The Quadratic Selective Travelling Salesman Problem

Tommy Thomadsen* Thomas Stidsen†

Informatics and Mathematical Modelling
Technical University of Denmark
DK-2800 Kongens Lyngby, Denmark

August 14, 2003

Abstract

A well-known extension of the Travelling Salesman Problem (TSP) is the Selective TSP (STSP): Each node has an associated profit and instead of visiting all nodes, the most profitable set of nodes, taking into account the tour cost, is visited. The Quadratic STSP (QSTSP) adds the additional complication that each *pair* of nodes have an associated profit which can be gained only if *both* nodes are visited. The QSTSP is a subproblem when constructing hierarchical ring networks.

We describe an integer linear programming model for the QSTSP. The QSTSP is solved by two construction heuristics and by a branch-and-cut algorithm. Computational results are presented for two budget constraints limiting the length of the ring and the number of nodes in the ring respectively. It is investigated how the tightness of the budgets affect the running time of the branch-and-cut algorithm and correspondingly how this affect the solution quality of the construction heuristics.

The construction heuristics have difficulties in finding good solutions. One heuristic is best when tight budgets are considered, the other heuristic is best when loose budgets are considered. The branch-and-cut algorithm finds the optimal solutions at a cost of much higher running time. All problems with up to 50 nodes are solved within one hour.

Keywords: Ring Networks, (Selective) Traveling Salesman Problem, Branch-and-Cut, Integer Programming

*Email: `tt@imm.dtu.dk`
†Email: `tks@imm.dtu.dk`

1 Introduction

When building telecommunication networks, the ring topology is widely used due to its inherent single link/node breakdown protection and the simple and fast restoration scheme. When a service provider choose to build a ring, it is of major importance to build a ring which gives a good tradeoff between the service provided (i.e. gives a revenue) and the node- and link-costs of building the ring. The ring chosen should also be implementable in the technology chosen, e.g. the number of nodes and/or the length of the ring is limited due to delay problems when transmitting or robustness requirements (the number of links increases with the number of nodes and after all only *single* link protection is provided by a ring).

Single-ring design can be generalized to multi-ring design where the service provider is contracted to bring service to a set of customers using more rings. In that case the same tradeoff is apparent, but for more rings at the same time. Evaluating the quality of single rings is at least a good start for such designs. In particular one could imagine generating a set of potentially good rings and choosing a subset of those which cover all customers.

The single-ring design can be seen as an extension to the classical Travelling Salesman Problem (TSP) and it has been termed, e.g. the Orienteering problem, Prize collecting TSP and Selective TSP (STSP). We will use the term STSP. In the STSP, a revenue is associated with each customer and the idea is then to establish a good tradeoff between the length travelled and the revenue obtained for visiting nodes. STSP is studied in e.g. [2], [3], [9], [11] and [12]. A related problem is the Generalized TSP (GTSP) which is considered in [6] and [7]. In GTSP the set of nodes is divided into disjoint subsets and the shortest ring which passes at least one node in each subset is to be determined.

The most notable difference between the different versions of STSP and GTSP is the way a limit is put on the size of the ring. (The limit may be a minimum size or a maximum size determined directly or indirectly.) Some of the possibilities (they are sometimes combined) are: 1) a tradeoff in distance travelled and revenue collected (i.e. both the distance travelled and the revenue appear in the objective) [2, 3] 2) A lower bound on the revenue collected [2, 3] 3) An upper bound on the distance travelled [9, 11, 12] 4) Subset controlled (GTSP) [6, 7].

Another notable difference is whether a particular node (usually denoted the depot) is required to be in the ring [9, 11, 12] or not [2, 3, 6, 7]. In [9], a non-empty subset of the nodes are required nodes. Finally [6] and [7] falls a bit out of category - one of each of the subsets must be selected (which include singleton subsets).

The Quadratic STSP (QSTSP) adds the additional complication that each pair of nodes have an associated profit which can be gained only if *both* nodes are visited. QSTSP is considered in [8] and [10]. In [8] a model for the QSTSP with quadratic objective is presented and heuristics are developed and tested. The size of the ring is controlled by an upper limit on the distance travelled combined with a tradeoff in the objective between the distance travelled and the profit associated with each pair of nodes. No nodes are required to be in the ring. In [10] alternative models for QSTSP are considered. The size of the ring is controlled by a bound on the number of nodes in the ring and also a set of nodes is required. Three integer linear models are evaluated by solving test instances with up to 80 nodes using Cplex. For the test instances, all pairs of nodes have a demand, but only a fixed number (200 for test instances with 80 nodes) of the shortest edges are considered. Some instances cannot be solved within 24 hours.

We present an integer linear programming model for the QSTSP and suggest a branch-and-cut algorithm. Given the similarities with GTSP and STSP, cuts and corresponding separation-routines found in [7] are very useful.

The outline of the paper is the following. We present the model for QSTSP in section 2. In section 3 we describe the branch-and-cut algorithm and in section 4 we describe two construction heuristics. In section 5 we test the algorithms with two different budget constraints limiting the length of the ring and the number of nodes in the ring respectively. Finally we give conclusions in section 6.

2 The Model

In this section we present an integer linear programming model for the QSTSP problem. The model is in essence a linearization of the model presented in [8]. We restrict ourselves to the undirected/symmetric version of QSTSP as is the case in [8] and [10].

Let $V = \{1, \ldots, n\}$ be a set of vertices and $E = \{\{i, j\} : i \in V, j \in V \backslash \{i\}\}$ be a set of node-pairs corresponding to (undirected) edges. Let r_i be the revenue (cost if negative) obtained if $i \in V$ is on the ring. Associate with each node-pair $\{i, j\} = e$ a cost c_e incurred if e is chosen and a revenue r_e obtained if i and j are on the ring.

Let $S \subseteq V$ and define $\delta(S) \subseteq E$ to be the set of edges with one endpoint in S and one endpoint not in S - the cut-set of S. Also define $\gamma(S) \subseteq E$ to be the set of edges with both endpoints in S:

$$\begin{aligned} \delta(S) &= \{\{i, j\} \in E : i \in S, j \notin S \vee i \notin S, j \in S\} \\ \gamma(S) &= \{\{i, j\} \in E : i \in S, j \in S\} \end{aligned}$$

$\delta(\{i\})$ is abbreviated $\delta(i)$.

