6 The Traveling Salesperson Problem

The Traveling Salesperson Problem (TSP)[1] is a very important discrete optimization problem that occurs in a variety of contexts. It is typically phrased as finding the cheapest round trip through n cities, where each connection between cities incurs a certain non-negative cost and no connection can be used twice. However, it occurs in contexts as varied as clustering, circuit design and genome assembly. Before we get into the details, let's get a few definitions over with.

Definition 1 (Hamilton cycle). *A Hamilton cycle is a cyclic subgraph of a graph G which visits each vertex exactly once. If G admits such a cycle, it is called a* Hamiltonian graph.

Deciding whether a graph is Hamiltonian is NP-complete.

Definition 2 (Euler cycle). *An Euler cycle is a trail (connected sequence of edges) of a graph G that visits each edge exactly once. If G admits such a cycle, it is called an* Eulerian graph.

If the graph is connected, then G is Eulerian if and only if all of its nodes have even degree. If it has more than one component, it cannot be Eulerian. This means that it is trivial to decide whether a graph is Eulerian. Also, finding such a cycle can be done in $O(|E|)$, e.g. using Hierholzer's algorithm.

Definition 3 (Spanning tree). *A spanning tree is a connected subgraph of G which contains no cycles and contains all nodes.*

Hamilton cycles, Euler cycles and spanning trees are all examples of *spanning subgraphs* of G, i.e. connected subgraphs containing all vertices. For weighted graphs, these definitions give rise to minimization problems, where we want to find the smallest spanning subgraph of a certain kind. Finding the cheapest Euler tour can be solved in polynomial time by first deciding whether such a cycle exists, and if it does, by solving the *Route Inspection Problem*, sometimes called the *Chinese Postman Problem*, which is polytime as well. The *minimum spanning tree* can be computed in polytime using simple algorithms such as Kruskal's.

Definition 4 (Traveling Salesperson Problem). *Given an edge-weighted graph, find the minimum-weight Hamilton path in G, if one exists.*

Note that we only talk about undirected graphs here. The extension to digraphs, where different costs are incurred for different directions is called the *asymmetric TSP*.

[1]This used to be called the "Travelling Salesman Problem", which is falling out of favor for obvious reasons. The P is therefore sometimes re-interpreted to stand for "person" instead of "problem", so the term "TSP problem" is not considered an example of RAS syndrome anymore.

TSP is NP-hard, since it contains the case of uniform edge weights, which amounts to finding a Hamilton cycle if one exists, the decision version of which is NP-complete (see above). Hence it is important to consider bounds on the optimal solution as well as approximation results.

6.1 Lower bounds

Let H^* be the optimal round trip which solves TSP. Let K be H^* with one edge removed. Obviously, $c(K) \leq c(H^*)$. However, notice that K is a, not necessarily minimal, spanning tree of our graph. Hence, the cost of the minimum spanning tree cannot be greater than that of K, so we have

$$c(\text{MST}) \leq c(K) \leq c(H^*)$$

In other words, the cost of the minimum spanning tree is a most as high as that of the optimal round trip. This idea is the basis for another bound. Say we knew the optimal tour H^*. Let P be the path obtained from H^* if we delete a node v as well as the two edges e_1, e_2 that are incident to v. Then, obviously,

$$c(H^*) = c(e_1) + c(e_2) + c(P)$$

Since we know neither H^* nor P, we have to derive a bound instead. Let $A \leq c(e_1)+c(e_2)$ and $B \leq c(P)$, then $A+B \leq c(H^*)$ is a lower bound on the TSP solution. Such an A can be found by picking the two cheapest edges incident to v, and B can be defined using the weight of the minimum spanning tree in $G \setminus \{v\}$. Hence

$$c_v^{\text{1-tree}} := \min_{e \in E} \left\{ c(e_1) + c(e_2) \,\middle|\, e_1, e_2 \in \delta(v), e_1 \neq e_2 \right\} + c(\text{MST}(G \setminus \{v\}))$$

is a lower bound on H^*. This is known as the *1-tree bound*.

