Using Linear Programming to Solve the Travelling Salesman Problem*

Computer Science
308-566A
Methods in Operations Research
Patrice Pominville
9745398

December 3, 1999

1 Introduction

The *Traveling Salesman Problem* or TSP, can be stated as follows:

> Given a finite number of "cities" along with the cost of travel between them, find the cheapest way of visiting all cities and returning to your starting point.

The cost function is symmetric so that the cost of travelling from city X to city Y is the same as that from city Y to city X. Any path starting at city Z, visiting all cities and returning to this same city is referred to as a *tour*. Hence the TSP consists of finding a tour (not necessarily unique) that minimizes the total traveling cost.

Although simply stated, this problem is known to be *NP-complete* and hence no efficient algorithm has been discovered to solve it. In particular, a brute force algorithm would have to weed through a search space of (n-1)!

*See `http://www.cs.mcgill.ca/~patrice/cs566/tsp_lp.html` for a complementary web site to this paper.

tours for a n city instance, which makes this approach completely infeasible all but for the smallest values of n.

To effectively tackle combinatorial problems of this nature, and NP-complete problems in general, many heuristic methods ranging from neural nets to genetic algorithms have been developed; although these heuristics often yield good solutions to the problem at hand, there is usually no way of verifying the optimality of the former. To date, the only known way of obtaining provably optimal solutions to large instances of the TSP problem has been through *linear programming* techniques.

This paper will focus on how linear programming was used to solve the first non-trivial TSP instance, as well as the importance and influence of the techniques pioneered in doing so. Using the exact same methodology, the results of solving this instance with a modern LP solver is given in the appendix.

2 Solving a TSP instance using Linear Programming

2.1 Historical Perspective

Danzig, Fulkerson and Johnson [1] were the first to apply linear programming techniques in an attempt to solve large instances of the TSP. In doing so they succeeded in 1954 in solving a 48 city instance of the problem in which each city corresponds to a city chosen from each of the 48 states of the continental United States; the travel costs between cities were taken to be the actual road distances between them as found in a road atlas circa 1954. Solving a problem of this magnitude was unheard of in those days and not surprisingly in doing so they pioneered many key techniques used in the field of *Operations Research* today.

2.2 Casting the TSP into a Linear Program

Mathematically a n city instance of the TSP can be stated as follows:

Let c be a vector of length n(n-1)/2 whose components are the travel costs for the city pairings.

Denote by $\mathcal{S}$ the set of all incidence vectors corresponding to valid tours. Such vectors will have value 1 for all entries corresponding to city pairings that are links in the tour, and 0 otherwise.

The TSP can now be formulated as:

$$\text{minimize } cx \text{ s.t. } x \in \mathcal{S} \tag{1}$$

The above is just a reformulation of the TSP in mathematical terms and makes it no more tractable. Faced with this problem they could not solve, Danzig, Fulkerson and Johnson sought to transform it into a similar problem they could solve, namely:

$$\text{minimize } cx \text{ s.t. } Ax \leq b \tag{2}$$

Under this form, the problem could then be solved by means of linear programming's *Simplex Method*. Unfortunately the TSP cannot be easily be characterized completely in terms of linear inequalities. In fact the number of such inequalities is exponential in the size of a TSP instance.

The key idea introduced by Danzig, Fulkerson and Johnson was not to try and impose all the constraints necessary in order to define $\mathcal{S}$ precisely, but only impose just enough constraints as to obtain a solution to (2) that is a valid tour. In other words their idea is to create a linear program (2) that is a relaxation of (1), in the sense that every solution of (1) is a solution of (2); in this context an optimal solution of (2) which happens to be also a solution of (1) will necessarily be an optimal solution of (1).

Thus the algorithm they proposed to solve the TSP was iterative in nature and took the following form:

let $x =$ some vector
let $Ax \leq b$ be an empty set of constraints
while(x not in $\mathcal{S}$)
  Add a new constraint to the system in (2) not satisfied by x.
  Use the simplex method to solve (2) yielding a new solution x.
end while

2.3 TSP cuts

As observed by Danzig, Fulkerson and Johnson, even though a TSP instance can only be fully specified by an astronomical number of constraints, in practice usually only a trivial number of well chosen constraints are necessary to obtain a valid tour. Nowadays, such constraints are referred to as *cuts* or *cutting planes*. Over the years this new paradigm of using only as many constraints as necessary to solve a problem has exerted a tremendous influence on the approach taken to solve computationally intensive problems, and has come to represents a key problem solving technique in many operations research fields, notably *Integer Programming*.

