7 are fathomed.
   $\mathcal{N} = \{N_0, N_9\}$.
10. Select N_9. $\overline{W} = 16$, $\underline{z} = \binom{0}{0}$. To compute $\overline{z}$ use $s_1 = 4$, $s_2 = 5$ so that

$$\overline{z} = \begin{pmatrix} 0 + 5 + 13 & + \max\{[7\frac{3}{5}], [7 - (8-7)\frac{13}{7}]\} \\ 0 + 16 + 2 + 4 & + \max\{[0], [5 - (7-1)\frac{4}{5}]\} \end{pmatrix} = \begin{pmatrix} 23 \\ 22 \end{pmatrix}.$$

    The upper bound $\overline{z}$ is dominated, N_9 is fathomed. Because $\mathcal{B}_1 = \emptyset$ at N_9, N_0 can be fathomed. Thus $\mathcal{N} = \emptyset$ and the algorithm stops.

Graphically, the solution process can be depicted as in Figure 10.1. Each node shows the node number and the sets $\mathcal{B}_1$ and $\mathcal{B}_0$. $\mathcal{L}$ is shown for a node whenever it is updated at that node.

There are three efficient solutions, x^1 with $x_1 = x_2 = x_3 = 1$, x^2 with $x_1 = x_2 = x_4 = 1$, and x^3 with $x_1 = x_4 = x_5 = 1$. □

Algorithm 10.1 finds a complete set of efficient solutions. If no duplicates of nondominated points are kept in $\mathcal{L}$ it will be a minimal complete set, otherwise the maximal complete set.

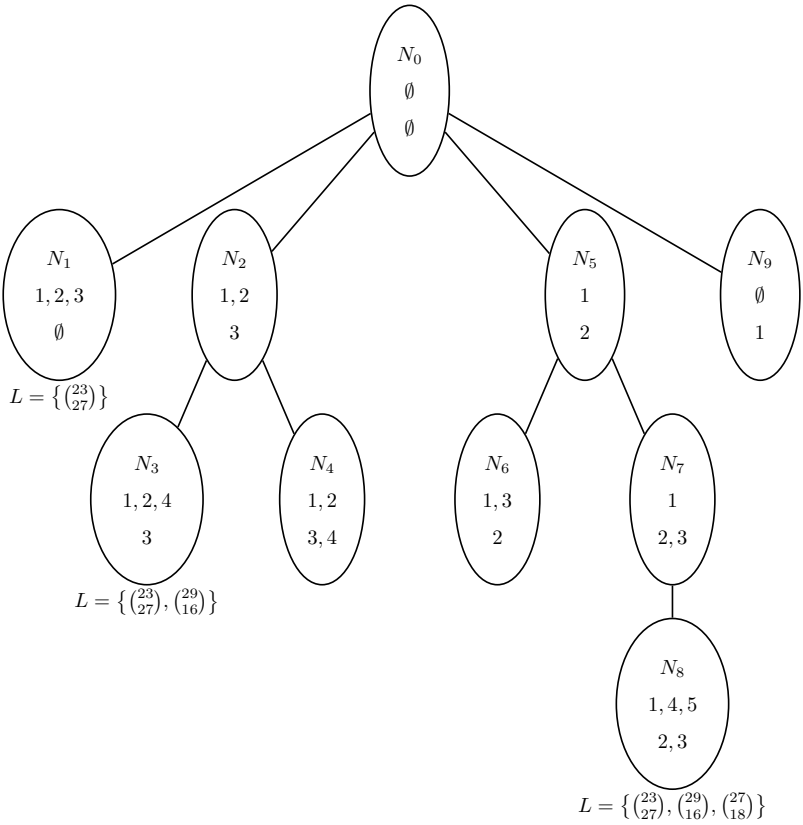

Fig. 10.1. Branch and bound tree in Example 10.1.