Let $x_e = 1$ if $e \in E$ is chosen in the ring, 0 otherwise. Let $y_i = 1$ if $i \in V$ is on the ring, 0 otherwise. And finally let $z_e = 1$, $\{i, j\} = e \in E$ if i and j are both on the ring. Given this, QSTSP is formulated as follows.

$$\text{Maximize} \qquad \sum_{i \in V} r_i \cdot y_i + \sum_{e \in E} r_e \cdot z_e - \sum_{e \in E} c_e \cdot x_e \tag{1}$$

Subject To

$$\sum_{e \in \delta(i)} x_e = 2y_i \text{ for } i \in V \tag{2}$$

$$z_e \leq y_i \text{ for } i \in V, e \in \delta(i) \tag{3}$$

$$z_e \geq y_i + y_j - 1 \text{ for } \{i,j\} = e \in E \tag{4}$$

$$\sum_{e \in \gamma(S)} x_e \leq \sum_{i \in S \setminus \{k\}} y_i - y_l + 1 \text{ for } \emptyset \subset S \subset V, k \in S, l \notin S \tag{5}$$

$$x_e \in \{0,1\} \text{ for } e \in E \tag{6}$$

$$y_i \in \{0,1\} \text{ for } i \in V \tag{7}$$

$$z_e \in \{0,1\} \text{ for } e \in E \tag{8}$$

Constraints (2) ensures that if i is on the ring, then exactly two edges are incident to i, otherwise no edges are incident to i. Constraints (3) and (4) connects the y and z variables. (3) ensures that $z_e = 0$ if either of the endpoint nodes are not on the ring and (4) ensures that if both endpoint nodes are on the ring, then $z_e = 1$.

Constraints (5) are the Generalized Subtour Elimination Constraints (GSECs) in inner form [6, 7]. The GSECs ensure that at most one subtour exists. Assume S is the nodes of a subtour and assume that a node in another subtour $l \notin S$ exists. For any $k \in S$, the corresponding GSEC cut is violated, since $\sum_{e \in \gamma(S)} x_e = |S| > \sum_{i \in S \setminus \{k\}} y_i - y_l + 1 = |S| - 1 - 1 + 1 = |S| - 1$ and thus cuts of such solutions. Thus in general GSECs exists which cut of any solution with more than one subtour.

Finally constraints (6), (7) and (8) are the integer domain constraints.

The GSECs can be reformulated in outer form (or cut form). The GSECs in outer form look as follows.

$$\sum_{e \in \delta(S)} x_e \geq 2(y_k + y_l - 1) \text{ for } \emptyset \subset S \subset V, k \in S, l \notin S \tag{9}$$

The reformulation is obtained by using constraints (2). The constraint is easier to interpret in outer form: If a node is selected on both sides of a cut, then at least two edges crossing the cut must be selected. Actually it suffice to consider GSECs such that $|S| \leq \lfloor n/2 \rfloor$. To see why, assume that a GSEC with $|S'| > \lfloor n/2 \rfloor, S \subset V$, $k' \in S'$ and $l \notin S'$ exists. A corresponding GSEC is: $S = V \setminus S'$ (hence $|S| \leq \lfloor n/2 \rfloor$), $k = l'$ and $l = k'$. Obviously this GSEC is violated if and only if the original GSEC is violated, hence it suffice to consider GSECs such that $|S| \leq \lfloor n/2 \rfloor$.

For $|S| \leq \lfloor n/2 \rfloor$, the number of non-zero variables are smaller for GSECs in inner form than in outer form. Hence GSECs in inner form are more suitable for cutting-plane approaches [6] and are thus used.

As noted earlier, it is customary to limit the ring-size in some way, at least for STSP. We will consider two different budget constraints, one which limits the length of the ring, and one which limits the number of nodes in the ring. Assuming b_x is the max length of the ring and b_y is the maximum number of nodes in the ring, the budget constraints are given below.

$$\sum_{e \in E} c_e x_e \leq b_x \tag{10}$$

$$\sum_{i \in V} y_i \leq b_y \tag{11}$$

Constraint (10) limits the length of the ring and is the same budget constraint used in [8]. Constraint (11) limits the number of nodes in a ring and corresponds exactly to a subproblem arising when using column generation to solve a hierarchical multi-ring network design problem similar to the one described in [14].

We note that the c_e's in (10) are the same c_e's as in the objective (1). The following constraint generalizes both (10) and (11):

$$\sum_{e \in E} b_e x_e \leq b \tag{12}$$

Replacing b_e by c_e we obtain (10) and by using (2), we obtain (12) with $b_e = 1$ for all $e \in E$.

If the budget constraint (11) is used (and arbitrary coefficients were allowed to be multiplied on the y_i's in (11)), then QSTSP is actually a generalization of the Quadratic Knapsack Problem (QKP) treated in e.g. [4] and [5].

The mathematical model is similar to several other models suggested in e.g. [2, 6, 7, 9, 11, 12] with the exception of the quadratic revenue. In [8] a model is presented with a quadratic revenue and budget constraint (10). The presented model has a quadratic objective and the model is obtained by replacing z_e with $y_i y_j$ and leaving out the (now redundant) constraints (3) and (4). Also no revenue exists for nodes, i.e. $r_i = 0$.

Integer requirements on z_e are unnecessary, since z_e is really a bookkeeping variable. If $r_e \geq 0$ (we will assume this in the sequel), then constraints (4) are redundant, since we maximize the demand-revenue ($\sum_{e \in E} r_e \cdot z_e$) and thus the constraints will be non-binding.

The advantage of the suggested model over the one given in [8] is clearly that the model is linear; hence the usual tools for solving integer programs can be used. The disadvantage is the additional $(n^2 - n)/2$ variables (z_e), the additional $n^2 - n$ constraints of type (3) and the additional $(n^2 - n)/2$ constraints of type (4).

In [6] the polytope of GTSP is studied and it is proved that GSECs are facet defining for GTSP. Additionally [6] study generalized comb inequalities which are proved to be facet defining for GTSP. A computational study of a branch-and-cut algorithm using these cuts is presented in [7], and showing that the GSECs are by far the most important of these two. In most problems solved, no or very few (≤ 5) generalized combs are generated. With this in mind, we will not consider the generalized combs any further in this paper.

2.1 Additional Cuts

In order to strengthen the LP-relaxation, we add additional cuts cutting off fractional (hence infeasible) solutions feasible in the LP-relaxation.

