SIAM REVIEW
Vol. 45, No. 1, pp. 116–123

Teaching Integer Programming Formulations Using the Traveling Salesman Problem*

Gábor Pataki†

Abstract. We designed a simple computational exercise to compare weak and strong integer programming formulations of the traveling salesman problem. Using commercial IP software, and a short (60 line long) MATLAB code, students can optimally solve instances with up to 70 cities in a few minutes by adding cuts from the stronger formulation to the weaker, but simpler one.

Key words. integer programming, traveling salesman problem, subtour elimination constraints, cutting planes

AMS subject classifications. 90C10, 90C11, 90C27, 90C57, 90-01, 97D40

PII. S0036144502368551

1. Introduction.

1.1. Integer Programs and Their Formulations.

An integer program (IP) is an optimization problem of the form

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & x \in X, \end{aligned}$$

where $X = \mathbb{Z}^n \cap \{ x \in \mathbb{R}^n \mid Ax \leq b \}$, with some matrix A, vector b, and the symbols $\mathbb{Z}^n$ and $\mathbb{R}^n$ denoting the set of integer and real n-vectors, respectively. The polyhedron $\{ x \in \mathbb{R}^n \mid Ax \leq b \}$ is a *formulation* of the set X. An IP has many formulations; for instance, in Figure 1 the solid and dashed lines enclose the same integer points; i.e., the corresponding polyhedra are both formulations of the same set.

Given two such polyhedra, P and Q (i.e., $P \cap \mathbb{Z}^n = Q \cap \mathbb{Z}^n$), we say that P is a *better*, or *stronger*, formulation, if $P \subsetneq Q$. (The definitions here are from [9].)

One of the most important skills that a practitioner of integer programming must acquire is that of designing a strong formulation for a particular problem. The main component of all commercial IP solvers is a branch-and-bound algorithm that uses the linear programming (LP) relaxation of an IP (i.e., the problem obtained from an IP by discarding the integrality constraints). Hence, a stronger formulation usually can be solved with fewer branch-and-bound nodes; if the tighter LPs are not "much" more time-consuming, this translates into a smaller overall solution time. Even if the

*Received by the editors May 2, 2001; accepted for publication (in revised form) May 28, 2002; published electronically February 3, 2003. Part of this work was done while the author was affiliated with the Department of IE/OR at Columbia University, and was supported by NSF grant DMS 95-27124.

http://www.siam.org/journals/sirev/45-1/36855.html

†Department of Operations Research, University of North Carolina at Chapel Hill, 212 Smith Building, Chapel Hill, NC 27599-3180 (gabor@unc.edu).

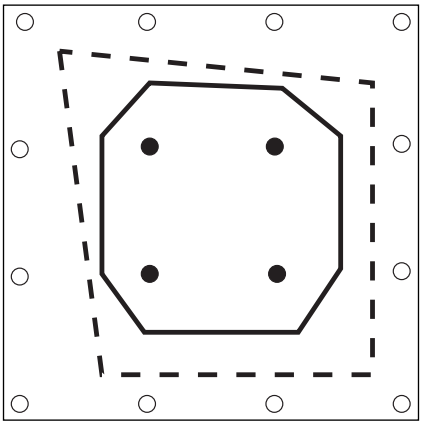

Fig. 1 *Two formulations of the same set.*

problem cannot be solved to optimality within the available time, a strong formulation provides a good bound on the optimal value of the problem. Hence it can also serve as a counterpoint to an effective heuristic, by proving that a solution provided by the latter is close enough to being optimal.

1.2. Exercises to Compare Formulations in Practice. A classroom lecture on comparing weak and strong formulations is best accompanied by an assignment asking students to do a computational comparison. Such a comparison can simply consist of feeding a strong and a weak formulation of the same problem to an IP solver and comparing the number of branch-and-bound nodes and times required to solve them to optimality. A problem is well suited for the purpose of a comparison if

(1) it is relevant in practice, interesting, and easy to understand;
(2) the advantages of the strong formulation are not immediately apparent, for some of the following reasons:
    - the strong formulation uses many more variables and/or constraints;
    - the weak one can accommodate more versatile objective functions;
(3) the weak and strong formulations are easy to generate.