10.2 The Travelling Salesperson Problem and Heuristics

The travelling salesperson problem (TSP) consists in finding a shortest tour through n cities. Given a distance matrix $C = (c_{ij})$, find a cyclic permutation π of $\{1, \ldots, n\}$ such that $\pi(n) = 1$ and $\sum_{i=1}^{n} c_{i\pi(i)}$ is minimal. It can also be formulated as an optimization problem on a complete graph. Let $\mathcal{K}_n = (\mathcal{V}, \mathcal{E})$ be a graph and $C : \mathcal{E} \to \mathbb{R}_{\geqq}$ be a cost (or distance) function on the edges. The TSP is to find a simple cycle that visits every vertex exactly once and has the smallest possible total cost (distance). Such cycles are called Hamiltonian cycles, and we shall use the notation HC.

In the multicriteria case we have p distance matrices $C^k, k = 1, \ldots, p$ and the problem is to find cyclic permutations of $\{1, \ldots, n\}$ that minimize $(f_1(\pi), \ldots, f_p(\pi))$, where $f_k(\pi) = \sum_{i=1}^{n} c^k_{i\pi(i)}$. The problem is $\mathcal{NP}$-hard for one objective, so also in the multicriteria case. $\#\mathcal{P}$-completeness is open, but we can prove intractability.

Proposition 10.2 (Emelichev and Perepelitsa (1992)). *The multicriteria TSP is intractable, even if $p = 2$.*

Proof. We consider the TSP on the graph $\mathcal{G} = \mathcal{K}_n$ with edge set $\{e_1, \ldots, e_{n(n-1)/2}\}$ and assign the costs $c(e_i) = (2^i, 2^{n^2} - 2^i)$. As shown earlier (Theorem 9.37) all feasible solutions have incomparable weights. Because there are $(n-1)!$ feasible solutions, the claim follows. □

Methods for finding efficient solutions usually imply solving many single objective TSPs, i.e. $\mathcal{NP}$-hard problems. The TSP is therefore well suited to establish and exemplify results on approximation algorithms for multiobjective combinatorial optimization problems. The results discussed in this section have been obtained in Ehrgott (2000).

To illustrate the idea of approximation algorithms, we first review briefly approximation algorithms for single objective combinatorial optimization problems $\min_{x \in \mathcal{X}} f(x)$.

Let $x \in \mathcal{X}$ be any feasible solution and $\hat{x} \in \mathcal{X}$ be an optimal solution. Then $R(x, \hat{x}) := f(x)/f(\hat{x}) \geq 1$ is called the *performance ratio* of x with respect to $\hat{x}$.

A polynomial time algorithm A for the problem is called an $r(n)$-*approximation algorithm*, if $R(A(I), \hat{x}) \leq r(|I|)$ for all instances I of the problem, where $A(I)$ is the solution found by A, $|I|$ denotes the size of the problem instance and $r : \mathbb{N} \to [1, \infty]$ is a function. $r(n) \equiv 1$ means that the problem is solvable in polynomial time by algorithm A. Note that $R(x, \hat{x}) = \varrho$ is equivalent to

$$\frac{f(x) - f(\hat{x})}{f(\hat{x})} = \varrho - 1.$$

We will investigate if it is possible to find *one* solution that is a good approximation for all efficient solutions of a multiobjective combinatorial optimization problem. In order to generalize the definition of approximation ratios, we will use norms. Let $\|\cdot\| : \mathbb{R}^p \to \mathbb{R}_{\geqq}$ be a norm. We say that $\|\cdot\|$ is monotone, if for $y^1, y^2 \in \mathbb{R}^p$ with $|y^1_k| \leq |y^2_k|$ for all $k = 1, \ldots, p$ we have $\|y^1\| \leq \|y^2\|$ (see Definition 4.19).

Now consider a multiobjective combinatorial optimization problem.