The following constraints strengthen the LP-relaxation.

$$x_e \leq y_i \text{ for } i \in V, e \in \delta(i) \tag{13}$$

Constraints (13) are valid, since if an edge is in the ring, then both endpoint nodes are selected. Also constraints (13) dominate the GSEC constraints (5) for $|S| = 2$ as can be seen from the following. Let $S = \{i, j\}$, $e = \{i, j\}$, $k = j$ and $l \notin S$, then the following GSEC is obtained:

$$x_e \leq y_i - y_l + 1 \tag{14}$$

This constraint is valid whenever constraint (13) is valid since $y_l \leq 1$ and thus $x_e \leq y_i \leq y_i - y_l + 1$. Also values exist for which constraint (14) is valid but constraint (13) is not (e.g. $x_e = 1$, $y_i = 0$ and $y_l = 0$). Thus constraint (13) dominate constraint (14). Computational experiments indicate that these improve the performance only marginally.

The constraints mentioned below are valid only when the number of nodes in the ring is limited i.e. constraint (11) is used.

$$\sum_{j:e=\{i,j\}\in E} z_e \leq (b_y - 1)y_j \text{ for } i \in V \tag{15}$$

The constraints are used in e.g. [4] and [5] for the QKP. They improve the LP-relaxation considerably. They can be seen to be valid by multiplying (11) by y_j and replacing $y_i y_j$ with z_e for $e = \{i, j\}$ and $y_j y_j$ by y_j.

3 Branch-and-Cut Algorithm

We solve QSTSP using branch-and-cut. The root problem consists of (1), (2), bounds on variables and one of the budget constraints (10) or (11). If constraint (11) is used, we also include (15). We generate (3), (13) and the GSECs (5).

We have implemented the branch-and-cut algorithm using the Branch-Cut-and-Price framework (BCP) which is part of the COmputational INfrastructure for Operations Research (COIN-OR or just COIN) [1]. COIN contains open-source software useful in the field of operations research. BCP is, as its name indicates, a framework for developing Branch-Cut-and-Price algorithms. The choice of using BCP instead of implementing from scratch was made to speed up development including additional benefits of e.g. having an advanced cut-pool without having to worry explicitly about it. We have used BCP standard setups and classes whenever possible.

The branch-and-cut algorithm is shown in figure 1. The algorithm maintains a set

```
INCUMBENT = A feasible solution found heuristically (may be the empty ring).
SET_OF_SUBPROBLEMS = {Root problem}
while SET_OF_SUBPROBLEMS ≠ ∅
  do
    Select subproblem
    Solve LP relaxation of subproblem (including generated cuts)
    Let OBJ_VAL = objective value of LP relaxation
    Generate and add violated cuts
  while New cuts added
  if LP solution is feasible (hence integer) and OBJ_VAL > INCUMBENT:
      Update incumbent: INCUMBENT = OBJ-VAL
      Fathom subproblem
  else if OBJ-VAL ≤ INCUMBENT: Fathom subproblem
  else Branch: Add two subproblems to SET_OF_SUBPROBLEMS
end while
```

Figure 1. *The branch-and-cut algorithm*

of subproblems, initially containing the root node only. An initial feasible solution is generated by selecting the best solution from the two heuristics described in section 4.

Subproblems are considered depth first. This allows for reuse of the obtained LP-tableau in the following iteration. Branching is done on a y if any non-integer exists. Secondarily we branch on a x. Selecting whether a node is in the solution or not (i.e. branch on a y) seems to affect the solution more than selecting whether an edge is in the solution (i.e. branch on a x). Thus swapping the order of branching (i.e. branch on x's before y's) is not a good idea. This is supported by computational experiments. When choosing among different y's (or x's) we choose the variable closest to $1/2$. In one subproblem we set the branch variable equal to one and in the other we set the branch variable equal to zero. We consider the subproblem with the branch variable equal to one first.

For the selected subproblem we generate violated cuts, add them and resolve. We generate as many cuts as possible before we branch. Ineffective cuts are taken out of the formulation and ineffective GSECs are kept in a cut pool (handled by BCP). The cut pool is purged when switching subproblem.

Generation of (3) and (13) is done by checking whether any is violated. Since there are $(n^2 - n)/2$ of each of those, this takes $O(n^2)$. Generation of GSECs is described in the following section.

3.1 Separation of GSECs

Separation of GSECs is due to [7] which presents an optimal and a heuristic separation routine. Optimal separation can be achieved in $O(n^4)$ but proves to be ineffective for STSP. For completeness we will however describe how optimal separation can be achieved.

Assume x_e^*, y_i^* and z_e^* is the value of an optimal solution to the relaxed problem. Set up an undirected graph with n nodes and capacity of edges equal to x_e^*.

Optimal separation of GSECs is best described based on constraint (9). Recall the interpretation of the constraint: If a node is selected on both sides of a cut, then at least two edges crossing the cut must be selected. Thus separation can be done by for all pairs of nodes k, l finding a min-cut separating k and l or equivalently finding a max-flow between k and l. Given the min-cut, evaluate (9) to check whether it is violated. This can be done in $O(n^5)$ since finding max-flow can be done in $O(n^3)$ and $O(n^2)$ pairs of nodes exists.

However we can do better using the following observation. For a given $S' \subset V$, the most violated cut is obtained for $k' \in S'$ and $l' \notin S'$ such that $k' = \arg\max_{i \in S'}\{y_i^*\}$ and $l' = \arg\max_{i \notin S'}\{y_i^*\}$. Assume without loss of generality that $y_{k'}^* \geq y_{l'}^*$. For any GSEC defined by S, k and l it holds, that $k' \in S$ or $k' \notin S$. But In that case (one of) the most violated GSECs will have $k = k'$ if $k' \in S$ or $l = k'$ if $k' \notin S$. Thus it suffice to pick any such k' and consider pairs for this k' and all other nodes l. This is only $O(n)$ pairs and thus in total the complexity is $O(n^4)$.

So the separation algorithm is as follows. Pick a node k such that $k = \arg\max_{i \in V}\{y_i^*\}$. For all nodes $l \neq k$ find a max-flow from k to l. Evaluate (9) to check whether the cut is violated and if so add it (in the form of (5)). Thus up to $n - 1$ cuts may be added.

