Methods and Models for Combinatorial Optimization

Exact methods for the Traveling Salesman Problem

L. De Giovanni M. Di Summa

The Traveling Salesman Problem (TSP) is an optimization problem on a directed or undirected graph. In the case of a directed graph $D = (N, A)$, there is a cost c_{ij} for every arc $(i, j) \in A$. In the case of an undirected graph $G = (V, E)$, there is a cost c_e for every edge $e \in E$. The TSP consists in determining a minimum cost *Hamiltonian cycle* in the graph (if any exists). We recall that a Hamiltonian cycle is a cycle that visits all the nodes of the graph exactly once. Note that we can assume without loss of generality that the graph is complete, i.e., it contains an arc/edge for every ordered/unordered pair of nodes: if this is not the case, one can define the cost of the missing arcs/edges to be ∞.[1] The models and the techniques for the solution of the TSP can be specialized for the case of a directed graph (asymmetric TSP) or undirected graph (symmetric TSP). For this reason, we analyze the two cases separately.[2] However, the TSP is an $\mathcal{NP}$-hard problem and therefore its exact solution requires algorithms with exponential complexity (unless $\mathcal{P} = \mathcal{NP}$).

1 Asymmetric TSP

The Asymmetric TSP (ATSP) is defined on a directed graph $D = (N, A)$ with a cost c_{ij} for every arc $(i, j) \in A$. One of the possible models for this problem has a variable for every arc:

$$\forall\, (i,j) \in A,\ x_{ij} = \begin{cases} 1 & \text{if } (i,j) \text{ is in the cycle;} \\ 0 & \text{otherwise.} \end{cases}$$

[1] Alternatively, one can assign to the missing arcs/edges the cost of the shortest path between the two end-nodes.

[2] Actually, the methods that we will present can be adapted to both cases: in every section, we present the methods that, from the viewpoint of computational efficiency, are more successful for one case or for the other.

$$
\begin{array}{lrcll}
\min & \displaystyle\sum_{(i,j)\in A} c_{ij}x_{ij} & & & \\
s.t. & \displaystyle\sum_{i\in N} x_{iv} & = & 1 & \forall \quad v \in N \\
& \displaystyle\sum_{j\in N} x_{vj} & = & 1 & \forall \quad v \in N \\
& \displaystyle\sum_{i,j\in S} x_{ij} & \leq & |S|-1 & \forall \quad S \subset N : 2 \leq |S| \leq |N|-1 \\
& x_{ij} & \in & \{0,1\} & \forall \quad (i,j) \in A
\end{array}
$$

The objective function is the sum of the costs of the selected arcs. In every node there must be exactly one of the selected arcs entering the node (first constraint) and exactly one of the selected arcs leaving the node (second constraint). With these conditions we traverse every node exactly once, but solutions consisting of more than one subcycle (or *subtour*) would be feasible: a subtour is a cycle that do not visit all nodes. The third group of constraints eliminates all solutions containing subtours, as for every proper subset of nodes ($S : 2 \leq |S| \leq |N|-1$) it forbids the number of selected arcs *within* S to be equal to or larger than the number of nodes in S. These inequalities are called *subtour elimination constraints*.

An alternative is to eliminate subtours by replacing the third group of constraints with the following (more explicit but larger) family of inequalities:

$$\sum_{(i,j)\in C} x_{ij} \leq |C| - 1 \; \forall \; C : \; C \text{ is a subtour in } G$$

The number of constraints of the model is $2|N|$ plus the number of subsets (first option) or subtours (second options), which is *exponential*. For this reason, the model cannot be used directly, unless the number of nodes is small. Therefore, it is necessary to develop solution approaches that are different from those seen so far, as an exponential number of constraints must be handled. The basic idea is to add to the model only those subtour elimination constraints that are really needed. The constraints that should be added can be obtained dynamically by means of an iterative approach.

The two methods that we present next are based on the following observation. If the subtour elimination constraints are removed from the model (but integrality of the variables is kept), we obtain precisely the model of an assignment problem: we can imagine a bipartite graph with the nodes of N on the left and another copy of them on the right; the cost of assigning node i (on the left) to node j (on the right) is the cost c_{ij} of going from i to j; in the solution, $x_{ij} = 1$ means that after node i we visit node j. Therefore, if *all* subtour elimination constraints are removed, the problem can be solved efficiently (for instance with the simplex method), because of the total unimodulairty of the matrix of the assignment problem. However, the solution that we obtain this way may be infeasible, as it may contain subtours. (If it has no subtours, it is the optimal solution

of the TSP.) In any case, this solution provides a lower bound for the minimum cost of a Hamiltonian cycle, as we have removed some constraints. A similar argument holds if only some subtour elimination constraints are removed: the solution obtained this way provides a lower bound for the optimum of the TSP. Summarizing:

Observation 1 *If all or some of the subtour elimination constraints are removed from the model, then the optimal solution of the resulting model may:*

- *contain subtours: then the optimal value of this model is a lower bound for the TSP;*
- *contain no subtours: then this solution is feasible and optimal for the TSP.*