In order to solve their instance of the TSP, Danzig, Fulkerson and Johnson introduced a total of 67 constraints that were applied to a 42 city subset of their problem instance[1].

The constraints applied fall into four categories:

Let x_{ij} represent an edge between cities i and j, where $x_{ij} \equiv x_{ji}$. Initially they imposed the following set of constraints:

$$\sum_j x_{ij} = 2 \quad \forall i \tag{C1}$$

These reflect the fact that in a tour, each city will be linked to exactly 2 other cities. The above template specifies 42 constraints, one for each of the cities in the problem instance.

Solving system (2) imposing only constraints (C1) yields a solution which is not a valid tour (see appendix A.1). In particular many of the variables in the solution take on values greater than 1. To bound these values, 16 constraints of the form:

$$x_{ij} \leq 1 \tag{C2}$$

were introduced (it was already assumed that $x_{ij} \geq 0 \quad \forall i \forall j$).

Solving (2) using the set of constraints (C1) and (C2) gives a solution where most of the variables take on values between 0 and 1. However some of these variables take on fractional values; concurrently, the solution obtained contains many subtours, that is tours with less than n cities (see appendix A.2).

The third family of constraints imposed address the subtour issue. A particular subtour consisting of a strict subset T of the n cities, can be eliminated by introducing the following constraint:

[1] As it will be explained, the optimal solution to this subproblem yields an optimal solution to the complete problem.

$$\sum_{T} x_{ij} \leq \quad \mid T \mid -1 \quad \text{where } i,j \in T \tag{C3a}$$

which reflects the fact that in any tour, given any strict subset T of the n cities, there are $\mid T \mid -1$ links of the tour in T.

Symmetrically,

$$\sum_{\bar{T}} x_{ij} \leq \quad n- \mid T \mid -1 \quad \text{where } i,j \in \bar{T} \tag{C3b}$$

And from (C3a) and (C3a') we get:

$$\sum_{T\bar{T}} x_{ij} \geq 2 \quad \text{where } i \in T, j \in \bar{T} \tag{C3c}$$

This last type of constraint reflects the fact that in a tour, given any strict subset T of the n cities, we find 2 links going from cities in T to cities not in T.

To break the subtours found in the previous solutions they imposed 7 of these subtour constraints where 5 of them are of the form (C3b) and the two others are of the form (C3a) (see appendix A.3).

Having imposed all the previous constraints and not obtained a solution that is a valid tour, Danzig, Fulkerson and Johnson imposed 2 extra constraints arising from ad-hoc combinatorial arguments, the details of which will not be given here. These two extra constraints, that will be referred to as (C4), enabled them to prove the optimality of a tour they had conjectured optimal: solving (2) with all of the preceding constraints yields an optimal solution of 698.6 (see appendix A.4). Because all entries in the cost vector c are integral this result imposes a lower bound of 699 on any tour; having manually identified a tour of length 699, this result proves it's optimality.

Hence this procedure directed them towards a provably optimal solution to the 42 city subproblem of the original. Now because the optimal tour (i.e.. the roads on the map) found for this 42 city subproblem passed through the 6 omitted cities, this tour was in fact optimal for the 48 city instance.

3 Current State of the TSP

Danzig, Fulkerson and Johnson's method of obtaining additional constraints or cuts for solving their TSP instance was based on manually inspecting the partial solutions obtained in the iterative process and, then adding relevant constraints from the sets of constraints (C1), (C2), (C3), as well as elaborating ad-hoc custom constraints (i.e (C4)), if need be.

To make their problem solving scheme efficient and viable, a way to automatically generate relevant cuts was needed. This need was addressed by Gomery who in 1958 introduced integer cuts that can be automatically generated from simplex dictionaries to impose integrality constraints. These were shown to guarantee integer solutions for integer variables in mixed linear programs.