Definition 10.3. *Let $x \in \mathcal{X}$, and $\hat{x} \in \mathcal{X}_E$.*

1. *The* performance ratio R_1 *of x with respect to x^* is*

$$R_1(x, \hat{x}) := \frac{|\ \|f(x)\| - \|f(\hat{x})\|\ |}{\|f(\hat{x})\|}.$$

Algorithm A that finds a feasible solution of the MOCO problem is an $r_1(n)$-approximation algorithm if

$$R_1(A(I), \hat{x}) \leq r_1(|I|)$$

for all instances I of the problem and for all efficient solutions of that instance of the MOCO problem.

2. *The* performance ratio R_2 *of x with respect to $\hat{x}$ is*

$$R_2(x, \hat{x}) := \frac{|\ \|f(x) - f(x^*)\|\ |}{\|f(x^*)\|}.$$

Algorithm A that finds a feasible solution of the MOCO problem is an $r_2(n)$-approximation algorithm if

$$R_2(A(I), \hat{x}) \leq r_2(|I|)$$

for all instances I of the problem and for all efficient solutions of that instance of the MOCO problem.

We have some general results on approximation of efficient solutions.

Corollary 10.4. *An $r(n)$-approximation algorithm according to R_2 is an $r(n)$-approximation algorithm according to R_1.*

Proof. If $R_2(x, \hat{x}) \leq \varrho$ then also $R_1(x, \hat{x}) \leq \varrho$. □

Since we actually compare the norms of the objective vectors of a heuristic solution x and efficient solutions $\hat{x}$, a straightforward idea is to use a feasible solution whose objective vector has minimal norm as an approximate solution. This approach gives a performance ratio of at most 1.

Theorem 10.5. *Let $x^n \in \mathcal{X}$ be such that $\|f(x^n)\| = \min_{x\in\mathcal{X}} \|f(x)\|$ and let $\hat{x}$ be efficient. Then*

$$R_1(x^n, \hat{x}) \leq 1.$$

Proof.

$$R_1(x^n, \hat{x}) = \frac{|\ \|f(x^n)\| - \|f(\hat{x})\|\ |}{\|f(\hat{x})\|} = \frac{\|f(\hat{x})\| - \|f(x^n)\|}{\|f(\hat{x})\|} \leq 1.$$

□

Note that there always exists some x^n with minimal norm $\|f(x^n)\|$, which is also efficient optimal. This can be seen from Theorem 4.20, using $y^U = 0$ as reference point, which is possible because distances c_{ij}^p are nonnegative.

With Theorem 10.5 two questions arise: Is the bound tight and can x^n be computed efficiently? The answer to the first question is given by an example.

Example 10.6. Let $\mathcal{E} = \{e_1, e_2, e_3, 3_4\}$ and $\mathcal{X} = \{x \subset \mathcal{E} : |x| = 2\}$. The costs of all $e \in \mathcal{E}$ are $c(e_1) = (M, 0), c(e_2) = (0, M), c(e_3) = c(e_4) = (1, 1)$, where M is a large number.

The efficient solutions are $\{e_1, e_3\}, \{e_1, e_4\}, \{e_2, e_3\}, \{e_2, e_4\}$, and $\{e_3, e_4\}$. The solution with minimal norm is $x^n = \{e_3, e_4\}$. Computing performance ratios we obtain

$$R_1(\{e_3, e_4\}, \{e_1, e_3\}) = \frac{|\ \|(2, 2)\| - \|(M + 1, 1)\|\ |}{\|(M + 1, 1)\|} \to 1$$

as $M \to \infty$ and

$$R_2(\{e_3, e_4\}, \{e_1, e_3\}) = \frac{|\ \|(2, 2) - (M + 1, 1)\|\ |}{\|(M + 1, 1)\|} \to 1$$

as $M \to \infty$. This example shows that the bound of 1 for the approximation ratio cannot be improved in general. □

For the second question, we can state two sufficient conditions which guarantee the existence of polynomial time algorithms to compute x^n.