The 1-tree bound is the basis for the *Held-Karp bound*. The problem with MST-based bounds is that the difference between $c(\text{MST})$ and $c(H^*)$ can be quite large, since the MST has no degree-constraints and thus far more flexibility to pick cheap edges. We would therefore like to modify our problem in a way that makes this gap small, by favoring MST for which the node degrees are close to 2. For that purpose, we try to change edge weights such that the cost for the MST increases as much as possible, while the cost of *every* Hamilton cycle in the graph changes by the *same* known constant Λ. We can then subtract Λ from the 1-tree weight to get a tighter lower bound. The first property is not hard to achieve, however, changing the edge weights such that the second property holds is a little trickier. Obviously, we could pick some constant λ and add it to each edge weight, thereby increasing the cost for H^* by λn. Unfortunately, this barely gets us anywhere, since the MST will contain the exact same edges as before.

This means we need different λ_e for different edges, however, we cannot pick those λ_e arbitrarily, since we don't know what edges are contained in H^*, and we don't want to alter the result. The solution to this conundrum is in the following observation: every Hamilton cycle will have to pass through every node in the graph exactly once. If we associate each node v with a constant λ_v, then $\sum_{v\in V} \lambda_v$ is the same for all Hamilton tours. To obtain new edge weights, we simply add λ_v to each edge incident to v, so that for all edges $e = (v, w) \in V$, we get $\lambda_e := \lambda_v + \lambda_w$.

Theorem 1 (Held-Karp bound). *Let $G = (V, E)$ be a graph with edge weights $c(e)$ for all $e \in E$. Let λ_v be some constant for each node $v \in V$. Let $G' = (V, E)$ with edge weights $c(e') := c(e) - \lambda_u - \lambda_v$ for all $e = (u, v)$. Let C be the 1-tree bound for G'. Then the* Held-Karp bound

$$C + 2\sum_{v\in V} \lambda_v$$

is a lower bound for $c(H^*)$.

It might seem a bit strange to phrase this approach by subtracting a negative number λ_v instead of adding a positive one. As we will see later, the Held-Karp bound is an example of Lagrange relaxation, and the negative sign comes from that proof.

There are various schemes to improve λ_e by iteratively changing λ_v and recalculating the MST. The general idea is to punish high node degrees in the MST. Whenever node v has a degree higher than 2, λ_v is changed such that edges incident to v become costlier. On the other hand, if v is a leaf, i.e. its degree is 1, λ_v is changed to make its incident edges cheaper.

6.2 Approximation results

Aside from its NP-hardness, the notoriety of TSP is due to the following result:

Theorem 2 (TSP inapproximability). *Unless P=NP, there exists no polytime α-approximation for TSP for any constant $\alpha \geq 1$.*

Proof. This is easy to prove by contradiction. We know that deciding whether a graph G is Hamiltonian or not is NP-complete. We turn G into a weighted graph G_α, by adding a weight of 1 to each of its edges, and adding an edge of weight αn for each edge missing in G (n is the number of nodes). Now assume we had an α-approximation heuristic for TSP. In case G has a Hamilton cycle, it would correspond to the optimal solution for TSP in G_α, and the cost of the TSP tour would be n. The heuristic would yield a result of cost at most αn. If G is not Hamiltonian, the shortest round-trip would have to go through at least one of the new edges with weight αn in G_α, so its cost would be greater than αn, and the heuristic cannot do better than the optimal solution. Hence, we would always be able to distinguish between cases where G is Hamiltonian or not in polynomial time, which is NP-complete. □

In other words, it is NP-hard to even find a constant-factor approximation for TSP! This is obviously bad news, and demonstrates the importance of algorithms to solve NP-hard problems optimally with reasonable effort. There are, however, special cases in which polytime constant-factor approximation is possible and efficient:

Definition 5 (Metric TSP). *A TSP instance is called* metric *iff it has a complete graph, i.e. there is an edge between any two node, and for any set of three nodes, the edge weights observe the triangle inequality, i.e. $c(u, w) \leq c(u, v) + c(v, w)$.*

The most important case of this is the *Euclidean TSP*, in which the nodes represent points in the plane and edge weights represent the Euclidean distance (length of line segment between points). Metric TSP has two important approximation schemes: the *MST heuristic* yields a 2-approximation, and the *Christofides heuristic* yields a $\frac{3}{2}$-approximation.