Even though at present there are many algorithms to generate cuts for TSP instances and other such problems, these still remain very difficult to solve but for small instances. This can be explained by the fact that it is not sufficient to uniquely generate cuts; these cuts must be efficient in carving out non trivial areas of the solution space of a problem, otherwise an exponential number of cuts might be required. Thus the ongoing research in solving the TSP and other combinatorial optimisation problems, has been in developing techniques and algorithms that generate efficient cuts. These cuts can either be generic in nature as to be applicable to any mixed linear program or of more restricted scope relying on domain specific knowledge such as *comb inequalities*[4] in the case of the TSP.

At present only a few instances of the TSP having more than a few hundred cities have been solved. Notably in 1991 a non-trivial 13509 instance TSP problem was solved by Applegate, Bixby, Chvátal and Cook [3] using the all the latest development in field to generate efficient cuts[2]. Their work is still on going and promises to shed light on the efficiency of some interesting new techniques.

4 Conclusion

The early work of Danzig, Fulkerson and Johnson on the TSP problem introduced new paradigms in solving computationally intensive problems whose impact has been far reaching in the field of Operations Research. For the first time, they made use of the concept of cuts that proved to be an invaluable problem solving technique. Even today more than forty years after the publication of their work, all work in solving the TSP is thoroughly based on variations and refinements of the few simple principles they pioneered and brought forth and for the time being, linear programming remains the only viable method to solve large scale TSP instances.

A Solving Danzig, Fulkerson and Johnson's original 48 city TSP instance using CPLEX

In order to experience for myself the glorious moments and the satisfaction of solving a large scale TSP problem using linear programming, I set out to solve Danzig, Fulkerson and Johnson's quintessential 48 city TSP instance. I used the software package CPLEX as my LP solver. To generate the actual linear program, I built a small C program that emits the LP in CPLEX syntax, generating all constraints and the objective function. The emitted LP instance is parametrized in terms of the travel cost variables; these variables are defined in a separate file . Using the C preprocessor, the symbolic travel costs in the generated LP are replaced by their actual values. This yields the final version of the linear program which can then be fed to the CPLEX engine. The generated CPLEX linear program instance, the C program, the travel costs, along with other material can be accessed at `http://www.mcgill.ca/~patrice/cs566/tsp_lp.html`.

The following results where obtained while trying to minimize:

$$\sum_{i=1,i<j}^{42} c_{ij} x_{ij} \tag{C4}$$

where the variables x_{ij} correspond to the edges in the complete graph of the 42 cities and the c_{ij} are the corresponding edge costs.

A.1 Solution Obtained Using Constraint Set (C1)

```
CPLEX> read out1.lp
Problem 'out1.lp' read.
Read time =    0.02 sec.
CPLEX> opt
Tried aggregator 1 time.
No LP presolve or aggregator reductions.
Presolve time =    0.02 sec.

Iteration log . . .
Iteration:     1    Infeasibility =            6.000000
Iteration:    10    Objective     =          778.000000

Primal - Optimal:  Objective =    5.3200000000e+02
Solution time =    0.03 sec.  Iterations = 63 (9)

CPLEX> di so va -
Variable Name           Solution Value
x_2_1                         2.000000
x_4_3                         2.000000
x_38_5                        2.000000
x_7_6                         2.000000
x_9_8                         2.000000
x_24_10                       1.000000
x_25_10                       1.000000
x_12_11                       2.000000
x_16_13                       1.000000
x_17_13                       1.000000
x_15_14                       2.000000
x_17_16                       1.000000
x_19_18                       2.000000
x_21_20                       2.000000
x_23_22                       2.000000
x_25_24                       1.000000
x_27_26                       2.000000
x_29_28                       2.000000
x_31_30                       2.000000
```

```
x_33_32                    2.000000
x_35_34                    2.000000
x_37_36                    2.000000
x_40_39                    2.000000
x_42_41                    2.000000
All other variables in the range 1-861 are zero.
```