Proposition 10.7. *The problem $\min_{x\in\mathcal{X}} \|f(x)\|$ can be solved in polynomial time if one of the two conditions below is satisfied.*

1. *$(\mathcal{X}, 1\text{-}\sum, \mathbb{Z})/\operatorname{id}/(\mathbb{Z}, <)$ can be solved in polynomial time and $\|\cdot\| = \|\ \|_1$.*
2. *$(\mathcal{X}, 1\text{-}\max, \mathbb{Z})/\operatorname{id}/(\mathbb{Z}, <)$ can be solved in polynomial time and $\|\cdot\| = \|\cdot\|_\infty$.*

Proof. 1. In the first case,

$$\begin{aligned}\min_{x\in\mathcal{X}} \|f(x)\| &= \min_{x\in\mathcal{X}} \sum_{k=1}^{p} f_k(x)\\ &= \min_{x\in\mathcal{X}} \sum_{k=1}^{p} \sum_{e\in x} c^k(e)\\ &= \min_{x\in\mathcal{X}} \sum_{e\in x} \sum_{k=1}^{p} c^k(e)\\ &= \min_{x\in\mathcal{X}} \sum_{e\in x} \hat{c}(e)\end{aligned}$$

where $\hat{c}(e) = \sum_{k=1}^{p} c^k(e)$.

2. In the second case,

$$\begin{aligned}\min_{x\in\mathcal{X}} \|f(x)\| &= \min_{x\in\mathcal{X}} \max_{k=1,\ldots,p} f_k(x)\\ &= \min_{x\in\mathcal{X}} \max_{k=1,\ldots,p} \max_{e\in x} c^k(e)\\ &= \min_{x\in\mathcal{X}} \max_{e\in x} \max_{k=1,\ldots,p} c^k(e)\\ &= \min_{x\in\mathcal{X}} \max_{e\in x} \hat{c}(e),\end{aligned}$$

where $\hat{c}(e) = \max_{k=1}^{p} c^k(e)$.

Under the assumptions of the proposition, these problems are solvable in polynomial time. □

After these general results on approximability, we turn attention to the multicriteria TSP again. We consider two well known heuristic methods for the single objective problem, and analyze their performance for the multiobjective TSP. To apply these methods, we have to assume that the distances satisfy the triangle inequality and are symmetric, i.e. $c_{ij}^k \le c_{il}^k + c_{lj}^k$ and $c_{ij}^k = c_{ji}^k$ for all i, j, k, l.

The first heuristic generalizes the tree heuristic, which generates a tour from a spanning tree via an Eulerian tour. A Eulerian tour of a graph $\mathcal{G}$ is an alternating sequence of nodes and edges with identical first and last node, which contains each edge of $\mathcal{G}$ exactly once. It is well known that a graph $\mathcal{G}$ is Eulerian (i.e. has a Eulerian tour) if and only if each node has even degree, see e.g. (Papadimitriou and Steiglitz, 1982, p. 412) for a proof.

Algorithm 10.2 (Tree heuristic for the TSP.)

Input: Distance matrices $C^k, k = 1, \ldots, p$.

Find $ST \in \operatorname{argmin}\{\|f(T)\| : T$ is a spanning tree of $\mathcal{K}_n\}$.

Define $\mathcal{G} := (\mathcal{V}(\mathcal{K}_n), \mathcal{E})$, where $\mathcal{E}$ consists of two copies of every $e \in \mathcal{E}(ST)$

Find a Eulerian tour embedded in $\mathcal{G}$, and the corresponding TSP tour HC by eliminating duplicate nodes in the Eulerian tour.

Output: A TSP tour HC.

Note that the graph $\mathcal{G}$, which has two copies of each edge of the spanning tree ST is Eulerian because each node is incident to an even number of edges. From the Eulerian tour a TSP tour HC can be constructed through "shortcuts". This is where the triangle inequality for the cost functions is important.