The MST heuristic is very simple:

1. Compute a minimum spanning tree on G.
2. Pick an arbitrary start node and walk along the edges, making the sharpest possible right-turn at each vertex (alternatively, choose the sharpest possible left turn). This creates a cycle of nodes, which visits some nodes more than once.
3. To create the Hamilton cycle, simply ignore all but the first occurrence of each node, and take the set of edges between adjacent nodes in that cycle.

It is easy to see that this is a Hamiltonian cycle, since the MST contains all nodes, and by ignoring all but the first occurrences of a node, we have a cycle. For the approximation ratio, note that in step 2, we use each edge twice, so the cost incurred in step 2 is $2c(\text{MST})$. By the triangle inequality, the shortcuts we create in step 3 cannot be longer than the path we replace, so we don't increase the total cost in step 3. Since the cost of an MST is a lower bound on the cost of the shortest Hamilton cycle, we obtain a 2-approximation of TSP.

The Christofides heuristic uses a similar idea, but adds more light edges to the initial set we traverse:

- Compute a minimum spanning tree T for G.
- Select all nodes in G which have an odd degree in T (there will always be an even number of them).

- Calculate a minimum weight maximum matching M for those nodes. This will always be a perfect matching.
- Combine T and M into a multigraph C, i.e. there can be multiple edges between pairs of nodes. The result is Eulerian, since we add exactly one edge to each odd-degree vertex in T.
- Compute an Euler cycle in C.
- Compute a Hamilton cycle based on this Euler cycle by skipping over repeated nodes, just like in the MST heuristic.

One can show that, due to the triangle inequality, this yields a $\frac{3}{2}$-approximation for metric TSP.

For many decades, the Christofides heuristic was the best known approximation algorithm for metric TSP. However, it was only known that the metric TSP cannot be approximated within a factor of $\frac{123}{122}$, which suggested that $\frac{3}{2}$ might not have been the best we can do. Indeed, there have been quite a few breakthroughs within the last years which led to better approximation ratios; the following article provides a high-level overview, as well as references for these exciting developments: https://www.wired.com/2013/01/traveling-salesman-problem/. A more technical, peer-reviewed work can be found at https://link.springer.com/article/10.1007/s00493-014-2960-3.

6.3 ILP formulation

The ILP formulation of TSP uses binary variables to select edges, minimizing the total edge weight. Typically, the following set of constraints is used:

Degree constraints (DC) Each node must be entered and exited exactly once, so its degree in the solution must be 2. These are also called *2-factor constraints*.

Subtour eliminination constraints (SEC) Degree constraints do not guarantee that the result is a single cycle, but could consist of the union of smaller cycles called *subtours*. SEC must ensure that any feasible solution is a single cycle.

There are various ways to define SEC once the degree constraints are in place. The first one is often called the *outer formulation of SEC*: Consider a cycle (not necessarily Hamiltonian) on a node set $C \subseteq V$. There are no edges between V and $V^{\complement}$, due to the DC. On the other hand, if we choose any smaller subset $S \subset C$, there will be at least two edges connecting nodes in S to nodes outside of S. In order to eliminate subtours, we hence require that any true subset of V has at least two edges "crossing its boundaries". It is easy to see that this eliminates subtours, by contradiction: if a subtour existed on some $C \subset V$, then C would violate that constraint. The ILP then becomes

$$\begin{array}{rl} & \min \mathbf{c}^\intercal \mathbf{x} \\ \text{s.t.} \quad \forall v \in V : \sum_{e \in \delta v} x_e & = 2 \\ \emptyset \subset S \subset V : \sum_{v \in S, w \in S^{\complement}} x_{v,w} & \geq 2 \\ x_e & \in \mathbb{B} \end{array}$$

