ENGRI 1101 **Engineering Applications of OR** **Fall 2016** **Handout 10**

Integer Programming and the TSP

For many of the linear programming formulation problems that we have considered in this course, we have come to the conclusion that it would have been more appropriate to formulate the problem as an integer linear program; that is, the variables in question should be restricted to take only integer values, and we really wish to find the optimal solution subject to this additional constraint.

Why are we tempted to try a linear programming formulation? The main reason is that whereas the simplex method can be used to solve large linear programs quite quickly, there is no universally successful algorithm to solve integer programs of comparable size: all known algorithms simply take too long. Given the difficulty in solving the integer program, it is extremely tempting to hope that the linear program will produce a solution that is good enough.

There are cases in which the linear programming relaxation is sufficiently good. Sometimes the the optimal solution x^* to the linear program is already integer. What does this mean? It means that this solution is also optimal for the integer program. Why? For any minimization problem, the optimal value for the linear programming relaxation of an integer program is always less than or equal to the optimal value for the integer program itself. This can be explained as follows: consider the optimal solution to the integer program (whatever it might be); this is also a feasible solution to the linear relaxation, and hence the optimal value to the LP is at most this solution's value; but this solution's value is the optimal value for the integer program. We have just argued that the optimal value for a (minimization) linear relaxation is always at most the optimal value for the original integer program.

But why does this prove that x^* is an optimal solution to the integer program? First, it is an integer solution to the linear relaxation; hence, it is a feasible solution to the integer program. Second, it is an optimal solution to the linear program, and so we know that no feasible integer (or even fractional) solution can have value better than the value of x^*. But this means that x^* is an optimal solution to the integer program.

There is one additional conclusion to be drawn here. Suppose that we have a feasible integer solution y. Even if the optimal solution to the linear relaxation, x^*, is not integer, we can use its value to limit the extent to which we can find feasible integer solutions that are better than y. For example, if y has cost 17, and x^* has cost 15, we know that no feasible integer solution has cost less than 15, and hence the solution y has cost at most 2 more than optimal. In fact, y might be optimal and we simply can't conclude that yet, or else there might exist better solutions. But the bottom line is that we know that we can't improve the cost by more than 2.

We will now make some of these ideas a bit more concrete by applying them to the traveling salesman problem. In order to do this, we first need an integer linear programming formulation of this problem. We shall only consider the symmetric traveling salesman problem, in which the input consists of a cost matrix $C = (c_{ij})$ where the costs have the property that $c_{ij} = c_{ji}$, the distance from city i to city j is always the same as the distance from city j to city i.

It is easy to see that the symmetric traveling salesman problem can be restated as follows: we are given an undirected graph $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$ which is complete, i.e., for each pair of nodes $i \neq j$, there is the edge $\{i, j\} \in E$; each edge $e = \{i, j\}$ has a given cost $c(\{i, j\})$; we wish to find a cycle that visits each node exactly once (a tour) for which the total cost is minimized.

To start the formulation, we need to choose decision variables. In this case, we need to decide, for each edge $\{i, j\}$, whether that edge is used in the optimal tour. So, we shall introduce a variable for each edge. It might be natural to just write that we have a variable x_{ij} for each edge, but this might introduce two variables for each edge, since we can write the same edge as both $\{i, j\}$ and $\{j, i\}$. To get around this, we will only introduce a variable whenever $i < j$. Each variable x_{ij}, $1 \leq i < j \leq n$, will indicate whether that edge is used in the tour in the following way: either $x_{ij} = 1$, which means that it is used in the tour, or $x_{ij} = 0$, which means that it is not. We can capture this restriction by

$$0 \leq x_{ij} \leq 1, \text{ integer}, \quad \text{for each } i = 1, \ldots, n-1;\ j = i+1, \ldots, n. \tag{1}$$

If we consider the set of edges $\mathcal{C}$ in any particular feasible solution, in any tour, it satisfies the property that for

each node $i = 1, \ldots, n$, there are exactly two edges $\{i, j\} \in E$ and $\{i, k\} \in E$, for some $j \neq k \in V$, such that $\{i, j\} \in \mathcal{C}$ and $\{i, k\} \in \mathcal{C}$. In words, that means that we have selected a subset of edges such that if we look at any node i, then there are exactly two edges selected that have node i as one of their endpoints; any subset of edges in the graph that satisfies this property is called a *2-matching*.

Is it true that if a set of edges is a 2-matching, then it is a feasible solution for the traveling salesman problem? No, it is not true. Consider the example in Figure 1, in which the bold edges form a 2-matching.