Example 10.8. We apply the algorithm to a TSP with three objectives. The distance matrices are

$$C^1 = \begin{pmatrix} - & 1 & 6 & 5 & 5 & 5 \\ 1 & - & 5 & 4 & 6 & 4 \\ 6 & 5 & - & 5 & 6 & 1 \\ 5 & 4 & 5 & - & 6 & 8 \\ 5 & 6 & 6 & 6 & - & 2 \\ 5 & 4 & 1 & 8 & 2 & - \end{pmatrix},$$

$$C^2 = \begin{pmatrix} - & 57 & 55 & 24 & 19 & 46 \\ 57 & - & 151 & 126 & 121 & 137 \\ 55 & 151 & - & 121 & 90 & 117 \\ 24 & 126 & 121 & - & 34 & 61 \\ 19 & 121 & 90 & 34 & - & 27 \\ 46 & 137 & 117 & 61 & 27 & - \end{pmatrix},$$

$$C^3 = \begin{pmatrix} - & 39 & 173 & 6 & 249 & 45 \\ 39 & - & 354 & 348 & 430 & 25 \\ 173 & 354 & - & 511 & 76 & 404 \\ 6 & 348 & 511 & - & 251 & 39 \\ 249 & 430 & 76 & 251 & - & 328 \\ 45 & 25 & 404 & 39 & 328 & - \end{pmatrix}.$$

Using either the l_2- or the l_1-norm, we get the tree of Figure 10.2 with objective vector $f(ST) = (23, 272, 339)$. Its l_1-norm is 634, the l_2-norm is 435.24. So $\mathcal{G}$ is the graph shown in Figure 10.3.

One possible TSP tour HC, $(1, 2, 4, 3, 5, 6, 1)$ is indicated by the broken lines. The objective function value is $f(HC) = (18, 467, 1879)$ and HC is indeed efficient, which can be verified in this small example by complete enumeration. □

The multicriteria tree heuristic has the theoretically best performance ratio of 1.

Proposition 10.9. *Algorithm 10.2 is a 1-approximation algorithm according to performance ratio R_1.*

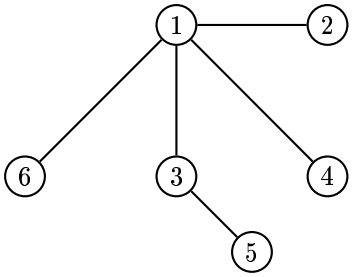

Fig. 10.2. Minimum norm spanning tree ST of $\mathcal{K}_6$.

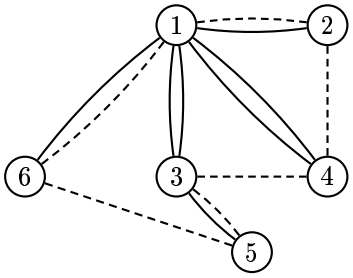

Fig. 10.3. $\mathcal{G}$ is obtained by duplicating all edges of ST.

Proof. Let HC be the tour found by Algorithm 10.2 and let $\hat{HC}$ be an efficient tour. We show that

$$-\|f(\hat{HC})\| \;\leq\; \|f(HC)\| - \|f(\hat{HC})\| \;\leq\; \|f(\hat{HC})\|. \tag{10.8}$$

The left inequality is trivial. To prove the right hand one, we first apply the the triangle inequality, which gives

$$f(HC) \leq 2f(ST) = f(G).$$

From monotonicity of the norm $\|\cdot\|$ we now conclude

$$\|f(C)\| \leq 2\|f(ST)\|. \tag{10.9}$$

By the choice of ST and since deleting an edge from $\hat{HC}$ gives a spanning tree we also have

$$\|f(ST)\| \leq \|f(\hat{HC})\|. \tag{10.10}$$

Combining (10.9) and (10.10) we conclude

$$\|f(HC)\| \leq 2\|f(\hat{HC})\|.$$

□

We can improve Algorithm 10.2 by adding copies of fewer edges in ST when creating $\mathcal{G}$. All we need is to add as many edges as necessary to be sure that all edges in $\mathcal{G}$ have even degree. Thus, we get a multiobjective version of Christofides' algorithm (Papadimitriou and Steiglitz, 1982).