Several such problems are (see, e.g., [9])

- facility location problems (with their weak, i.e., the aggregated, and the strong, i.e., the disaggregated formulation);
- knapsack problems (their usual formulation can easily be strengthened by cover inequalities);
- lot-sizing problems (their usual formulation can be strengthened by various inequalities).

For many problems, however, the advances in IP software disguise the advantage of providing a stronger formulation to solvers. Most IP solvers now incorporate automatic reformulation techniques that can substantially strengthen a weak formulation. Such techniques include disaggregation (i.e., replacing the constraint $\sum_{i=1}^{m} x_i \leq my$ on the 0–1 variables x_i and y with the inequalities $x_i \leq y\,(i = 1, \ldots, m)$), generating many of the inequalities that one would add to the knapsack, and lot-sizing problems (e.g., covers, and flow-covers). Hence, in the case of the models listed above,

- frequently there is no significant difference in the solution times, when feeding a weak or a strong formulation to the solver;

- sometimes the solution times are even better for the weaker formulation. The reason is that IP solvers strengthen the weak formulation quite intelligently. For instance, when disaggregating, they only add those $x_i \leq y$ inequalities, which are violated by the current LP solution, thus not enlarging the LP relaxation unnecessarily;
- or, the strong one must be quite sophisticated and hence not too useful at the beginning of a course, when the importance of strong formulations should already "sink in." Such an example is strengthening the lot-sizing problem with path-inequalities ([9, p. 223]).

Of course, one could always ask the students to turn off the preprocessor, or the cut-generator of the solver; this would hardly make a convincing exercise, though.

A problem that well fits criteria (1) and (2) is the traveling salesman problem (TSP): given n cities, and pairwise travel costs between them, find the shortest tour, i.e., a directed cycle containing all cities. (In fact, students invariably find the TSP more fascinating than the somewhat mundane knapsack, location, or lot-sizing problems.) The formulation that serves as a benchmark for all others is the subtour formulation; there are also many weaker ones. Three articles that survey and analytically compare some of them are [1], [5], and [8]. The weaker formulations all use extra variables; i.e., the set of tours X is written as $X = \mathbb{Z}^n \cap P$, where $P = \{\, x \,|\, Bx + Du \leq f \text{ for some } u \in \mathbb{R}^m \,\}$. Now P is the *projection* of the polyhedron $\{\, (x,u) \,|\, Bx + Du \leq f \,\}$, hence it is also a polyhedron. Thus we can write $P = \{\, x \,|\, Gx \leq h \,\}$ for some matrix G and vector h. The latter representation, however, may contain many more and/or more complicated inequalities than the first.

Designing an experiment that allows students to do a computational comparison is not quite obvious though. This note gives a simple guide to comparing the subtour formulation to a weak one due to Miller, Tucker, and Zemlin [6] on relatively small, but far from trivial instances. It utilizes a primitive "cutting plane algorithm" of about 60 lines of MATLAB code that, used with a commercial IP solver, allows students to do the comparison and solve real world instances with up to 70 nodes to optimality.

2. The Formulations.

2.1. Problem Statement and the Node Constraints.

In the complete directed graph $D = (N, A)$, with arc-costs c_{ij}, we seek the tour (a directed cycle that contains all n cities) of minimal length. Define the variables

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

The feasible solutions of the integer program (in fact, there is always an integer optimal solution to the LP relaxation)

$$\begin{array}{lll} \min & \sum_{i,j} c_{ij} x_{ij} & \\ \text{s.t.} & \sum_i x_{ij} = 1 & \forall i, \\ & \sum_j x_{ji} = 1 & \forall i, \\ & 0 \leq x_{ij} \leq 1, & x_{ij} \text{ integer,} \end{array} \tag{2.1}$$

may contain several directed cycles, called subtours. The constraints of (2.1) are called the *assignment constraints*.