Suppose that the optimal solution of the assignment problem is not feasible for the TSP because it contains subtours. Let $\hat{C}$ be one of the subtours and let $\hat{S}$ the set of nodes in $\hat{C}$. It is reasonable to add to the model a constraint that eliminate all solutions containing $\hat{C}$ as a subtour. To do so, we can add the inequality:

$$\sum_{i,j \in \hat{S}} x_{ij} \leq |\hat{S}| - 1$$

or the inequality

$$\sum_{(i,j) \in \hat{C}} x_{ij} \leq |\hat{C}| - 1.$$

For instance, if the current solution contains the subtour $1 \to 2 \to 3 \to 4 \to 1$, we can add the constraint

$$x_{12} + x_{21} + x_{13} + x_{31} + x_{14} + x_{41} + x_{23} + x_{32} + x_{24} + x_{42} + x_{34} + x_{43} \leq 3$$

or the constraint

$$x_{12} + x_{23} + x_{34} + x_{41} \leq 3.$$

However, the addition of this constraint destroy (in general) the total unimodularity of the matrix and therefore the new problem cannot be solved efficiently.

1.1 Constraint generation for the ATSP

From the previous observations we obtain a solution method for the ATSP in which the constraints are generated dynamically:

1. `Remove all subtour elimination constraints and all integrality constraints, and solve the corresponding assignment problem.`
2. *`Reintroduce the integrality constraints.`*
3. `If the current solution does not contain subtours, then STOP: optimal solution.`

4. Find one (or more) subtours in the current solution.
5. Add to the model the subtour elimination constraint for at least one of the subtours.
6. Solve the current problem and go to step 3.

The convergence of the algorithm is guaranteed by the fact that, in the worst case, we will add all subtour elimination constraints, thus obtaining the complete model of the ATSP. The idea behind the algorithm is that in fact not all constraints are needed to eliminate subtours. Experiments show that a Hamiltonian cycle is usually obtained after adding a limited number of subtour elimination constraints. However, from the viewpoint of computational complexity, the number of iterations in the worst case is equal to the total number of subtour elimination constraints.
Apart from the number of iterations, there is another computational issue: at every iteration we have to solve an integer linear programming problem and therefore every single iteration has exponential complexity.
It is possible to show that, in order to obtain an integer solution at every iteration, it is sufficient to enforce only the integrality of the variables appearing in the subtour elimination constraints already added to the model. Then the constraint generation algorithm becomes the following:

Constraint generation for the ATSP

1. Remove all subtour elimination constraints and all integrality constraints, and solve the corresponding assignment problem.
2. If the current solution does not contain subtours, then STOP: optimal solution.
3. Find one (or more) subtours in the current solution.
4. Add to the model the subtour elimination constraint for at least one of the subtours and the integrality constraints for the variables appearing in the constraint.
5. Solve the current problem and go to step 2

Therefore at every iteration we can limit the number of integer variables. However, at every iteration we have a mixed integer linear programming problem, whose complexity is still exponential.

We also note that at every iteration, finding subtours in the current solution $x^* \in \{0,1\}^{|E|}$ is easy: it is sufficient to detect the connected components of the graph $D_{x^*} = (N, A_{x^*})$ containing only the arcs (i,j) such that $x^*_{ij} > 0$. D_{x^*} is called the support graph of x^*.

Example 1 *Solve the ATSP with the following distance matrix between pairs of nodes:*

	1	**2**	**3**	**4**	**5**	**6**
1	–	33.6	14.0	40.9	14.5	11.5
2	34.7	–	21.7	13.0	20.2	23.4
3	14.8	21.5	–	29.3	2.0	3.9
4	41.7	13.1	29.4	–	27.6	30.3
5	15.0	20.2	2.0	27.5	–	3.9
6	12.0	22.8	2.0	30.1	4.0	–

We solve the problem with the constraint generation algorithm (the reader can verify the results with AMPL or some other software).

Initialization
We remove all subtour elimination constraints and integrality constraints, and we solve the resulting assignment problem. We obtain the following solution:

$$x_{16} = x_{61} = x_{24} = x_{42} = x_{35} = x_{53} = 1; \text{ cost } 53.6$$

Iteration 1 There are the following subtours:

$$1 - 6 - 1 \qquad 2 - 4 - 2 \qquad 3 - 5 - 3$$

We add the subtour elimination constraints for the three subtours:

$$x_{16} + x_{61} \leq 1$$
$$x_{24} + x_{42} \leq 1$$
$$x_{35} + x_{53} \leq 1$$

and the integrality constraints

$$x_{16}, x_{61}, x_{24}, x_{42}, x_{35}, x_{53} \in \{0, 1\}.$$

We solve the new problem and obtain the solution

$$x_{16} = x_{63} = x_{31} = x_{24} = x_{45} = x_{52} = 1; \text{ cost } 89.1$$

Iteration 2
The new solution contains the subtours

$$1 - 6 - 3 - 1 \qquad 2 - 4 - 5 - 2$$

We add the subtour elimination constraints for the two subtours:

$$x_{16} + x_{63} + x_{31} \leq 2$$
$$x_{24} + x_{45} + x_{52} \leq 2$$

and the integrality constraints

$$x_{63}, x_{31}, x_{45}, x_{52} \in \{0, 1\}.$$

(Constraints $x_{16}, x_{24} \in \{0, 1\}$ have been already added at iteration 1.) We solve the new problem and obtain the solution

$$x_{16} = x_{63} = x_{35} = x_{52} = x_{24} = x_{41} = 1; \text{ cost } 90.4$$

Iteration 3
The current solution contains no subtours and therefore gives an optimal Hamiltonian cycle:

$$1 - 6 - 3 - 5 - 2 - 4 - 1$$

1.2 Branch-and-bound for the ATSP

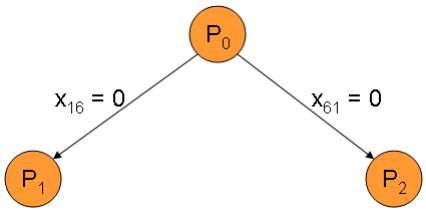
Based on Observation 1, we can use the lower-bound to implement a Branch-and-Bound approach. At the root node of the Branch-and-Bound tree we consider the problem without subtour elimination constraints. This can be solved as an assignment problem, and its solution gives a lower bound. If the current solution contains no subtours, we have the optimal solution for the ATSP, thus no further node has to be developed. Otherwise, we branch and generate some subproblems whose feasible solutions do not contain any of the subtours appearing in the current solution. This procedure can then be iterated until all nodes of the Branch-and-Bound tree are closed. However, also in this case the subtour elimination constraints added to the problem destroy the total unimodularity of the matrix and therefore an exponential time would be needed to solve every node. Furthermore, the number of nodes to explore is, in the worst case, exponential, thus this would result in an inefficient method.

The situation can be improved if we observe that:

- in order to achieve feasibility, we do not need to eliminate *all* the subtours in the current solution: it suffices to eliminate one subtour at every iteration;
- in order to be sure that no subtour containing a specified arc (i, j) appears again, it is sufficient to impose $x_{ij} = 0$ (if the arc cannot be used, no cycle containing that arc can appear).

The branching scheme can then be defined as follows: let $\hat{C}$ be *any* subtour in the solution of the assignment problem (at the root node). For every arc $(i, j) \in \hat{C}$, generate a subproblem by adding constraint $x_{ij} = 0$.

If we consider Example 1, after solving the assignment problem at the root node we obtain the subtours $1 - 6 - 1$, $2 - 4 - 2$ and $3 - 5 - 3$. Let us choose the subtour $1 - 6 - 1$ and branch according to the following scheme:



Note that with this branching scheme the following properties hold:

- all solutions containing the subtour $1-6-1$ are eliminated from both P_1 and P_2;
- no feasible solution is eliminated: the *Hamiltonian* cycles containing arc $(1,6)$, which are removed from P_1, are present in P_2; and the *Hamiltonian* cycles containing arc $(6,1)$, which are removed from P_2, are present in P_1.

This branching scheme is particularly convenient because of the simplicity of the constraints that we add to the problem. Indeed, there is no need to really add new constraints:

> *in order to enforce* $x_{ij} = 0$ *one can set* $c_{ij} = +\infty$.

By doing this, no constraint is added and we still have an assignment problem: we only changed the costs. Therefore at every node we *preserve the total unimodularity of the constraint matrix* and we can solve the corresponding problem efficiently (the simplex method will return an integer solution).

In the example, the lower bound for node P_1 is obtained by solving the assignment problem with the following costs:

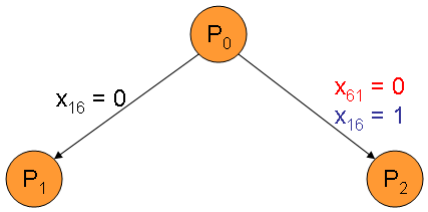
	1	**2**	**3**	**4**	**5**	**6**
1	–	33.6	14.0	40.9	14.5	$+\infty$
2	34.7	–	21.7	13.0	20.2	23.4
3	14.8	21.5	–	29.3	2.0	3.9
4	41.7	13.1	29.4	–	27.6	30.3
5	15.0	20.2	2.0	27.5	–	3.9
6	12.0	22.8	2.0	30.1	4.0	–

while for P_2 we have

	1	**2**	**3**	**4**	**5**	**6**
1	–	33.6	14.0	40.9	14.5	11.5
2	34.7	–	21.7	13.0	20.2	23.4
3	14.8	21.5	–	29.3	2.0	3.9
4	41.7	13.1	29.4	–	27.6	30.3
5	15.0	20.2	2.0	27.5	–	3.9
6	$+\infty$	22.8	2.0	30.1	4.0	–

Note that the subsets of feasible solutions in P_1 and P_2 *are not disjoint*. In the example, the Hamiltonian cycle $1-2-3-4-6-5-1$ belongs to both P_1 and P_2. This is not a problem, as in a Branch-and-Bound scheme we only have to make sure that no feasible solution is lost. Nonetheless, the computational efficiency of the method may be affected, as if the same solution appears in more than one node, the number of nodes to explore tends to be larger.

In order to limit the presence of the same solutions in different nodes, we can impose, for instance, $x_{16} = 1$ in node P_2 of the example, because the solutions with $x_{16} = 0$ are already included in P_1 and therefore do not need to be considered again in P_2. We then obtain:



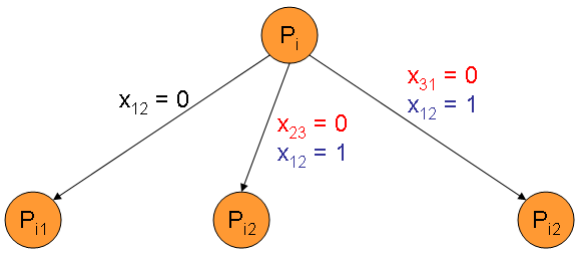
Also a condition of this type ($x_{ij} = 1$) can be enforced without modifying the constraint matrix, but just changing the costs:

in order to enforce $x_{ij} = 1$ *one can set* $c_{ij} = +\infty \ \forall\, j \neq i$.