Algorithm 10.3 (Christofides' heuristic for the TSP.)
Input: Distance matrices $C^k, k = 1, \ldots, p$.
Find $ST \in \operatorname{argmin}\{\|f(T)\| : T$ *is a spanning tree of* $\mathcal{K}_n\}$.
Let $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{E}^*)$, *where* $\mathcal{V}^* = \{v : v$ *has odd degree in* $ST\}$, *and* $\mathcal{E}^* = \{[u, v] : u, v \in \mathcal{V}^*\}$.
Find $PM \in \operatorname{argmin}\{\|f(M)\| : M$ *is a perfect matching in* $\mathcal{G}^*\}$.
$\mathcal{G} = (\mathcal{V}, \mathcal{E}(ST) \cup \mathcal{E}(PM))$.
Find a Eulerian tour of $\mathcal{G}$, *and the embedded TSP tour* HC.
Output: A TSP tour HC.

Instead of duplicating all edges of ST only additional edges between those nodes that have odd degree in ST are added. There is always an even number of nodes with odd degree in a spanning tree (because the sum of the degrees of all nodes is even). Therefore $\mathcal{G}^*$ is a complete graph on an even number of nodes. Thus $\mathcal{G}^*$ has a perfect matching with $|\mathcal{V}^*|/2$ edges.

Example 10.10. We apply Algorithm 10.3 to the instance presented in Example 10.8. With the same spanning tree of minimal norm as before (Figure 10.2) nodes $2, 4, 5$, and 6 have odd degree. The (unique) perfect matching with

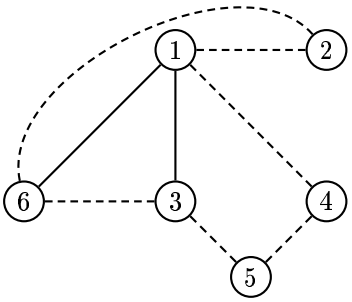

Fig. 10.4. Eulerian graph $\mathcal{G}$ obtained by Algorithm 10.3.

minimal norm for both the l_1 and l_2 norm is $PM = \{[2,6],[4,5]\}$. We get $\mathcal{G}$ as shown in Figure 10.4.

A TSP tour that can be extracted from $\mathcal{G}$ is $(1,2,6,3,5,4,1)$ with objective function values $(23,459,801)$, which is also an efficient TSP tour. □

The Christofides' heuristic has a performance ratio of 1/2 in the single objective case. This can no longer hold for multiple criteria. But it should not come as a surprise, that it also has the performance ratio 1.

Proposition 10.11. *Algorithm 10.3 is a 1-approximation algorithm for the multicriteria TSP according to performance ratio R_1.*

Proof. Let HC be the TSP tour found by Algorithm 10.3 and let $\hat{HC}$ be an efficient TSP tour. We show (10.8) again.

Let $\{i_1,\ldots,i_{2m}\}$ be the odd degree nodes in ST in the order they appear in $\hat{HC}$, i.e.

$$\hat{HC} = (\alpha_0, i_1, \alpha_1, i_2, \ldots, \alpha_{2m-1}, i_{2m}, \alpha_{2m}),$$

where α_i are sequences of other nodes.

$M_1 = \{[i_1,i_2],[i_3,i_4],\ldots,[i_{2m-1},i_{2m}]\}$ and $M_2 = \{[i_2,i_3],\ldots,[i_{2m},i_1]\}$ are two perfect matchings on the nodes $\{i_1,\ldots,i_{2m}\}$. By the triangle inequality we get $f(\hat{HC}) \geq f(M_1) + f(M_2)$ and by definition of PM $\|f(M_i)\| \geq \|f(PM)\|$ for $i = 1,2$. Therefore

$$\begin{aligned}\|f(\hat{HC})\| &\geq \|f(M_1) + f(M_2)\| \\ &\geq \max\{\|f(M_1)\|, \|f(M_2)\|\} \;\geq\; \|f(PM)\|. \qquad (10.11)\end{aligned}$$