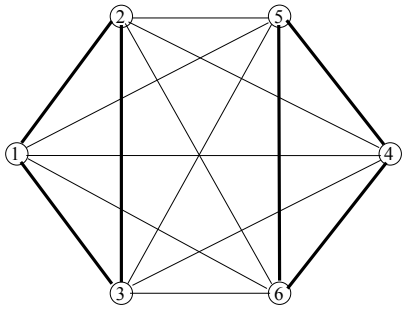

Figure 1: A 2-matching

However, the converse is true: for any feasible solution to the traveling salesman problem, the edges of this tour form a 2-matching. We want to capture the restriction that a set of edges form a 2-matching by linear constraints in our decision variables; that is, we should express the fact that, for each node i, of the $n - 1$ edges $\{1, i\}, \{2, i\}, \ldots, \{i, n\}$ (note that $\{i, i\}$ is not an edge), exactly two of them are selected. This can be done by requiring that

$$\sum_{j=1}^{i-1} x_{ji} + \sum_{j=i+1}^{n} x_{ij} = 2, \quad \text{for each } i = 1, \ldots, n. \tag{2}$$

(Make sure that you understand why this summation was divided into two parts; this has to do with the fact that we only have variables x_{ij} when $i < j$.) The total cost of any subset of edges $\mathcal{C}$ is the sum of the costs of the edges in $\mathcal{C}$, $\sum_{\{i,j\} \in \mathcal{C}} c_{ij}$). If we instead specify a subset of edges by our decision variables, x_{ij}, we get that the total cost is equal to

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} x_{ij}. \tag{3}$$

Putting these pieces together, we have obtained an integer programming formulation of the minimum-cost 2-matching problem: minimize (3) subject to the constraints (1) and (2).

How does the cost of the optimal 2-matching compare to the cost of the optimal tour? The optimal tour $\mathcal{T}$ is a feasible 2-matching. Hence, the minimum-cost 2-matching costs at most as much as $\mathcal{T}$; the minimum cost of a 2-matching is less than or equal to the minimum cost of a tour.

Let us return to the issue of formulating the traveling salesman problem as an integer programming problem. We know from the example in Figure 1 that constraints (1) and (2) are not sufficient to ensure that a feasible setting of the decision variables specify a tour. What goes wrong in this example? Instead of having one big tour that visits all of the nodes, there are two separate pieces, which are usually called *subtours*.

To figure out how to overcome this difficulty, focus on the two subsets of nodes $\{1, 2, 3\}$ and $\{4, 5, 6\}$. *Any* tour for the traveling salesman problem must at some point cross over from the first set of nodes, and then later return back. So if the variables x_{ij} do specify a feasible tour, then at least two of

$$x_{14}, x_{15}, x_{16}, x_{24}, x_{25}, x_{26}, x_{34}, x_{35}, x_{36}$$

must be set to 1. (To make sure that you are following everything, take this moment to find a tour in which exactly 2 of these variables would be set to 1, another in which exactly 4 of them would be set to 1, and yet another tour

in which exactly 6 of them would be set to 1.) So it is natural to add the constraint that

$$x_{14} + x_{15} + x_{16} + x_{24} + x_{25} + x_{26} + x_{34} + x_{35} + x_{36} \geq 2.$$

However, while this one constraint makes the solution indicated in Figure 1 infeasible, there are still other feasible solutions to this expanded integer program that are 2-matchings, but not tours. One such example is the subset of edges

$$\{\{1,2\},\{1,5\},\{2,5\},\{3,4\},\{3,6\},\{4,6\}\}.$$

Of course, this would now suggest a new constraint. (Figure out what it is!!) At first glance, it might seem that this process will continue indefinitely, but it doesn't. To make sure that a 2-matching does not have *any subtours*, we can simply require that for any subset of nodes S (on which we might form a subtour), a tour must use at least two of the edges which have one endpoint in S, and the other endpoint outside of S (or equivalently, in $V - S$). For each $S \subset \{1, 2, \ldots, n\}$, we let $\delta(S)$ denote the set of edges that connect between S and $V - S$, or more formally, $\{(i,j) : |\{i,j\} \cap S = 1,\ 1 \leq i < j \leq n\}$. (The expression $|\{i,j\} \cap S| = 1$ just means that exactly one of i and j is in the set S.) In fact, when we partition the nodes into two parts, S and $V - S$, we can always view S as being the side with fewer nodes, and so we can just add the constraints

$$\sum_{(i,j)\in\delta(S)} x_{ij} \geq 2, \quad \text{for each } S \subset V,\ 3 \leq |S| \leq n/2. \tag{4}$$

These constraints are often referred to as the *subtour elimination constraints*, or the *cut constraints*. Notice that we have also not included a constraint for each subset S of sizes 1 or 2. The reason for this is that those constraints are satisfied automatically, when all of the constraints specified in (4), (2), and (1) are satisfied. Another way to understand this apparent omission is that, in any 2-matching, there cannot be a subtour that contains exactly 1 or 2 nodes. So we have now completed our integer programming formulation of the traveling salesman problem: minimize (3) subject to (2), (4), and (1).