In the example, the lower bound for node P_2 is obtained by solving the assignment problem with the following costs:

	1	**2**	**3**	**4**	**5**	**6**
1	–	$+\infty$	$+\infty$	$+\infty$	$+\infty$	11.5
2	34.7	–	21.7	13.0	20.2	23.4
3	14.8	21.5	–	29.3	2.0	3.9
4	41.7	13.1	29.4	–	27.6	30.3
5	15.0	20.2	2.0	27.5	–	3.9
6	$+\infty$	22.8	2.0	30.1	4.0	–

We remark that this new branching scheme *limits* the presence of the same solutions in different nodes, but does not exclude this possibility completely. For instance, if at some node P_i we obtain the subtour $1-2-3-1$, we have:



and the feasible solutions with $x_{23} = x_{31} = 0$ are present both in P_{i_2} and P_{i_3}. In order to avoid any overlapping, in this example we could generate nodes $P_{i_1}, P_{i_2}, P_{i_3}$ as follows: in P_{i_1} we impose $x_{12} = 0$; in P_{i_2} we impose $x_{12} = 1$ and $x_{23} = 0$; in P_{i_3} we impose $x_{12} = x_{23} = 1$ and $x_{31} = 0$. By doing this, the feasible regions of these three subproblems form a partition of the feasible region of the parent node. It is important to keep in mind that, although on the one hand avoiding overlapping between the nodes is an advantage, on the other hand this technique tends to develop an unbalanced tree (there are many more solutions in the first child node than in the last one), and this may create computational problems.

Example 2 *Use the above Branch-and-Bound method to solve the ATSP of Example 1.*

We refer to Figure 1:

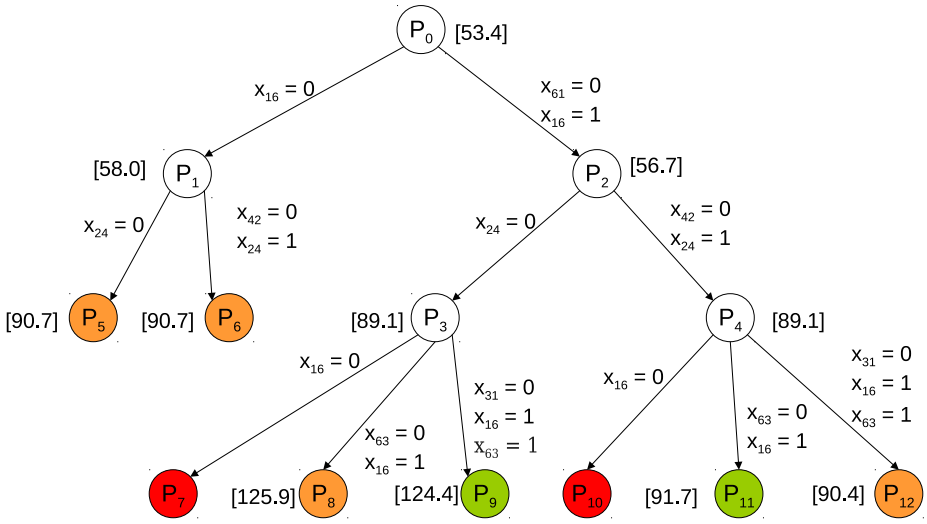


Figure 1: Branch-and-Bound tree for Example 1. Red nodes are pruned by infeasibility; green nodes are pruned by optimality; orange nodes are pruned by bound

$\underline{P_0}$: solution $1-6-1$, $2-4-2$, $3-5-3$, with cost 53.4.

We choose subtour $1-6-1$ for branching and generate:

$\underline{P_1}$: solution $1-3-5-6-1$, $2-4-2$, with cost 58.0;

$\underline{P_2}$: solution $1-6-3-5-1$, $2-4-2$, with cost 56.7.

We apply the *Best node* strategy and branch on P_2, by choosing subtour $2-4-2$ (we choose the shortest subtour to generate a small number of nodes), and obtain:

$\underline{P_3}$: solution $1-6-3-1$, $2-5-4-2$, with cost 89.1;

$\underline{P_4}$: solution $1-6-3-1$, $2-4-5-2$, with cost 89.1.

The best node is now P_1 and we choose subtour $2-4-2$, generating:

$\underline{P_5}$: solution $1-3-6-1$, $2-5-4-2$, with cost 90.7;

$\underline{P_6}$: solution $1-3-6-1$, $2-4-5-2$ with cost 90.7.

We develop P_3 by considering the subtour $1-6-3-1$:

$\underline{P_7}$: infeasible (the constraints $x_{16}=1$ and $x_{16}=0$ are inconsistent); P_7 is pruned by infeasibility;

$\underline{P_8}$: solution $1-6-2-3-1$, $4-5-4$, with cost 125.9;

$\underline{P_9}$: solution $1-6-3-2-5-4-1$, with cost 124.4. This solution is feasible: $1-6-3-2-5-4-1$ is the incumbent solution, P_9 is pruned by optimality, P_8 is pruned by bound.