Computational experiments shows that substantial time is spent on optimal separation, however the most important problem is that the generated cuts do not span the graph [7]. Suppose that three disjoint sets $S_a \subset V, a = 1, 2, 3$ exists for which $\delta(S_a) = 0$. Assume the k chosen during optimal separation is in S_1 fixed. Then (at least) two GSECs will be generated with $S = S_1$ and $l \in S_2$ or $l \in S_3$. The cut with $S = S_2$, $k \in S_2$ and $l \in S_3$ will not be generated, and this may be important.

This can be ensured by using the $O(n^5)$ separation algorithm, but in that case the cut generation is even higher. An alternative is to use heuristic separation as described in [7] and described in the following.

Again set up an undirected graph with n nodes and capacity of edges equal to x_e^*. The idea is then to generate a minimum spanning forest using Kruskal's algorithm and each time two components are merged, a cut is identified and added if it is violated. Initially all nodes constitutes a component. Consider all edges $e = \{i, j\}$, $x_e^* > 0$ in non-increasing order. Merge components if i and j are in different components. When two components merged, let the resulting component be S. Also let $k = \arg\max_{i \in S}\{y_i^*\}$ and $l = \arg\max_{i \notin S}\{y_i^*\}$. Add the corresponding GSEC if it is violated.

Checking whether a GSEC is violated can be done in $O(n^2)$ and this may be done at most $n-1$ times (once for each merge) i.e. $O(n^3)$. However, this can be reduced to $O(n^2)$ in total by doing the following. Maintain for each component the weight internally in the component and the weight to all other components. For each merge of two components, update the weights, which can be accomplished in linear time in n. Since at most n merges are carried out the total complexity is $O(n^2)$. The complexity of Kruskal's algorithm is $O(m \log m)$ where m is the number of edges. Since m may be up to $O(n^2)$ the complexity expressed in terms of n is $O(n^2 \log n)$. Thus the complexity of the heuristic separation is $O(n^2 \log n)$.

As mentioned, the primary advantage of the heuristic separation routine is that it generates GSECs that span the graph. This is confirmed by computational experiments. The heuristic separation routine is used in all computational experiments presented in section 5.

4 Heuristics

We have developed two types of greedy heuristics. One builds up a ring from scratch (similar to the heuristics used in [8]) and one removes nodes from a full ring initially including all nodes. To distinguish the two heuristics, we will denote them the construction heuristic and the destruction heuristic, respectively.

The heuristics are used for generating an initial solution (this does not give a significant speed up, though) and for comparison. Comparing the solutions found by heuristics with the optimal solutions justify the need for better methods. In [8] better heuristics are presented (along with the construction heuristic presented here). The better heuristics do however rely on more advanced codes unavailable to us. Also the test-instances used in [8] are not available, hence we cannot compare directly. The comparison in [8] shows that for instances with 40, 80 and 120 nodes, the results for the best heuristic (average over 10 instances) are 15%, 43% and 55%, respectively, better than the construction algorithm.

The construction heuristic is shown in figure 2. For notational convenience we use the following notation $r_{ij} = r_e$ for $\{i,j\} = e$ and $c_{ij} = c_e$ for $\{i,j\} = e$. Let Δ_{bud} be the change in budget, equal to 1 when the number of nodes is limited and equal to $c_{ij} - c_{ik} - c_{jk}$ when the length of the ring is limited. The construction heuristic

Let $\{i,j\} = \arg\max\{r_{ij} + r_i + r_j - c_{ij} | \text{budget holds}\} \in E$
Let $k = \arg\max\{r_{ik} + r_{jk} + r_k - c_{ik} - c_{jk} | \text{budget holds}\} \in (V \backslash \{i,j\})$
Let $R = \{i,j,k\} \subseteq V$
Let $F = \{\{i,j\},\{j,k\},\{k,i\}\} \subseteq E$
while a node exists which can be added without violating the budget
   Find $k \in V \backslash R$ and $ij \in F$:
   $k,i,j = \arg\max\{(\sum_{i \in R} r_{ik} + r_k + c_{ij} - c_{ik} - c_{jk})/\Delta_{bud} | \text{budget holds}\}$
   $R = R \cup \{k\}$
   $F = F \backslash \{\{i,j\}\} \cup \{\{i,k\},\{j,k\}\}$
end while
Run 2-opt on the final set of nodes R.

Figure 2. *The construction heuristic*

builds the ring by first (greedily) identifying a feasible ring with three nodes (first four lines). The ring is then extended until the budget is entirely used up. This is done by picking during each iteration the node which gives the most revenue minus additional cost relative to the increase in budget. Finally the tour is improved by running 2-opt on the selected set of nodes. Running 2-opt is defined as follows: For all non-adjacent edges ij and kl in the tour take out ij and kl and obtain two paths. Assume i and k are on the same path. Connect the two paths by inserting edge il and jk thus obtaining a new ring. If the new ring is shorter than the old ring, accept the new ring, otherwise keep the old ring.

The construction algorithm is aborted if it is not possible to find a ring with three nodes. In that case the solution consisting of no ring and with a value of 0 is reported.

The destruction heuristic is shown in figure 3. The destruction algorithm initially builds a tour containing all nodes. This is done using farthest insertion followed by running 2-opt. Farthest insertion constructs a tour as follows: Pick three nodes at random obtaining an initial ring. Add remaining nodes one by one by doing the following. For all nodes not in the ring, find the shortest distance to a node in the ring and denote this value $short_i$ where i is a node not in the ring. For all nodes not in the ring, select the node which has the highest $short_i$ value and insert this node between two nodes in the ring such that the length-increase is as small as possible.

Now the tour is shortened by removing one node during each iteration. The node removed is picked greedily as the node which decrease the objective the least relative to the decrease in budget. Finally 2-opt is run on the remaining set of nodes.

The idea of selecting all nodes and then removing one at a time comes from Quadratic

Find a tour containing all nodes using farthest insertion and 2-opt.
Let $R = V$
Let $F =$ Set of edges in the tour found
while Budget does not hold
  Find $k \in R$ and let i and j be neighbours, i.e. $\{i,k\}$, $\{j,k\} \in F$:
  $k = \arg\min\{(\sum_{i \in R \setminus \{k\}} r_{ik} + r_k + c_{ij} - c_{ik} - c_{jk})/\Delta_{bud}\}$
  $R = R \setminus \{k\}$
  $F = F \setminus \{\{i,k\},\{j,k\}\} \cup \{\{i,j\}\}$
end while
Run 2-opt on the final set of nodes R.