An equivalent, and somewhat easier SEC is the so-called *inner formulation*. Note that any cycle (Hamiltonian or not) has as many edges as it has nodes. We can hence require for all subsets S except V itself that it has less edges than nodes, so it is impossible to close a cycle on S:

$$\begin{array}{rl} & \min \mathbf{c}^\intercal \mathbf{x} \\ \text{s.t.} \quad \forall v \in V : \sum_{e \in \delta v} x_e & = 2 \\ \emptyset \subset S \subset V : \sum_{v,w \in S} x_{v,w} & \leq |S| - 1 \\ x_e & \in \mathbb{B} \end{array}$$

6.4 Cutting-plane method

Typically, the LP relaxation can be used as an easy bound for the ILP solution. However, the ILP formulation above presents us with an additional problem: Since we need one SEC for each subset S, we have exponentially many constraints, so even formulating the ILP and LP relaxation explicitely takes exponential time and is therefore as hard as solving TSP by brute force!

In convex optimization, we often encounter problems problems situations like these. The *cutting plane method (CPM)* is a general approach for tackling these problems. Say we have a hard problem P and an easy to solve relaxation R of P. R is obtained by dropping or relaxing constraints from P. For instance, R could be the LP relaxation of an ILP, where the integrality constraints are relaxed. We can also have situations where P is an LP, but has way too many constraints (e.g. exponentially many), so that even checking feasibility explicitly is prohibitive. In that case, R could be P with some constraints removed. The CPM proceeds as follows:

1. Obtain $\mathbf{x}_R^*$ by solving R.

2. Find a halfspace constraint (the "cutting plane") which is obeyed by all feasible solutions to P, but violated by $\mathbf{x}_R^*$. This "cuts off" a subset of the feasible region of R, including $\mathbf{x}_R^*$, without "cutting into" the feasible region of P (finding such a cut is called the *separation problem*).

3. Repeat until $\mathbf{x}_R^*$ is feasible in P, thus solving P, the solution is close enough or no further cut can be obtained.

In ILP, this approach is useful for refining LP relaxations. Given some set of vectors $M \subseteq \mathbb{R}^n$, let $\operatorname{conv} M$ be the smallest convex set such that $M \subseteq \operatorname{conv} M$. This is called the *convex hull* of M. For integer points, the convex hull of all feasible integer points is a polytope contained within the polytope of the LP relaxation, and thus could in principle be described by a finite set of half-space constraints just as an LP. The problem is that the inequalities are not explicitly known from the LP relaxation. Instead, we can use the CPM to solve an ILP:

1. Obtain $\mathbf{x}_{\mathrm{LP}}^*$ as the solution of the LP relaxation.
2. Find a halfspace constraint which is obeyed by all feasible ILP solutions, but violated by $\mathbf{x}_{\mathrm{LP}}^*$. This "cuts off" the vertex $\mathbf{x}_{\mathrm{LP}}^*$ from the LP polytope without "cutting into" the ILP polytope.
3. Repeat until $\mathbf{x}_{\mathrm{LP}}^*$ is integer and thus optimally solves the ILP, $\mathbf{c}^\mathsf{T}\mathbf{x}_{\mathrm{LP}}^*$ is good enough, or no further cut can be obtained.

There are many different ways to add these constraints, depending on the problem at hand. If the reason for using the CPM is the number of constraints in the problem itself, we simply add those which is violated by the relaxed solution. If we need to find unknown cutting planes for ILP, one of the most generally applicable methods is using *Chvátal-Gomory cuts*: if $\lambda^\mathsf{T}\mathbf{A}$ for some vector $\lambda \in [0,1]^n$, then $\lambda^\mathsf{T}\mathbf{A}\mathbf{x}$ i also integer for integer $\mathbf{x}$. Hence, if we have a general constraint of the form $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ with $\mathbf{b} \in \mathbb{R}^n$, we also know that

$$\lambda^\mathsf{T}\mathbf{A}\mathbf{x} \leq \lfloor \lambda^\mathsf{T}\mathbf{b} \rfloor$$

For instance, if we have constraints $x_1 + 2x_2 \leq 4$ and $x_1 \leq 1$, then for $\lambda = (\frac{1}{2}, \frac{1}{2})^\mathsf{T}$, we get $x_1 + x_2 \leq \lfloor \frac{5}{2} \rfloor = 2$. Finding good λ is not always easy, and we might still end up adding exponentially many cutting planes when trying to solve an ILP (NP-hard) using a series of LP relaxations (polytime). Furthermore, this approach alone is impractical due to numerical issues.