2.2. The MTZ Formulation. To exclude subtours, one can use extra variables u_i $(i = 1, \ldots, n)$, and the constraints

$$\begin{aligned} & u_1 = 1, \\ & 2 \le u_i \le n \quad \forall i \ne 1, \\ & u_i - u_j + 1 \le (n-1)(1 - x_{ij}) \quad \forall i \ne 1, \forall j \ne 1. \end{aligned} \tag{2.2}$$

We call the last inequality in (2.2) an arc-constraint. The formulation consisting of (2.1) and (2.2) is called the Miller–Tucker–Zemlin (MTZ) formulation of the TSP; see [6]. It indeed excludes subtours, as (1) the arc-constraint for (i, j) forces $u_j \ge u_i + 1$, when $x_{ij} = 1$; (2) if a feasible solution of (2.1)–(2.2) contained more than one subtour, then at least one of these would not contain node 1, and along this subtour the u_i values would have to increase to infinity. This argument, with the bounds on the u_i variables, also implies that the only feasible value of u_i is the position of node i in the tour. The advantages of the MTZ formulation are

- its small size (we need only n extra variables and roughly $n^2/2$ extra constraints),
- if it is preferable to visit, say, city i early in the tour, one can easily model this by adding a term $-\alpha u_i$ with some $\alpha > 0$ to the objective.

2.3. The Subtour Formulation. The other way to exclude subtours is by adding to (2.1) the family of subtour (or subtour elimination) constraints

$$\sum_{i \in S, j \in S} x_{ij} \le |S| - 1 \ \ (S \subsetneq V, |S| > 1), \tag{2.3}$$

to obtain the subtour formulation of the TSP consisting of (2.1) and (2.3). (As the subtour inequality for $N \setminus S$ is a linear combination of the inequality for S and of the assignment constraints, it is enough to use the subtour inequalities with S having size at most $n/2$.) It does not have the advantages of the MTZ formulation. The disadvantage of its exponential size is mitigated by the fact that not all subtour inequalities must be put into the formulation from the start. They can be generated as needed by a *separation algorithm*: one can start with the formulation (2.1), then generate subtour inequalities that are violated by the current LP solution. The separation algorithm for subtour constraints is based on network flow techniques; for further details, see [4].

3. The Comparison.

3.1. The Strength of the Two Formulations. Solving a reasonably large (with at least, say, 50 cities) problem to optimality is only possible using the subtour formulation; at least, we are not aware of any published computational studies that use the pure MTZ formulation. There is an analytical explanation of this fact: the LP relaxation of the MTZ formulation is much weaker. Since it contains the extra u_i variables, for the comparison we need to eliminate them by taking appropriate linear combinations of the inequalities in (2.1)–(2.2). One way of doing this is by summing the arc-inequalities along a directed cycle. If the arc set of the cycle is denoted by C, then the result is

$$\sum_{(i,j) \in C} x_{ij} \le \left(1 - \frac{1}{n-1}\right) |C|. \tag{3.1}$$

On the other hand, if we denote the node set of the cycle by $N(C)$, then the subtour

formulation contains the obviously stronger inequality

$$\sum_{i,j \in N(C)} x_{ij} \leq |C| - 1. \tag{3.2}$$

The intuitive fact that adding the arc-inequalities along a cycle is the only "essential" way to eliminate the u_i variables can be made precise. Balas [1] and Padberg and Sung [8] showed that all nondominated inequalities in the projection of the MTZ formulation onto the space of the x variables are of the form (3.1).

3.2. How to Combine Them? An efficient implementation of a separation routine for subtour inequalities is beyond the ability of most practitioners and undergraduate/MS students. A collection of excellent public-domain routines for the TSP, called Concorde, is available, with such routines among them; see [3]. However, most problems encountered in practice are not pure TSPs, only "TSP-like." One example is the vehicle routing problem, in which there are k "salesmen" (i.e., trucks), and each of the cities must be visited by one to make a given amount of delivery, while obeying capacity restrictions. For TSP-like problems the TSP *formulations* are usually not hard to generalize, but the TSP *separation routines* cannot be used.

One can combine the MTZ and subtour formulations to obtain the ease of use of the first and some of the strength of the second. Violated subtour inequalities are easy to identify if a solution satisfies the assignment constraints and is 0–1: one can take S to be the union of several subtours. Such an approach is likely to work for TSP-like problems as well. For small problems, one can identify such subtour inequalities even by inspection: see, for instance, the article [7] for such an approach to a variant of the TSP arising in aircraft mission planning. We used the following "cutting plane algorithm" to solve several TSP instances to optimality, with **maxrounds** an integer between 0 and 2, depending on how much we wanted to strengthen the MTZ formulation.