Figure 3. *The destruction heuristic*

Knapsack algorithms which do the same thing [4, 5].

In some cases the algorithms return solutions of negative value. These are rejected and the solution consisting of the empty ring with a value of 0 (which is better) is reported.

5 Computational Results

In this section computational results are presented. It is investigated how the branch-and-cut algorithm performs and how the heuristics perform. We conjecture that the budget type and tightness of the budget have major influence on the problem difficulty. In particular the problem is expected to be easier if the budget is either very loose or very tight. If the budget is very tight, the set of feasible tours is smaller than average budget tightness. Hence selecting one node will have more affect on the bound value and hence branching may finish faster.

When the budget is loose, assume the revenue is considerable in comparison with the distance cost. Thus almost all nodes can and will be selected and thus the optimal revenue is close to the total possible revenue ($= \sum_{i \in V} r_i + \sum_{e \in E} r_e$). Such problems reduce to TSP problems plus a constant corresponding to the revenue. The branch-and-cut algorithm in essence solves these problems as one would solve TSP problems, by generating subtour elimination constraints. Thus the bound on the optimal distance cost corresponds to the value of the LP relaxation of the TSP problem which is good. Thus in total one obtains a rather good bound on the optimal solution value and thus problems with loose budgets are easy. The most difficult problems are the problems with an optimal solution which has approximately $n/2$ nodes.

Thus in the tests, the number of nodes, and the budget tightness for both type of budgets is varied. First it is described how the test instances used for testing are generated.

5.1 Generation of Test Instances

The test instances are generated the same way as suggested in [8] except that node-values which do not exist in [8] are generated. n points are generated in the plane with coordinates uniformly distributed between 0 and 100. Coefficient c_e is the Euclidean distance (rounded to integer) between points i and j, $e = \{i, j\}$.

When limiting the length of the ring, set $b_x = 0.75\sqrt{n/2}$ which will have the effect that somewhat more than $n/2$ nodes will be in the optimal ring [8]. Since c_e is the coefficients in both the objective and in the budget constraint limiting the ring length, the distance-cost incurred is at most b_x. Given this an expected total revenue of similar size is constructed. To make sure that no instances are generated where it does not pay off to have any nodes, the revenues are constructed such that the total expected revenue R is $1.15b_x$. The aim is to distribute the total revenue equally over node-revenues and demand-revenues. Thus the test instances have the property that the expected node-revenue and demand-revenue are both $R/2$.

Given the total node- and demand-revenue and the expected number of nodes in the ring ($= n/2$), the average node- and demand-revenue can be determined. The average node-revenue $\overline{r}_n$ is:

$$\frac{n}{2}\overline{r}_n = \frac{R}{2} = \frac{1.15b_x}{2} \Leftrightarrow \overline{r}_n = \frac{1.15b_x}{n} \tag{16}$$

The number of demands for l nodes is $(l^2 - l)/2$, thus the total demand revenue is:

$$R/2 = \overline{r}_d((n/2)^2 - n/2)/2 \approx \overline{r}_d((n/2)^2)/2 \tag{17}$$

Isolating $\overline{r}_d$, an expression for the average demand-revenue is obtained:

$$\overline{r}_d \approx \frac{R}{(n/2)^2} = \frac{4 \cdot 1.15 \cdot b_x}{n^2} = \frac{4.6b_x}{n^2} \tag{18}$$

The revenue of a node is now uniformly distributed between 0 and $2\overline{r}_n$ and the revenue of an edge is uniformly distributed between 0 and $2\overline{r}_d$. The revenue is turns out to be much higher than expected, when using limiting the length of the ring, since the number of nodes in optimal solutions is on average higher than $n/2$. 10 instances of networks with 10, 20, 30, 40 and 50 nodes have been generated.

5.2 Results

In this section computational results for the branch-and-cut algorithm and for the two heuristics are presented. As mentioned BCP [1] is used for the branch-and-cut algorithm and Cplex 7.5 is used for solving LP problems. The tests are run on a pc with a 1.6 Ghz Intel Xeon processor running Linux.

It is investigated how the tightness and the type of the budgets affect running time. This is done by varying the budget tightness. When the length of the ring is limited, tests are carried out for a budget of value equal to 0.2, 0.6 1.0, 1.4 and 1.8 times the b_x described above. When limiting the number of nodes in the ring, b_y is set equal to 5, 10, 20, 30, 40 and 50. Results for instances where $b_y > n$ is not reported, since the results are the same as for $b_y = n$. For values of Results given are averages over 10 instances. All tests completed within one hour. The results when limiting the length of the ring are given in table 1.

The columns of table 1 show the number of nodes in the test instance and the budget relative to the normal budget which identifies the test instance. The following column contains the average number of nodes in solutions, and the following column contains the average optimal solution value. The following three columns show the components of the objective value corresponding to x, y and z (see (1)). The following four columns contain the LP relaxation value in the root (after adding cuts), the gap between the optimal solution value and the Root LP-value relative to the optimal solution, the number of subproblems and the maximum depth of the branch-tree. Finally the running times are shown. The total time and the two most time consuming operations are reported; the time spent on generating cuts and the time spent on solving LPs.

The results for limit on the number of nodes in the ring are given in table 2. The table contains the same columns as table 1, except the budget column which gives the maximum number of nodes in the ring (b_y).

Note that distance-costs and the node- and demand-revenues all contribute significantly to the objective value. This corresponds to how they were generated the test instances.

It can be seen from the results that the budget has considerable impact on the running times. As conjectured in the beginning of section 5, the running time is highest when approximately $n/2$ nodes are in the optimal solution but also it can be seen that instances with limit on the length of the ring are solved faster than instances with a limit on the number of nodes.

Note that the main part of the running time is spent on solving LPs. Also note that the value of the LP relaxation and generated GSECs in the root is good for high budget values and bad for low budget values as conjectured.