For many ILP, however, there exist better, problem-specific cutting planes. For the TSP, the idea is to simply ignore the SEC in the beginning and add them one-by-one to separate a non-integer ILP solution from the integer polytope. This way, we can avoid adding *unnecessary* cutting planes and drastically reduce their numbers. Solving the LP relaxation, one of two things can happen: either, the non-zero edges separate into different subtour components $C_1, C_2, \ldots$, in which case the SEC for each C_i can be added as a separating hyperplane to the problem. However, after a while no such clear components will occur, so we must find a better way to find subsets violating the SEC; in other words, we need to find the edge set of minimum weight that separates V into S and $S^\complement$; if the value of the cut is < 2, S and $S^\complement$ violate an SEC. This is called the *minimum weight cut problem* for weighted graphs. Remember that we can solve the *minimum s-t cut problem* in polytime. Obviously, a minimum weight cut is just a minimum s-t cut for some $s \in S$ and $t \in S^\complement$, and can be find by taking the minimum s-t cut among all $s, t \in V, s \neq t$. If the components are separated, the minimum cut is 0 There are more efficient algorithms than that, but the important message is that we can find SEC violations in polytime. We hence have an algorithmic approach for solving the LP relaxation of TSP (algorithm 1)

Algorithm 1 A cutting-plane method to solve the LP relaxation of TSP. The goal is to avoid adding huge numbers of subtour elimination constraints, by cutting off SEC-violating solutions.

```
Solve LP relaxation without SEC to obtain edge-weighted graph $G$.
while Minimum cut in $G$ is $< 2$ do              ▷ Some SEC is violated by $S$ and $S^\complement$.
    Add SEC for vertex sets $S$ and $S^\complement$ to LP.        ▷ This is a cutting plane.
    Solve LP relaxation to obtain edge-weighted graph $G$.
All constraints (except for integrality) are satisfied.
```

After finding a solution to that reduced problem, we try to find a constraint that is violated by that solution and add it to our problem formulation. We solve again and continue to do so until we get a solution that is a Hamilton cycle.

6.5 Lagrangian relaxation

There exists a different relaxation technique. Rather than enforcing constraints, violations are penalized in the objective function. For instance, the minimization LP

$$\begin{aligned} &\min \mathbf{c}^\mathsf{T}\mathbf{x} \\ &\text{s.t.} \quad \mathbf{A}\mathbf{x} \geq b \end{aligned}$$

can be relaxed by adding a positive penalty term whenever $\mathbf{A}\mathbf{x} < \mathbf{b}$:

$$\begin{aligned} &\min \mathbf{c}^\mathsf{T}\mathbf{x} + \lambda^\mathsf{T}(\mathbf{b} - \mathbf{A}\mathbf{x}) \\ &\text{s.t.} \quad \lambda \geq 0 \end{aligned}$$

For TSP, there exists a surprising Lagrange relaxation that allows us to find and improve feasible solutions using combinatorial graph algorithms, without ever explicitly formulating the exponentially many SEC. In fact, this formulation turns out to be the Held-Karp bound described earlier. The idea is that, instead of dropping the integrality constraints as in LP, we relax the degree constraints $\sum_{e\in\delta v} x_e = 2$. As a result we get an ILP, but one which can be solved exactly using a combinatorial algorithm. Pick some vertex $\hat{v}$, and consider the following problem:

$$\min \mathbf{c}^\intercal \mathbf{x} + \sum_{v\in V} \lambda_v \left(\sum_{e\in\delta(v)} x_e - 2 \right)$$

$$\begin{aligned} \text{s.t.}\quad \lambda_{\hat{v}} &= 0 \\ \textstyle\sum_{e\in E} x_e &= n \\ \textstyle\sum_{e\in\delta(\hat{v})} x_e &= 2 \\ &\text{SEC} \\ \mathbf{x} &\in \mathbb{B}^n \end{aligned}$$