On the other hand

$$\begin{aligned}\|f(C)\| \leq \|f(G)\| &= \|f(ST) + f(PM)\| \\ &\leq \|f(ST)\| + \|f(PM)\|. \end{aligned} \tag{10.12}$$

Putting (10.11) and (10.12) together we obtain

$$\|f(HC)\| \leq \|f(ST)\| + \|f(PM)\| \leq \|f(ST)\| + \|f(\hat{HC})\| \leq 2\|f(\hat{HC}\|,$$

because, as in the proof of Proposition 10.11,

$$\|f(\hat{HC})\| \ \geq \ \|f(ST)\|.$$

□

Note that if $p = 1$, $\|f(x)\| = f(x)$, and (10.11) can be strengthened to $f(\hat{HC}) \geq 2f(PM)$ which gives $f(HC) \leq 3f(\hat{HC})$ and $R_1(HC, \hat{HC}) = 1/2$. The reason why no better result is obtained in the multicriteria case is the maximum of $\|f(M_1)\|$ and $\|f(M_2\|$. We cannot replace this by the sum of the two terms in general.

To prove the approximation result for approximation ratio R_2, we restrict ourselves to l_{p}-norms

$$\|y\|_{\mathsf{p}} = \left(\sum_{k=1}^{\mathsf{p}} |y_k|^p \right)^{\frac{1}{\mathsf{p}}}.$$

Theorem 10.12 (Ehrgott (2000)). *Algorithms 10.2 and 10.3 are* $(2^p+1)^{\frac{1}{p}}$*-approximation algorithms according to performance ratio* R_2*.*

Proof. Let HC be the TSP tour found by either Algorithm 10.2 or Algorithm 10.3 and let $\hat{HC}$ be an efficient TSP tour.

$$\frac{\|f(HC) - f(\hat{HC})\|}{\|f(\hat{HC})\|} = \frac{\left(\sum_{k=1}^{p} |f_k(HC) - f_k(\hat{HC})|^{\mathsf{p}}\right)^{\frac{1}{\mathsf{p}}}}{\left(\sum_{k=1}^{p} (f_k(\hat{HC}))^{\mathsf{p}}\right)^{\frac{1}{\mathsf{p}}}}$$

$$\leq \frac{\left(\sum_{k=1}^{p} ((f_k(HC))^{\mathsf{p}} + (f_k(\hat{HC}))^{\mathsf{p}})\right)^{\frac{1}{\mathsf{p}}}}{\left(\sum_{k=1}^{p} (f_k(\hat{HC}))^{\mathsf{p}}\right)^{\frac{1}{\mathsf{p}}}} \tag{10.13}$$

$$= \left(\frac{\|f(HC)\|^{\mathsf{p}} + \|f(\hat{HC})\|^{\mathsf{p}}}{\|f(\hat{HC})\|^{\mathsf{p}}}\right)^{\frac{1}{\mathsf{p}}}$$

$$\leq \left(\frac{2^{\mathsf{p}}\|f(\hat{HC})\|^{\mathsf{p}} + \|f(\hat{HC})\|^{\mathsf{p}}}{\|f(\hat{HC})\|^{\mathsf{p}}}\right)^{\frac{1}{\mathsf{p}}} \tag{10.14}$$

$$= (2^{\mathsf{p}} + 1)^{\frac{1}{\mathsf{p}}} \tag{10.15}$$

For inequality (10.13) we used the crude estimate

$$\left(\sum_{k=1}^{p} |y_k^1 - y_k^2|^{\mathsf{p}}\right)^{\frac{1}{\mathsf{p}}} \leq \left(\sum_{k=1}^{p} ((y_k^1)^{\mathsf{p}} + (y_k^2)^{\mathsf{p}})\right)^{\frac{1}{\mathsf{p}}}.$$