The two different heuristics are tested for the two different budget types with varying budget tightness. The same values of b_x and b_y are used as described above. Running times of the heuristics are insignificant - less than 1 second for the instance sizes considered. The results are given in table 3 and table 4. The tables repeat the optimal solution value found by the branch-and-cut algorithm and gives result for

n	Budget	Nodes in Solution	Optimal value	Distance cost	Node revenue	Demand revenue	Root LP-value	Gap (%)	Number of subproblems	Branch-tree depth	Time (seconds) Total	Cut	LP
10	0.2	0.6	7.5	5.9	8.4	5.0	43.8	482.8	9.2	4.1	0.0	0.0	0.0
10	0.6	3.9	54.6	75.7	82.3	48.0	117.3	115.0	21.6	5.0	0.1	0.0	0.0
10	1.0	6.2	113.8	144.6	128.2	130.1	172.4	51.5	20.2	5.2	0.1	0.0	0.0
10	1.4	8.3	182.2	219.4	163.2	238.4	220.3	20.9	12.4	4.9	0.0	0.0	0.0
10	1.8	9.5	227.9	259.4	181.3	305.9	239.4	5.1	5.0	1.8	0.0	0.0	0.0
20	0.2	3.5	32.7	41.7	61.1	13.2	75.5	131.2	24.6	6.5	0.4	0.0	0.3
20	0.6	8.7	104.5	127.3	137.3	94.6	207.1	98.1	66.8	8.9	0.9	0.0	0.7
20	1.0	13.6	201.0	231.8	197.4	235.3	308.9	53.7	133.0	10.9	1.4	0.1	1.1
20	1.4	17.9	361.7	316.9	260.6	418.0	393.3	8.7	49.0	8.8	0.6	0.0	0.4
20	1.8	20.0	418.1	377.0	279.9	515.2	419.1	0.2	2.8	0.4	0.0	0.0	0.0
30	0.2	4.7	36.4	46.1	67.8	14.7	95.5	162.2	82.8	12.6	3.1	0.1	2.5
30	0.6	13.4	120.2	167.4	162.3	125.3	264.2	119.7	326.8	16.2	11.1	0.3	9.2
30	1.0	22.0	316.8	286.2	259.5	343.5	396.7	25.2	233.2	16.9	7.7	0.3	6.3
30	1.4	28.3	498.3	396.8	325.0	570.0	510.6	2.5	36.2	10.5	1.5	0.1	1.1
30	1.8	30.0	530.2	454.6	341.3	643.5	531.7	0.3	8.6	1.6	0.2	0.0	0.1
40	0.2	6.5	38.1	56.3	75.8	18.6	111.3	192.0	304.4	20.1	23.2	0.5	19.6
40	0.6	18.9	166.6	197.7	196.9	167.4	303.5	82.1	475.2	16.2	44.7	1.2	38.6
40	1.0	30.1	387.4	331.0	293.3	425.1	453.8	17.1	303.4	22.3	26.2	0.9	22.5
40	1.4	37.8	568.9	462.5	358.4	672.9	581.6	2.2	285.8	21.8	20.3	1.2	15.1
40	1.8	40.0	608.5	519.8	377.6	750.7	610.1	0.3	11.4	2.6	0.3	0.0	0.2
50	0.2	9.0	50.0	69.2	92.3	26.9	136.8	173.5	566.4	27.0	86.3	1.9	74.7
50	0.6	24.4	215.1	221.5	234.9	201.8	368.6	71.3	1459.0	23.2	283.4	7.0	249.3
50	1.0	38.7	490.3	372.3	354.4	508.2	556.3	13.5	438.6	28.1	83.6	2.5	73.6
50	1.4	48.0	694.2	511.5	429.9	775.8	707.5	1.9	229.2	28.3	35.2	1.8	28.7
50	1.8	50.0	725.5	559.4	443.8	841.2	727.6	0.3	25.6	3.2	1.1	0.1	0.8

Table 1. *Results for the branch-and-cut algorithm, limit on the length of the ring.*

n	Budget	Nodes in Solution	Optimal value	Distance cost	Node revenue	Demand revenue	Root LP-value	Gap (%)	Number of subproblems	Branch-tree depth	Time (seconds)		
											Total	Cut	LP
10	5	4.9	70.0	122.2	110.6	81.6	91.2	30.3	12.6	3.6	0.0	0.0	0.0
10	10	10.0	242.4	285.2	188.1	339.5	242.4	0.0	1.0	0.0	0.0	0.0	0.0
20	5	4.8	44.4	70.5	87.0	27.8	59.9	35.0	18.6	4.4	0.3	0.0	0.2
20	10	9.9	122.5	165.7	163.7	124.5	169.2	38.1	65.6	7.9	1.2	0.0	1.1
20	20	20.0	418.1	377.0	279.9	515.2	419.1	0.2	2.6	0.5	0.1	0.0	0.0
30	5	4.5	36.8	47.5	71.0	13.4	47.4	28.6	17.0	4.2	0.6	0.0	0.5
30	10	9.9	75.0	131.8	139.4	67.4	116.9	55.9	129.4	10.4	5.9	0.1	5.3
30	20	20.0	271.4	270.4	252.2	289.5	328.5	21.1	352.0	14.1	17.4	0.4	15.8
30	30	30.0	530.2	454.6	341.3	643.5	532.1	0.4	5.8	1.4	0.2	0.0	0.1
40	5	4.7	28.9	40.3	61.1	8.1	38.8	34.1	19.4	5.9	1.2	0.0	1.0
40	10	10.0	56.6	106.6	117.5	45.7	92.4	63.4	159.0	10.1	16.0	0.3	14.6
40	20	20.0	189.5	229.6	231.0	188.1	250.4	32.2	1123.8	19.0	139.9	2.0	129.1
40	30	30.0	393.8	349.3	314.8	428.4	441.0	12.0	693.6	23.2	86.1	1.1	80.7
40	40	40.0	608.5	519.8	377.6	750.7	610.1	0.3	9.8	2.8	0.5	0.0	0.4
50	5	4.4	28.7	30.7	53.5	6.0	38.1	32.9	22.2	6.0	1.9	0.0	1.6
50	10	9.8	54.2	84.9	107.1	32.0	84.7	56.3	134.8	10.7	24.4	0.4	22.6
50	20	20.0	157.3	197.6	218.3	136.6	216.3	37.5	947.6	18.9	273.6	3.3	259.2
50	30	30.0	317.8	300.4	312.6	305.6	387.5	21.9	5320.8	26.0	1525.9	16.1	1431.7
50	40	40.0	520.3	406.1	382.7	543.7	571.4	9.8	2257.0	32.6	541.5	4.4	516.7
50	50	50.0	725.5	559.4	443.8	841.2	727.6	0.3	30.2	3.3	1.9	0.0	1.6