Notice that we added the constraint $\sum_{e\in E} x_e = n$. While it did not occur in the original ILP formulation, it was implied by the degree constraints, since we had n nodes, each node had to have 2 edges and each edge is incident to 2 nodes. It was thus a valid constraint in the original ILP and we could have added it there without changing the feasible region – it would simply have been redundant.

Let's take a closer look at those new constraints. Since we select two edges incident to $\hat{v}$ and select n edges in total, we select $n-2$ edges between $n-1$ nodes in $G \setminus \{\hat{v}\}$. The SEC ensure that we have no cycles. By the outer SEC, we also know that

$$\forall \emptyset \subset S \subset V \setminus \{\hat{v}\} : \sum_{e\in\delta S} x_e \geq 1$$

so the edges on $G \setminus \{\hat{v}\}$ form a connected graph. Since it has no degree constraints, no cycles and is connected, these constraints describe a spanning tree on $G \setminus \{\hat{v}\}$! Now lets look at the objective function:

$$\min \mathbf{c}^\intercal \mathbf{x} - \sum_{v\in V} \lambda_v \left(\sum_{e\in\delta(v)} x_e - 2 \right)$$

$$= \min \sum_{e\in E} c_e x_e - \sum_{v\in V} \lambda_v \sum_{e\in\delta(v)} x_e + 2\sum_{v\in V} \lambda_v$$

Since $\sum_{v\in V} \lambda_v \sum_{e\in\delta(v)}$ sums over all nodes and incurs a weight to each incident edge, every edge is hit twice, once by each of its incident nodes, thus

$$\sum_{v\in V} \lambda_v \sum_{e\in\delta(v)} x_e = \sum_{(u,w)=e\in E} (\lambda_u x_e + \lambda_w x_e)$$

so the objective function becomes

$$\min \sum_{e\in E} c_e x_e - \sum_{(u,w)=e\in E} (\lambda_u x_e + \lambda_w x_e) + 2\sum_{v\in V} \lambda_v$$

$$= \min 2\sum_{v\in V} \lambda_v + \sum_{(u,w)=e\in E} (c_e - \lambda_u - \lambda_w) x_e$$

As we can see, the solution to the Lagrangian relaxation is a minimum spanning tree with edge weights modified by node weights λ_v. This is exactly the Held-Karp bound, and we can solve the ILP obtained by Lagrange-relaxing the degree constraints can be solved efficiently by a combinatorial algorithm: pick the two cheapest edges incident to $\hat{v}$, then solve MST on $G \setminus \{\hat{v}\}$! There are ways to find the best λ for Lagrangian relaxations using duality, which we did not have time to talk about in the lecture. For the TSP, it amounts to the approach we briefly sketched for the Held-Karp bound. It can be shown that, for the best λ the solution is at least as good as the LP relaxation; in practice, it often turns out to be much better.

7 Branch-and-bound

Branch-and-bound (B&B) refers to a large class of algorithms for solving hard optimization problems optimally, or at least within some provable bound. It comes in many shapes and forms. For a minimization problem over a feasible domain Ω, we consider the following setting:

- It is hard to compute an optimal solution directly.
- Subsets of the feasible region can be described efficiently, e.g. by a reasonably sized set of constraints.
- For specific subsets of the feasible region, it is easy to calculate a lower bound f_L and find a feasible solution within the set as an upper bound.

The properties above are used to recursively split the feasible set (the branching step) and obtain bounds on f^* for each of those regions (the bounding step). Specifically, we maintain the best (global) feasible solution found so far across all regions as an upper bound, and compute a (local) lower bound for each region. Whenever the local lower bound is greater or equal than the best solution found so far, we now that this region cannot contain any solution that would improve upon the one we already have. Thus, the region and all its subsets can be safely ignores. This cutting off of subtrees in the recursion is called *pruning*. While B&B is still an exponential-time algorithm for most instances, pruning decreases the exponent and yields more acceptable running times.