1. Let the IP formulation consist of the assignment constraints only, and $k = 1$.
2. **while** $k \leq$ **maxrounds**
   - 2a. Solve the IP over the current formulation. Assume that the optimal solution consists of r subtours $S_1, \ldots, S_r$.
   - 2b. If $r = 1$, STOP; the solution is optimal to the TSP. Otherwise, add to the formulation at most 1000 subtour constraints, in which S is the union of several S_i sets and $|S| \leq \lfloor n/2 \rfloor$. Set $k = k + 1$.

   end while
3. Add the arc constraints to the formulation, and solve the IP to optimality.

We used a simple implementation of step 2b: we output the (i, j) pairs corresponding to the nonzero x_{ij} values into a file, then ran a MATLAB routine to generate the required constraints. These were then pasted into the formulation. For example, suppose that the number of cities is 10, and the arcs corresponding to the variables at 1 are as shown in Figure 2. The nontrivial parts (i.e., excluding input, output, and obvious initializations) of the MATLAB code are given in Figure 3. The code works as follows:

- The first part constructs the list of subtours from the list of arcs. In the example, it will create the list $\{1, 4, 6\}, \{2, 3, 8\}, \{5, 7\}, \{9, 10\}$.
- The second part constructs all appropriate subsets of nodes in the variable `S`, and the index-pairs corresponding to arcs within `S`, in the variable `arcs_in_S`.

The code works with all IP solvers that have a "reasonable" user interface: all we need to do is output the index-pairs corresponding to the variables at 1 and paste

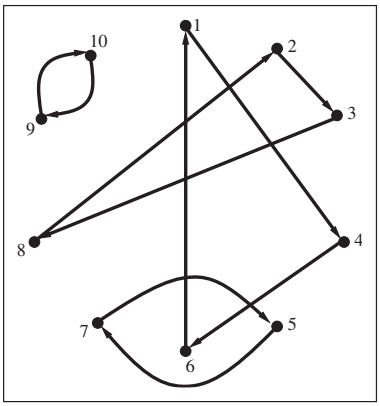

Fig. 2 *A solution with four subtours.*

```
%
% arcs is a vector of the index-pairs of the nonzero arcs
%
nsubtours = 0;
while length(arcs)>0,

    nsubtours=nsubtours+1; subtours(nsubtours,1) = arcs(1,1);
    j = 1;
    next = 1;

    while 1,

       j = j+1; subtours(nsubtours,j) = arcs(next,2);
       arcs(next,:)  = [];
       next = find(arcs(:, 1)==subtours(nsubtours,j));

       if arcs(next,2) == subtours(nsubtours,1),
          arcs(next,:)  = [];
          break;
       end %if

    end %while

end %while

b = zeros(1,nsubtours); % Indicator of which subtours we choose
for i=1:power(2,nsubtours)-1,

    for j=1:nsubtours, b(j) = bitget(i,j);
    end
    S = nonzeros( subtours(logical(b), :)  );

    if (length(S) <= n/2 ),
       arcs_in_S = [nchoosek(S, 2); fliplr((nchoosek(S, 2))) ]';
       %
       % Write the subtour constraint with arcs_in_S
       %
    end % if

end % for
```

Fig. 3 *The MATLAB cut-generator.*

Table 1 *Problem data.*

Problem name	Nodes	Optimal value
br17	17	39
ftv33	34	1286
ftv55	56	1608
ftv70	70	38673
bays29	29	2020
berlin52	52	7542

Table 2 *Empirical comparison.*

Problem	Zero rounds		One round		Two rounds	
name	Time	B&B nodes	Time	B&B nodes	Time	B&B nodes
br17	5980	2315173	1	10	—	—
ftv33	300	24075	91	1382	18	64
ftv55	***	***	10932	88358	555	1687
ft70	***	***	1654	2257	35	11
bays29	***	***	214	4182	9	27
berlin52	***	***	***	***	8743	49659

the file containing the generated constraints into the solver's input. After every paste operation the IP must be resolved from scratch. Most of the time is taken up by solving the final formulation, i.e., the IP with the subtour- and arc-constraints. The IPs encountered during the cutting phase solve very quickly, and the running time of the MATLAB program is negligible.