Table 2. *Results for the branch-and-cut algorithm, limit on the number of nodes.*

n	Budget	Optimal solution value	Construct					Destruct				
			Solution value	Gap (%)	Distance cost	Node value	Demand value	Solution value	Gap (%)	Distance cost	Node value	Demand value
10	0.2	7.5	3.7	50.2	3.2	4.2	2.7	0.0	100.0	0.0	0.0	0.0
10	0.6	54.6	40.0	26.6	72.5	73.3	39.3	27.0	50.4	52.3	50.7	28.7
10	1.0	113.8	89.5	21.4	130.3	110.1	109.6	85.7	24.6	133.5	107.8	111.4
10	1.4	182.2	173.3	4.9	213.7	155.0	232.0	162.3	10.9	210.4	151.5	221.2
10	1.8	227.9	193.1	15.2	245.4	168.3	270.2	221.7	2.7	262.3	180.0	304.0
20	0.2	32.7	23.4	28.3	34.0	49.0	8.4	12.4	62.2	17.3	24.7	5.0
20	0.6	104.5	69.9	33.1	117.2	112.7	74.4	39.6	62.1	74.0	74.7	38.9
20	1.0	201.0	135.4	32.6	219.0	172.5	181.9	116.2	42.2	205.8	160.5	161.5
20	1.4	361.7	260.3	28.0	320.0	230.8	349.5	339.8	6.1	314.2	256.0	398.0
20	1.8	418.1	363.8	13.0	371.1	267.3	467.6	413.7	1.1	381.4	279.9	515.2
30	0.2	36.4	27.3	25.0	42.5	57.6	12.2	24.4	33.0	31.5	47.9	8.0
30	0.6	120.2	84.1	30.0	148.6	138.0	94.7	22.5	81.3	116.3	87.6	51.2
30	1.0	316.8	188.0	40.6	283.9	220.4	251.5	213.5	32.6	284.2	225.4	272.3
30	1.4	498.3	332.2	33.3	394.3	283.9	442.7	466.4	6.4	390.7	316.0	541.1
30	1.8	530.2	452.6	14.6	474.6	329.7	597.5	523.8	1.2	461.0	341.3	643.5
40	0.2	38.1	23.1	39.3	55.9	64.0	15.0	2.0	94.7	10.5	9.9	2.6
40	0.6	166.6	95.7	42.6	193.4	167.8	121.3	40.3	75.8	130.6	99.3	71.7
40	1.0	387.4	241.6	37.6	323.5	255.2	309.9	272.4	29.7	327.5	260.9	338.9
40	1.4	568.9	392.8	31.0	450.5	316.6	526.6	526.2	7.5	458.7	348.2	636.7
40	1.8	608.5	530.1	12.9	546.8	370.1	706.8	596.9	1.9	531.4	377.6	750.7
50	0.2	50.0	22.6	54.9	63.5	68.8	17.2	14.7	70.5	38.1	44.5	8.3
50	0.6	215.1	119.3	44.6	218.7	196.5	141.4	68.4	68.2	172.6	146.0	95.0
50	1.0	490.3	309.4	36.9	368.3	308.3	369.4	362.0	26.2	366.8	322.0	406.8
50	1.4	694.2	532.7	23.3	516.4	394.5	654.6	647.7	6.7	513.9	420.4	741.2
50	1.8	725.5	624.4	13.9	619.2	435.8	807.8	703.8	3.0	581.1	443.8	841.2

Table 3. *Results for heuristics, limit on the length of the ring.*

n	Budget	Optimal solution value	Construct					Destruct				
			Solution value	Gap (%)	Distance cost	Node value	Demand value	Solution value	Gap (%)	Distance cost	Node value	Demand value
10	5	70.0	61.6	12.0	115.2	100.0	76.8	53.8	23.1	126.8	104.1	76.5
10	10	242.4	192.6	20.5	260.7	172.2	281.0	240.1	0.9	287.5	188.1	339.5
20	5	44.4	35.2	20.7	71.9	80.2	26.9	20.0	55.0	43.4	49.4	14.0
20	10	122.5	93.1	24.0	193.1	157.9	128.3	87.1	28.9	230.4	180.8	136.7
20	20	418.1	357.6	14.5	376.4	267.5	466.9	413.7	1.1	381.4	279.9	515.2
30	5	36.8	26.0	29.4	54.3	65.8	14.5	1.6	95.6	19.9	18.4	3.1
30	10	75.0	52.2	30.3	126.5	116.5	62.2	34.4	54.1	168.8	140.8	62.4
30	20	271.4	216.9	20.1	305.7	232.6	289.8	246.9	9.0	311.7	267.3	291.2
30	30	530.2	458.7	13.5	468.4	328.6	600.1	523.8	1.2	461.0	341.3	643.5
40	5	28.9	22.0	24.0	42.0	54.8	9.1	8.7	70.0	13.3	19.1	2.8
40	10	56.6	37.6	33.6	99.3	96.4	39.6	13.3	76.4	96.5	82.4	27.4
40	20	189.5	125.9	33.6	236.8	191.8	166.7	144.9	23.5	297.4	249.1	193.2
40	30	393.8	314.1	20.2	407.9	297.5	417.8	362.0	8.1	385.1	318.3	428.8
40	40	608.5	522.9	14.1	566.3	370.6	714.5	596.9	1.9	531.4	377.6	750.7
50	5	28.7	19.7	31.4	50.1	62.5	7.3	2.6	91.0	19.5	19.6	2.5
50	10	54.2	35.4	34.6	86.9	93.5	28.9	5.6	89.7	75.2	65.1	15.7
50	20	157.3	110.4	29.8	220.9	199.7	131.5	93.5	40.6	281.3	238.0	136.8
50	30	317.8	240.1	24.5	339.1	279.0	300.2	270.7	14.8	363.7	325.5	308.9
50	40	520.3	427.2	17.9	479.0	365.4	540.8	489.7	5.9	446.4	390.0	546.1
50	50	725.5	632.6	12.8	612.7	436.5	809.4	703.8	3.0	581.1	443.8	841.2

Table 4. *Results for heuristics, limit on the number of nodes.*

the two heuristics. For each heuristic, the solution value, the gap relative to the optimal solution and the components of the objective value are reported.

The construction algorithm perform the best if the budget is tight whereas the destruction algorithm performs the best if the budget is loose. Comparing the heuristic solutions found with the proven optimal solutions, there is a substantial gap.