Algorithm 2 A general, non-recursive branch-and-bound algorithm for minimization problems over a feasible domain Ω. A is a general container supporting push() and pop() operations. If A is a stack, the search tree is expanded in depth-first search (DFS) order. If it is a queue, it is expanded in breadth-first search (BFS) order. A could also be a more complex container class, such as a priority queue (which yields a best-first search). Note that A is used here for purposes of generality and clarity. In implementations, it is often not explicitly used; instead, B&B is implemented recursively using the appropriate call order for different traversal schemes.

```
procedure BRANCH&BOUND($\Omega$)
    $\mathbf{x}^*$ arbitrary                         ▷ Current optimal solution, possibly infeasible.
    $f^* \leftarrow \infty$                          ▷ Global upper bound on $f^*$.
    $A \leftarrow \{\Omega\}$                        ▷ Container for unexpanded subregions.
    while $|A| > 0$ do                               ▷ Potential solutions left.
        $\Phi \leftarrow$ pop($A$)                   ▷ Expand search tree by a subregion $\Phi$ (recurse).
        if $\Phi$ contains a feasible solution then
            $(f_L, \mathbf{x}_L) \leftarrow$ lowerBound($\Phi$)   ▷ Get lower bound on current subregion.
            if $f_L < f_U^*$ then    ▷ Potentially better solution in subregion? Bound if not.
                if $\mathbf{x}_L$ is feasible and $f_L < f^*$ then ▷ Found better solution in subregion.
                    $f^* \leftarrow f_L$                 ▷ Update objective.
                    $\mathbf{x}^* \leftarrow \mathbf{x}_L$   ▷ Update optimal solution.
                    $f_U^* \leftarrow f^*$               ▷ Bound future suboptimal solutions.
                else                     ▷ No better solution found, have to branch on subsets.
                    $(f_U, \mathbf{x}_U) \leftarrow$ feasibleSolution($\Phi$)   ▷ Get current upper bound.
                    $f^* \leftarrow \min\{f^*, f_U\}$        ▷ Tighten global upper bound.
                    Partition $\Phi$ into $\Phi_1, \Phi_2, \ldots$   ▷ Branch into subsets.
                    Push all $\Phi_i$ to $A$              ▷ Mark subsets as unexpanded.
    return $(f^*, \mathbf{x}^*)$                     ▷ $f^* = \infty$ means the problem was infeasible.
```

Algorithm 2 contains the pseudocode for a general version of B&B for a minimization problem.

It is not unusual to improve bounds on a subregion using the cutting-plane algorithm. The extra effort often turns out to be worthwhile, in particular to obtain tight bounds early on, which prunes off large parts of the search tree. Such an approach is called *branch-and-cut*, and has been successfully applied to TSP to great benefit.

Since we would like to enforce the encounter of feasible solutions at some point, the right branching scheme is crucial. For ILP, we know that the solution cannot contain non-integer values. If we use LP relaxation to get a lower bound, we simply choose any coordinate $x_i = c$, where c is non-integer. We branch by adding the first constraint $x_i \leq \lfloor c \rfloor$ to create Φ_1, and $\lceil c \rceil \leq x_i$ to create Φ_2. Note that this introduces hyperplanes orthogonal to the x_i-axis, and hence an integer facet for the feasible polytope. The hope is that, by accumulating such hyperplanes, at some point further down the search tree the optimal solution to the LP relaxation will be integer and thus a feasible ILP solution. If we split in at an arbitrary position $\lfloor c \rfloor < p < \lceil c \rceil$ instead, we might end up recursing forever (at least until we exhaust the machine precision for floating point numbers).

Given the results discussed in previous lectures, we get the following bounds for a primal minimizing ILP:

- Solution to LP relaxation
- Solution to LP relaxation of dual (same as above by strong duality)
- Any feasible solution for dual and its LP relaxation
- Solution to Lagrangian relaxation
- Solutions to any other relaxation technique