How big is this integer programming formulation? For example, suppose that there are 100 nodes in your input to the traveling salesman problem; how many constraints are there in this formulation? Counting just the subtour elimination constraints, there are roughly $2^{50} > 10^{15}$ constraints. Clearly, this is beyond the memory capabilities of current computing, not even considering the difficulty in solving the integer program.

However, we want to solve this integer program. As a first step, we also want to solve the linear relaxation of this integer program, in which the constraints (1) are replaced by

$$0 \leq x_{ij} \leq 1, \quad \text{for each } i = 1, \ldots, n-1;\ j = i+1, \ldots, n. \tag{5}$$

We will now explain how this very large linear program can still be solved efficiently, by using a so-called *cutting plane* approach to supplement the usual simplex method.

We will repeatedly solve a linear program with the objective to minimize (3) subject to a set of constraints $\mathcal{S}$ that grows iteration by iteration. Initially, $\mathcal{S}$ consists of the constraints (2) and (5). This is a linear program of relatively modest size, and so we can use the simplex method to find an optimal solution to it. Each iteration works as follows. We find an optimal solution x^* to the current linear program. If this optimal solution x^* satisfies all of the constraints (4), then it is an optimal solution to the linear relaxation of the TSP. (Make sure you understand why this is true!) If there does exist some constraint in (4) that x^* fails to satisfy, then we add this *one* constraint to $\mathcal{S}$, and start the next iteration.

Of course, in the worst of all possible worlds, we might simply add all of the constraints (4) in this one at a time fashion. However, in practice, this is an extremely efficient method, and terminates after only a few iterations. Of course, we still need an efficient way to decide if a given solution satisfies all of the constraints (4) without checking them one-by-one, but it is not hard to show that the minimum cut procedure that we studied earlier in the course can be adapted to either identify a violated subtour elimination constraint, or else prove that the current solution satisfies all of these constraints.

Why is it called a cutting plane approach? The reason is best understood by considering Figure 2. This figure shows a feasible region for a linear program. Suppose that we are trying to minimize x subject to the constraints shown. If we optimize subject to all of the constraints except the one indicated by a dashed line, then get the point indicated by a (*) as an optimal solution. This point is not feasible with respect to all of the constraints.

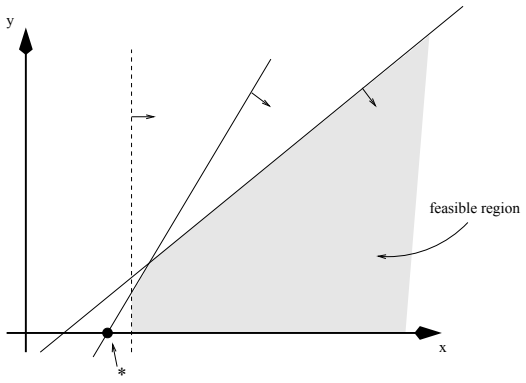

Figure 2: A cutting plane

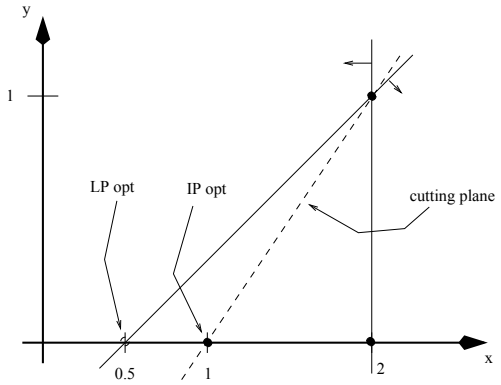

Figure 3: Cutting planes for an integer program

(More specifically, for the dashed one.) By ignoring the dashed constraint, the feasible region has become bigger, and we have found a solution outside of this region. Now, by adding back in the dashed constraint to our linear program, we "cut off" the current solution, and so when reoptimizing, we must get a new optimal solution. Since in 3 dimensions, constraints correspond to planes, or in general, correspond to so-called hyperplanes, this gives rise to the term "cutting plane" approach.

The cutting plane approach can also be used to solve integer programs. In Figure 3, the lines give the feasible region for the linear programming relaxation, and the circles indicate the true feasible solutions to this integer program.

Once again, an optimal solution to the linear relaxation is not feasible; that is, it is not integer. However, if we add the linear constraint $x - y \geq 1$ (which every feasible integer solution satisfies) to the original linear relaxation, and solve this linear program, then the optimal solution to the new linear program is an integer solution, and hence we can conclude that it is optimal. The difficulty in making this approach work is in finding cutting planes that don't cut off any feasible integer solutions, and yet help cut off enough of the current linear programming feasible region to ensure that the process will end up with an integer optimum quickly.