Among the currently open nodes P_5, P_6 and P_4, the best one is P_4, with subtour $1-6-3-1$ and children:

$\underline{P_{10}}$: infeasible (because of constraints $x_{16}=1$ and $x_{16}=0$); pruned by infeasibility;

$\underline{P_{11}}$: solution $1-6-2-4-5-3-1$, with cost 91.7. This solution is feasible and better than the incumbent solution: $1-6-2-4-5-3-1$ becomes the new incumbent solution, P_{11} is pruned by optimality, but no other nodes can be pruned by bound.

$\underline{P_{12}}$: solution $1-6-3-5-2-4-1$, with cost 90.4. This solution is feasible and better than the incumbent solution: $1-6-3-5-2-4-1$ becomes the new incumbent solution, P_{12} is pruned by optimality, P_5 and P_6 are pruned by bound.

Since there are no open nodes, $1-6-3-5-2-4-1$ is an optimal solution, with cost 90.4.

2 Symmetric TSP

The symmetric TSP (STSP) is defined on an undirected graph $G=(V,E)$ with a cost c_e for every edge $e \in E$. The problem can be formulated and solved as an asymmetric TSP after transforming G into a directed graph with two arcs for every edge of G: for every edge $e \in E$ with ends $i \in V$ and $j \in V$ one can create two arcs (i,j) e (j,i) with $c_{ij}=c_{ji}=c_e$. However, the symmetry of the costs allows us to improve the model an the methods seen above. Indeed, an alternative formulation for the symmetric TSP uses a variable for every edge (and thus half the number of variables needed if the problem is transformed in an asymmetric TSP):

$$\forall\, e \in E,\ x_e = \begin{cases} 1, & \text{if } e \text{ is in the cycle;} \\ 0, & \text{otherwise.} \end{cases}$$

$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e & & & & & \\
s.t. \quad & \sum_{e \in \delta(v)} x_e & = & 2 & \forall & v \in V & (STSP:1) \\
& \sum_{e \in E(S)} x_e & \leq & |S| - 1 & \forall & S \subset V : 3 \leq |S| \leq |V| - 1 & (STSP:2) \\
& x_e & \in & \{0,1\} & \forall & e \in E & (STSP:3)
\end{aligned}
$$

Note that for every node $v \in V$ it is sufficient to choose two edges in $\delta(v)$ (i.e., the set of edges having an end in v), without specifying the direction along which the edge is traversed. The second group of constraints is equivalent to subtour elimination: $E(S)$ is the set of edges with both end-nodes in S. Unlike the asymmetric case, there is no need to eliminate subtours of length 2, as these subtours are already excluded by the first group of constraints. Also in this case we have to deal with an exponential number of constraints.

2.1 Constraint generation for the STSP

Similar to the case of the ATSP, we can iteratively solve a relaxed problem with a limited number of subtour elimination constraints (keeping integrality). At every iteration, we select one or more subtours in the current solution and we add the corresponding subtour elimination constraints. This is repeated until the solution is a Hamiltonian cycle. The procedure is the same as that described in Section 1.1; we illustrate it here with a flowchart (Figure 2).

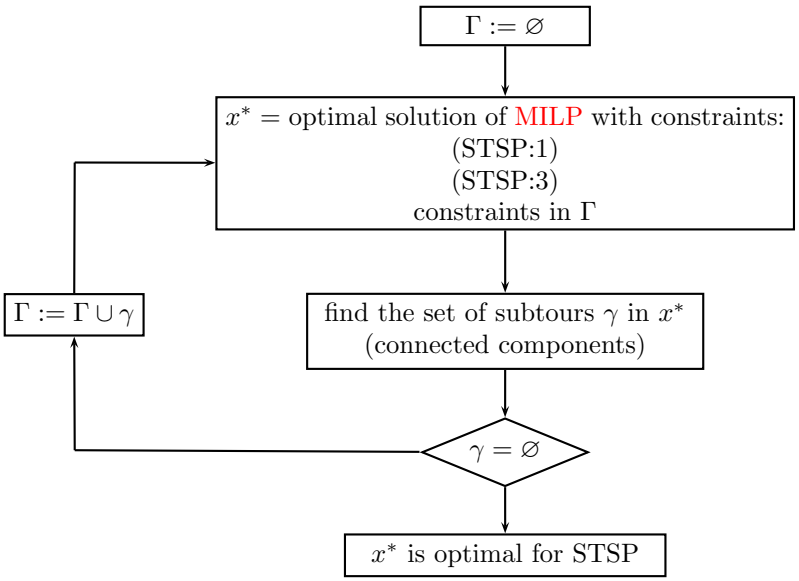


Figure 2: Constraint generation for the STSP.

As in the case of the ATSP, also for the STSP at every iteration an integer linear programming problem has to be solved, and also in this case it is enough to impose integrality

of the variables appearing in the subtour elimination constraints added to the problem, thus obtaining a mixed-integer linear programming (MILP) problem.

This procedure is not very efficient from the computational point of view: the number of iterations may be exponential, and at every iteration a mixed integer linear programming problem must be solved.

In the case of the STSP, instead of applying the Branch-and-Bound method described for the ATSP, it is more convenient to proceed in a different way by solving the continuous relaxation of the problem (i.e., with conditions $x_e \in \{0,1\}$ replaced by $0 \leq x_e \leq 1$). Note however that the linear relaxation has again an exponential number of constraints (the subtour elimination constraints), thus we need a method to generate them dynamically. In this situation we will destroy the total unimodularity of the matrix, but at every iteration we will be able to solve the problem as it will be a linear programming problem. This method is described below.