Inequality (10.14) follows from the fact $\|f(HC)\| \leq 2\|f(\hat{HC})\|$ from the proofs of Proposition (10.9) and 10.11, which are true for any monotone norm. □

Even though the theoretical bounds for Algorithms 10.2 and 10.3 are the same, 10.3 will often yield better results in practice, see Exercise 10.4, which continues Examples 10.8 and 10.10. A more detailed analysis of Algorithms 10.2 and 10.3 can be found in Ehrgott (2000). For instance, the bound of Theorem 10.12 can be improved, when the l_1-norm is used. The reader is asked to obtain this better bound in Exercise 10.3

10.3 Notes

References on multiobjective versions of $\mathcal{NP}$-hard combinatorial optimization problems are fewer than for polynomially solvable ones.

The most popular is the knapsack problem. Apart from the branch and bound algorithm presented here algorithms based on dynamic programming are known (e.g. Eben-Chaime (1996); Klamroth and Wiecek (2000)). Heuristics and metaheuristics to approximate $\mathcal{X}_E$ are found in Gandibleux and

Fréville (2000); Hansen (1998); Safer and Orlin (1995); Salman *et al.* (1999). Metaheuristics have also been used to solve multi-constraint knapsack problems (Jaszkiewicz, 2001; Zitzler and Thiele, 1999).

Some further references on the TSP are Fischer and Richter (1982); Hansen (2000); Melamed and Sigal (1997). A few references must suffice to indicate that other problems have also been addressed, e.g. set partitioning problems in Ehrgott and Ryan (2002) and location problems in Fernández and Puerto (2003). For scheduling problems there is a vast amount of literature, see T'Kindt and Billaut (2002) and in Chapter 8 of Ehrgott and Gandibleux (2002b).

Exercises

10.1. Let x^1 and x^2 be two optimal solutions of the weighted sum knapsack problem

$$\min \lambda \sum_{i=1}^{n} c_i^1 x_i + (1-\lambda) \sum_{i=1}^{n} c_i^2 x_i$$
$$x_i \in \{0,1\}; \quad i = 1, \ldots, n$$

with $0 < \lambda < 1$. Let $x^{MT} \in [0,1]^n$ be a vector which attains the Martello-Toth bound (10.7) for this single objective knapsack problem. Show that $(\sum_{i=1}^{n} c_i^1 x_i^{MT}, \sum i = 1^n c_i^2 x_i^{MT})$ is an upper bound for all efficient solutions of the bicriterion knapsack problem with $\sum_{i=1}^{n} c_i^1 x_i^1 \leq \sum_{i=1} c_i^1 x_i \leq \sum_{i=1}^{n} c_i^1 x_i^2$ and $\sum_{i=1}^{n} c_i^2 x_i^2 \leq \sum_{i=1} c_i^2 x_i \leq \sum_{i=1}^{n} c_i^2 x_i^1$.

10.2. Solve the following bicriterion knapsack problem using Algorithm 10.1.

$$\begin{array}{lll} \max & 10x_1 + 3x_2 + 6x_3 + 8x_4 + 2x_5 & \\ \max & 12x_1 + 9x_2 + 11x_3 + 5x_4 + 6x_5 & \\ \text{subject to} & 4x_1 + 5x_2 + 2x_3 + 5x_4 + 6x_5 \leq 17 & \\ & x_j \in \{0,1\}, \quad j = 1, \ldots, 5. & \end{array}$$

10.3. Compute the approximation ratio $r_2(n)$ of Algorithms 10.2 and 10.3 explicitly when the l_1-norm or the l_∞-norm is used. For the l_1 norm you should obtain a better result than that of Theorem 10.12.

10.4. To see that the Christofides' heuristic (Algorithm 10.3) may yield much better results than the tree algorithm in practice, despite their having the same worst case approximation ratios, compute the actual deviations of all possible heuristic TSP tours from the efficient TSP tours. See Figures 10.3 and 10.4. There are seven efficient solutions.