5.3 Discussion of Results

As noted the main part of computational time is spent on solving LPs. This has three explanations. 1) The bound obtained is weak, so the number of processed subproblems is rather high. 2) For each subproblem the LP-solver is invoked several times in order to strengthen the bound from the newly added cuts (and to be able to generate more cuts). 3) The LPs are of considerable size.

Improving on the first point can be done by identifying new cuts which improve the bound. Constraint (15) is an example of this and it is important for improving the performance in case of limiting the number of nodes in the ring. Improving on the second point can be done by generating more cuts and more importantly generate the "right" cuts [13]. For each subproblem, the LP is solved, additional cuts are generated and the LP is re-solved and so on until no more violated cuts exist. The right cuts are the cuts that are ineffective at the end of this process and if it were possible to identify them, only two calls to the LP solver would be necessary for each subproblem (one initial call and one re-solve). In general this is not possible to do without actually carrying out the iterations but one can try to develop heuristics which are better to do this than simply finding all violated cuts. In particular the heuristic separation routine used to generate GSECs is an example of a routine which generate better cuts than the ones generated using the optimal separation routine.

Finally improving on the third point at least two possibilities exists. One possibility is to evaluate the bound (or alternative bounds) combinatorially. This is done with success for the Quadratic Knapsack Problem [5]. The other possibility is to try to reduce the number of constraints in the formulation and the number of non-zero variables in the non-removable constraints. This is the reason for removing ineffective cuts and (re-)generating (3), (13) and GSECs and is important in order to achieve the reported performance. It is also possible to reduce the number of non-zero variables using the following observation. It may be that the generated cuts can be reduced as in [13] and still capture the essence of the cuts with fewer variables.

As an example of a simple reduction, let the current LP solution be x_e^*, y_i^* and z_e^*. Assume that a violated GSEC is given by $S_1 \subset V$, $|S_1| \leq |V|/2$, $k_1 \in S_1$, $l \notin S_1$ and $\sum_{e \in \delta(S_1)} x_e = 0$. Assume further that S contains two nonempty disjoint subsets $S_2 \cup S_3 = S_1$ such that $\sum_{e\in\delta(S_2)} x_e = \sum_{e\in\delta(S_3)} x_e = 0$. Instead of the GSEC corresponding to S_1, one can include two GSECs with S_2, $k_2 \in S_2$, $l \notin S$ and S_3, $k_3 \in S_3$, $l \notin S$. The nodes k_2 and k_3 are picked such that $y^*_{k_2} = \max_{i\in S_2}\{y^*_i\}$ and $y^*_{k_3} = \max_{i\in S_3}\{y^*_i\}$. The GSECs are used in inner form (constraint (5)) and thus the number of variables for a GSEC constraint is $(|S|^2 - |S|)/2 + |S|$. It can be checked that for $|S_1| \geq 3$ fewer non-zero variables are required to represent the two GSECs corresponding to S_2 and S_3 than the GSEC corresponding to S_1.

We believe that the most promising way to go is to look for new cuts related to the z_e variables which are the reason for the highly fractional solutions. Note that in order for better bounds to be really useful, better heuristics than the two construction heuristics presented were needed. In order to find the maximum speed-up which could be anticipated if good heuristics were used, the optimality verification process has been isolated, i.e. the following has been carried out. Obtain the optimal solution by branch-and-cut and measure the running time. Rerun the branch-and-cut algorithm but initialize the incumbent to the optimal solution. Doing this, the average reduction in computational time is approximately 40% for the instances which require more than 10 seconds. This is the absolutely highest computational improvement that can be realized due to improved heuristics. If the bounds are improved, this may change.

6 Conclusions

The Quadratic Selective Travelling Salesman Problem has been presented as an extension of the (Selective) TSP. An integer linear programming model for QSTSP has been presented. The problem is solved optimally using branch-and-cut and by two simple construction heuristics. Computational studies of the construction heuristics indicates that the problem is difficult - better heuristics and/or optimal algorithms do have a justification. Computational studies show that the branch-and-cut algorithm can find optimal solutions to all test problems with up to 50 nodes within one hour.

In order for the branch-and-cut algorithm to be really useful, it is necessary to decrease the running time. As noted, the main part of the computational time is spent on solving LPs. Hence, to get a substantial overall reduction in computation time, the time spent on solving LPs should be reduced. As discussed earlier, this requires finding better cuts.

An alternative to the branch-and-cut algorithm is to use a more advanced heuristic than the construction heuristics suggested here, e.g. a meta-heuristic.

Acknowledgments

The authors would like to thank Jens Clausen and Jesper Larsen for helpful discussions and comments.

References

[1] Coin-or: Computational infrastructure for operations research, `www.coin-or.org`.

[2] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–36, 1989.

[3] E. Balas. The prize collecting traveling salesman problem. ii. polyhedral results. *Networks*, 25(4):199–216, 1995.

[4] Alain Billionnet and Frederic Calmels. Linear programming for the 0-1 quadratic knapsack problem. *European Journal of Operational Research*, 92(2):310–325, 1996.

[5] A. Caprara, D. Pisinger, and P. Toth. Exact solution of the quadratic knapsack problem. *INFORMS Journal on Computing*, 11(2):125–37, 1999.

[6] M. Fischetti, J.J. Salazar Gonzalez, and P. Toth. The symmetric generalized traveling salesman polytope. *Networks*, 26(2):113–23, 1995.

[7] M. Fischetti, J.J Salazar Gonzalez, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–94, 1997.

[8] M. Gendreau, M. Labbe, and G. Laporte. Efficient heuristics for the design of ring networks. *Telecommunication Systems - Modeling, Analysis, Design and Management*, 4(3-4):177–88, 1995.

[9] M. Gendreau, G. Laporte, and F. Semet. A branch-and-cut algorithm for the undirected selective traveling salesman problem. *Networks*, 32(4):263–73, 1998.

[10] Luis Gouveia and Jose Manuel Pires. Models for a steiner ring network design problem with revenues. *European Journal of Operational Research*, 133(1):21–31, 2001.

[11] G. Laporte. Generalized subtour elimination constraints and connectivity constraints. *Journal of the Operational Research Society*, 37(5):509–14, 1986.

[12] G. Laporte and S. Martello. The selective travelling salesman problem. *Discrete Applied Mathematics*, 26(2-3):193–207, 1990.

[13] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991.

[14] A. Proestaki and M.C. Sinclair. Design and dimensioning of dual-homing hierarchical multi-ring networks. *IEE Proceedings-Communications*, 147(2):96–104, 2000.