2.2 Separation of the subtour elimination constraints ad solution of the continuous relaxation

Suppose that we solve the *continuous relaxation* of a STSP with a limited number of subtour elimination constraints, and let $x_R^* \in [0,1]^{|E|}$ be an optimal solution. In this situation, the support graph $G_{x_R^*}$ (containing only the edges with $x_e^* > 0$) does not help us to detect subtours, as the entries of $\bar{x}_R^*$ are not 0/1. Then the problem is the following:

> *Given an optimal solution $x_R^* \in [0,1]^{|E|}$ of the continuous relaxation of a STSP with a limited number of subtour elimination constraints, find one or more subtour elimination constraints violated by $\bar{x}_R^*$.*

This problem is called *separation problem* because allows us to *separate* the current optimal solution x_R^* from the set of solutions that do not violate any subtour elimination constraint. In order to solve the separation problem, it is convenient to rewrite constraints $(STSP:2)$ in the following equivalent form:

$$\sum_{e \in \delta(S)} x_e \quad \geq \quad 2 \quad \forall \quad S \subset V : 3 \leq |S| \leq |V| - 1 \quad (STSP:2')$$

Here $\delta(S)$ is the set of edges with one end in S and the other end in $V \backslash S$, and the constraint imposes that at least two of these edges are part of the solution. These constraints eliminate all solutions containing subtours: indeed, if there is a subtour (see Figure 3), one can choose $\hat{S}$ as the set of nodes in the subtour, thus obtaining a violated constraint, as $\sum_{e \in \delta(\hat{S})} x_e = 0 < 2$. If, on the contrary, there are no subtours, then for every choice of $\hat{S}$ there are at least two edges to go from $\hat{S}$ to $V \setminus \hat{S}$ and vice versa (see Figure 4).

Consider now the fractional solution x_R^*. This solution satisfies the degree constraints $(STSP:1)$ at every node, but may violate some subtour elimination constraints. Consider

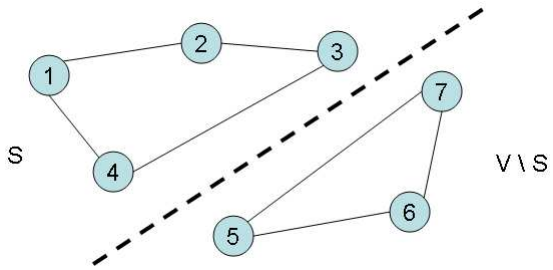


Figure 3: Presence of subtours.

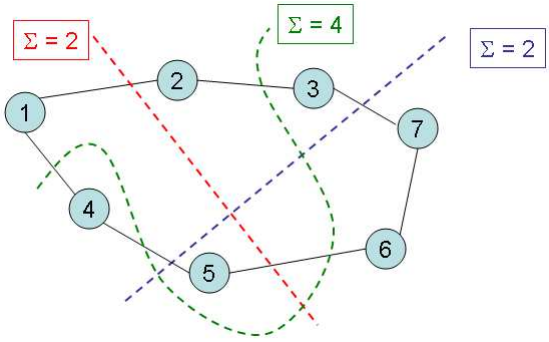


Figure 4: Absence of subtours. Σ denotes the number of arcs from $\hat{S}$ to $V \setminus \hat{S}$.

the example in Figure 5, where x_R^* has value 1 on the edges represented by a solid line, 1/2 on the dashed edges, and 0 on the edges not depicted.

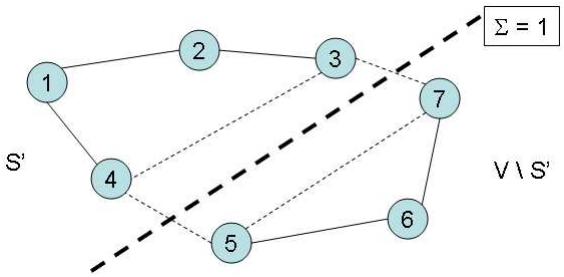


Figure 5: A constraint violated by a fractional solution.

The degree constraints are satisfied (as the sum of the entries of x_R^* at every node is 2), but the subtour elimination constraint corresponding to the set $S' = \{1, 2, 3, 4\}$ is violated. Then, in order to exclude (separate) this solution it would be sufficient to include the constraint:

$$\sum_{e \in \delta(S')} x_e \quad \geq \quad 2$$

But how can such a constraint be found in general? It is possible to show that, if we construct a graph $G_{sep} = (V, E)$ where every edge $e \in E$ has *capacity* $c_e = x^*_R(e)$, then $\sum_{e \in \delta(S)} x_e$ is precisely the value of a maximum flow from a node in S to a node in $V \setminus S$. Therefore:

> *The separation problem for finding a violated subtour elimination constraint can be reduced to a maximum flow problem.*[3]

In particular, we need to choose a node s as the source and solve $|V| - 1$ maximum flow problems (one for each possible choice of the sink node among the other nodes). If the maximum flow has value smaller than 2, one can find[4] a subset of nodes S giving a violated subtour elimination constraint.[5] If, on the contrary, the maximum flow is at least 2, no subtour elimination constraint is violated and thus x^*_R is an optimal solution of the continuous relaxation.