We tested this method on six relatively small but nontrivial TSP instances from the TSPLIB library [10]. The problem parameters are given in Table 1. The last two are symmetric instances, in which $c_{ij} = c_{ji}$ for all i, j pairs. (Symmetric TSPs are usually formulated on an undirected graph using edge-variables; however, with these variables the MTZ formulation has no compact size variant; see [8].)

In Table 2 we give the running times in seconds and the number of branch-and-bound nodes needed to solve the MTZ formulation to optimality, depending on how much it was strengthened with subtour inequalities. (We do not report the IP solution times during the cutting phase, nor the time for the constraint generation.) We used the CPLEX 6.5 IP solver run from within the AMPL modeling language, with default branching strategy and a memory limit of 150 MB, on a 337 MHz machine running Sun Solaris. We note that CPLEX was not able to generate any cuts, and that none of the successful runs used more than 100 MBs of memory.

The symmetric instances tend to be harder to solve, as the IP solutions during the cutting phase contain many small (mostly size 2) subtours. Therefore, there are more violated subtour constraints, and due to memory restrictions we cannot add all of them to the formulation (we add at most 1000). For the symmetric instances, we can consider the assignment constraints with all the inequalities $x_{ij} + x_{ji} \leq 1$ as the formulation with 0 rounds (i.e., we can add all subtour constraints where S is of size 2 at the beginning). This way they can be solved with one fewer round of cuts, but the solution times will usually be greater than for asymmetric instances of comparable size.

Table 3 *Subtours and IP values.*

Problem	zero rounds		one round		two rounds	
name	# subtours	IP val.	# subtours	IP val.	# subtours	IP val.
br17	8	0	1	39	—	—
ftv33	8	1185	7	1252	4	1267
ftv55	9	1435	5	1566	10	1582
ft70	10	37978	4	38649	3	38662
bays29	14	1764	5	1950	3	2017
berlin52	23	6287	14	7056	3	7449

3.3. Discussion. In a class at any level (undergrad./M.S./Ph.D.), developing the "cutting plane method" can be assigned as homework or part of a project. In a Ph.D. class, the method of computing the projection following [1] or [8] can also be covered.

There are several lessons to be learned from these experiments:

- the foremost one is the connection between the strength of a formulation and the required solution time. The dependence of the strength on the number of rounds is also shown in Table 3, with the number of subtours and the value of the IP without the arc-constraints (with them, the IP values are obviously the ones in Table 1). The strength of the formulation almost exclusively comes from the subtour constraints: the values of the LP optima of a formulation with and without the arc-constraints are very close.
- a larger problem is not necessarily harder: a small instance, as br17 can be surprisingly hard using the weak formulation; with 1 round of subtour constraints added, the optimal solution is a tour, even without the arc-constraints. (If one wants to assign only one problem in a student project, then probably this should be the one!) On the other hand, instances with randomly generated arc-costs are easy, even with the weak formulation, and for a hundred nodes.

Acknowledgments. Thanks are due to David Johnson, Mark Hartmann, Scott Provan, and referee 2 for their comments, and to Olga Raskina for her help in implementing the cutting plane method.

REFERENCES

[1] E. Balas, *Projection in Combinatorial Optimization*, in Computational Combinatorial Optimization, Lecture Notes in Comput. Sci. 2241, M. Jünger and D. Naddef, eds., Springer-Verlag, Berlin, 2001.
[2] http://www.caam.rice.edu/~bico/, home page of Bill Cook at Rice University.
[3] http://www.keck.caam.rice.edu/concorde.html.
[4] E. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds., *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester, UK, 1985
[5] A. Langevin, F. Soumis, and J. Desrosiers, *Classification of travelling salesman formulations*, Oper. Res. Lett., 9 (1990), pp. 127–132.
[6] C. E. Miller, A. W. Tucker, and R. A. Zemlin, *Integer programming formulations and traveling salesman problems*, J. ACM, 7 (1960), pp. 326–329.
[7] D. M. Panton and A. W. Elbers, *Mission planning for synthetic aperture radar surveillance*, Interfaces, 29 (1999), pp. 73–88.
[8] M. Padberg, and T.-Y. Sung, *An analytical comparison of different formulations of the travelling salesman problem*, Math. Programming, 52 (1991), pp. 315–357.
[9] L. A. Wolsey, *Integer Programming*, Wiley, Chichester, UK, 1999.
[10] http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/.