Summarizing, the availability of a solution technique for the separation problem of the subtour elimination constraints allows us to solve the continuous relaxation of the STSP with the method in Figure 6. We start from the problem with continuous variables and without subtour elimination constraints. At every iteration we solve a linear programming problem. Then we apply the maximum flow algorithm to solve the separation problem. If a violated subtour elimination constraint is found, this is included in the LP and we iterate. Otherwise, x^*_R satisfies *all* subtour elimination constraints and therefore it is an optimal solution for the continuous relaxation. Note that at every iteration two efficient algorithms are applied: one for the "master problem" (e.g., the simplex method), and the other for the separation problem. Finally, we remark that the convergence to an optimal solution is ensured by the finiteness of the number of subtour elimination constraints.

2.3 Constraint generation for the STSP

Although the optimal solution of the continuous relaxation satisfies all constraints $(STSP : 1)$ and $(STSP : 2)$, it is not, in general, a feasible solution, as the integrality of the variables is not guaranteed. Consider the example in Figure 7. The solution represented in the picture is feasible for the continuous relaxation, as it satisfies all constraints $(STSP : 1)$ and $(STSP : 2)$. The objective value of the solution is $z^*_R = 9$. An integer optimal solution on the same graph contains at most two edges of cost 1[6], and thus at least four edges

[3] More precisely, this is a maximum flow problem on an undirected graph that can be converted into the corresponding directed version by replacing every edge with two opposite arcs with the same capacity c_e.

[4] The details are omitted.

[5] Actually, there are more efficient methods for this particular maximum flow problem.

[6] It is not possible to construct a Hamiltonian cycle containing all the three edges of cost 1. To see this, note that these three edges are exactly all the edges between the two subsets of nodes $D = \{1, 3, 5\}$ and $P = \{2, 4, 6\}$. Suppose that there exists a Hamiltonian cycle containing the three edges, and imagine that we traverse it starting from a node in D. One of the three edges of cost 1 will lead us to P; then,

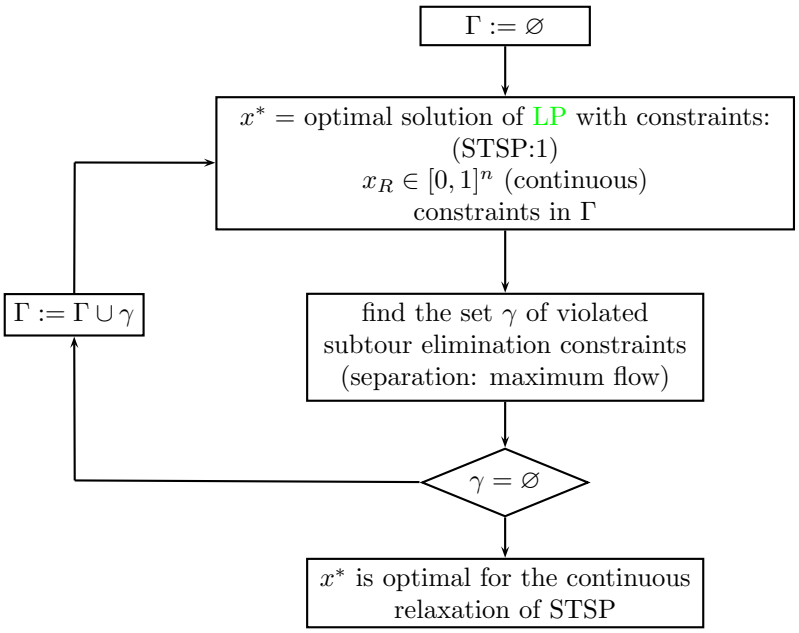


Figure 6: Solution of the continuous relaxation of the STSP.

of cost 2 (as every Hamiltonian cycle has $|V| = 6$ edges). Then the cost of every Hamiltonian cycle is at least 10, and therefore there is no optimal solution of the continuous relaxation that is also feasible for the STSP.

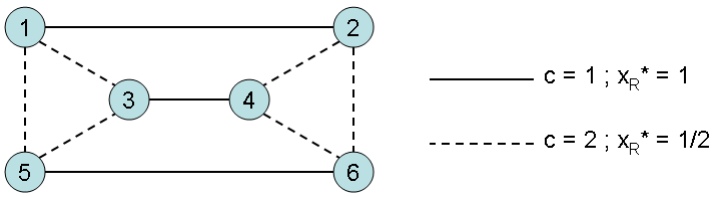


Figure 7: Fractional optimal solution of the continuous relaxation.

The above example shows that it is not sufficient to separate constraints $(STSP : 2)$ on the continuous relaxation and that after this phase we need to solve the problem with integer variables. One possibility is to use the constraint generation scheme starting with the subtour elimination constraints already added when solving the continuous relaxation. We obtain the solution scheme in Figure 8.

another edge of cost 1 will lead us back to D, and then with the third edge of cost 1 we move again to P. Since there are no more edges between D and P, it is not possible to go back to the starting node in D, a contradiction to the fact that C is a cycle.

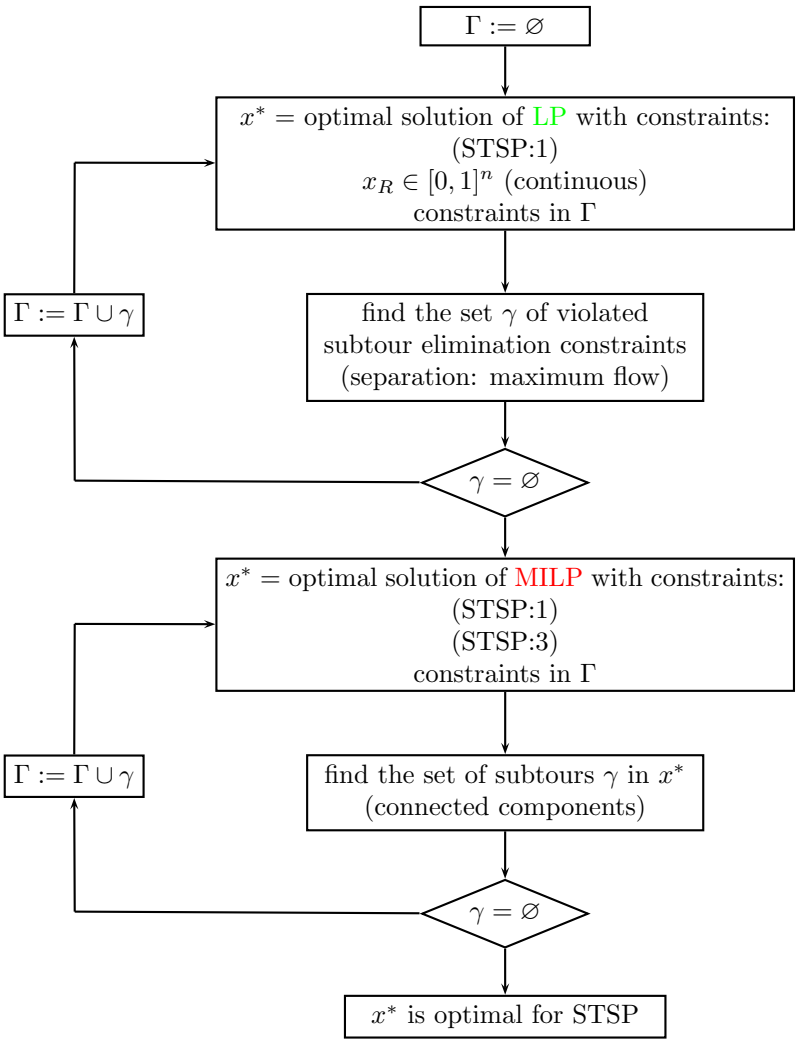


Figure 8: Constraint generation on LP and MILP for STSP.

The advantage of this scheme is that many subtour elimination constraints are generated via the *efficient* solution of linear programming problems and maximum flow problems. Thanks to this fact, the second phase, in which mixed integer linear programming problems are iteratively solved, converges faster to a Hamiltonian cycle. Overall, the computation time tends to be smaller: a larger number of linear programming problems (first phase) but a smaller number of mixed-integer linear programming problems (second phase) are solved. Nonetheless, this scheme is not completely satisfactory, as the second phase may require too many iterations.

2.4 Branch-and-Bound for the STSP

The best method currently available for the STSP is based on the following observation: since we have an *efficient* method for the solution of the continuous relaxation, we can

apply the classical Branch-and-Bound scheme for integer linear programming problems, in which a tree is constructed as follows:

- at every node of the Branch-and-Bound tree, a *lower bound* is obtained by solving the corresponding continuous relaxation with the constraint generation scheme described above;
- the following *branching* is adopted: if x_R^* is the optimal solution of the continuous relaxation and $x_R^*(e)$ is a fractional variable corresponding to some edge e, two child nodes are generated with the constraints $x_e = 0$ and $x_e = 1$.

The resulting scheme is very efficient, because the nodes inherit the subtour elimination constraints of the parent node and this drastically reduces the number of iterations (and thus of constraints) needed to compute the lower bound for the child node.

Moreover, the above scheme can be integrated with the use of specific valid inequalities for the STSP: these are inequalities that do not affect the set of feasible integer solutions but *cut off* some fractional solutions (similar to the *cover inequalities* for the knapsack problem, that we will see later). Thus, at every node of the Branch-and-Bound tree, both subtour elimination constraints and these specific inequalities are used, and this yield better lower bounds. With this approach STSPs on graphs with *millions (!!!)* of nodes have been solved (within few days of computation time).

3 Branch-and-Cut

The technique of strengthening the bound via the dynamic introduction of violated inequalities (or *cuts*) can be generalized to all (mixed) integer linear programming problems, provided that the following are available:

- a class of valid inequalities capable of yielding a *better formulation* of the problem (or even the ideal formulation, if possible), which cannot be used altogether because they are too many;
- an efficient separation algorithm for those inequalities.

As a special case, one may introduce violated inequalities only at the root node and then proceed with the classical Branch-and-Bound, where every node inherits the inequalities of the root node but no new inequality is separated. More generally, this scheme can be repeated at every node in order to strengthen the bound, by introducing inequalities that are violated at the current node (but not by the solution of the parent node). This technique, called Branch-and-Cut, has been successfully applied to several classes of combinatorial optimization problems. The key of the success is the study and the discovery of classes of valid inequalities capable of approximating the ideal formulation of the given problem, as well as the availability of efficient separation algorithm.