A.2 Solution Obtained Using Constraint Sets (C1) and (C2)

```
CPLEX> read out.lp
Problem 'out.lp' read.
Read time =    0.02 sec.
CPLEX> opt
Tried aggregator 1 time.
LP Presolve eliminated 16 rows and 0 columns.
Reduced LP has 42 rows, 861 columns, and 1722 nonzeros.
Presolve time =    0.01 sec.

Iteration log . . .
Iteration:     1    Infeasibility =            7.000000
Iteration:    14    Objective     =         1409.000000

Primal - Optimal:  Objective =    6.3400000000e+02
Solution time =    0.03 sec.  Iterations = 68 (13)

CPLEX> di so va -
Variable Name           Solution Value
x_2_1                         2.000000
x_4_3                         1.000000
x_8_3                         1.000000
x_5_4                         1.000000
x_6_5                         1.000000
x_7_6                         1.000000
x_9_7                         1.000000
x_9_8                         1.000000
x_12_10                       1.000000
x_25_10                       1.000000
x_12_11                       1.000000
x_23_11                       1.000000
x_14_13                       1.000000
x_17_13                       1.000000
x_15_14                       1.000000
x_16_15                       0.500000
x_18_15                       0.500000
```

```
x_17_16                      1.000000
x_18_16                      0.500000
x_19_18                      1.000000
x_20_19                      1.000000
x_21_20                      1.000000
x_22_21                      1.000000
x_23_22                      1.000000
x_25_24                      1.000000
x_27_24                      1.000000
x_27_26                      1.000000
x_28_26                      1.000000
x_29_28                      1.000000
x_30_29                      1.000000
x_31_30                      1.000000
x_32_31                      1.000000
x_33_32                      1.000000
x_34_33                      1.000000
x_35_34                      1.000000
x_36_35                      1.000000
x_37_36                      1.000000
x_38_37                      1.000000
x_39_38                      0.500000
x_40_38                      0.500000
x_40_39                      1.500000
x_42_41                      2.000000
All other variables in the range 1-861 are zero.
```

A.3 Solution Obtained Using Constraint Sets (C1), (C2) and (C3)

```
CPLEX> read out2.lp
Problem 'out2.lp' read.
Read time =    0.04 sec.
CPLEX> opt
Tried aggregator 1 time.
LP Presolve eliminated 16 rows and 0 columns.
Reduced LP has 49 rows, 861 columns, and 3290 nonzeros.
Presolve time =    0.01 sec.

Iteration log . . .
Iteration:     1    Infeasibility =           18.999999
Iteration:    14    Objective     =         1376.000000
Iteration:    83    Objective     =          721.000000

Primal - Optimal:  Objective =    6.9700000000e+02
Solution time =    0.05 sec.  Iterations = 98 (13)

CPLEX> di so va -
Variable Name           Solution Value
x_2_1                         1.000000
x_42_1                        1.000000
x_3_2                         1.000000
x_4_3                         1.000000
x_5_4                         1.000000
x_6_5                         1.000000
x_7_6                         1.000000
x_8_7                         1.000000
x_9_8                         1.000000
x_10_9                        1.000000
x_11_10                       1.000000
x_12_11                       1.000000
x_13_12                       1.000000
x_14_13                       1.000000
x_15_14                       1.000000
x_18_15                       1.000000
```

```
x_17_16                          2.000000
x_19_18                          1.000000
x_20_19                          1.000000
x_21_20                          1.000000
x_22_21                          1.000000
x_23_22                          1.000000
x_24_23                          1.000000
x_25_24                          1.000000
x_26_25                          1.000000
x_27_26                          1.000000
x_28_27                          1.000000
x_29_28                          1.000000
x_30_29                          1.000000
x_31_30                          1.000000
x_32_31                          1.000000
x_33_32                          1.000000
x_34_33                          1.000000
x_35_34                          1.000000
x_36_35                          1.000000
x_37_36                          1.000000
x_38_37                          1.000000
x_39_38                          1.000000
x_40_39                          1.000000
x_41_40                          1.000000
x_42_41                          1.000000
All other variables in the range 1-861 are zero.
```

A.4 Solution Obtained Using Constraint Sets (C1), (C2), (C3) and (C4)

```
CPLEX> read out.lp
Problem 'out.lp' read.
Read time =    0.05 sec.
CPLEX> opt
Tried aggregator 1 time.
LP Presolve eliminated 16 rows and 0 columns.
Reduced LP has 51 rows, 861 columns, and 3751 nonzeros.
Presolve time =    0.01 sec.
Iteration log . . .
Iteration:     1    Infeasibility =           14.999999
Iteration:    26    Objective     =         1258.000000
Iteration:    94    Objective     =          738.500000

Primal - Optimal:  Objective =    6.9860000000e+02
Solution time =    0.04 sec.  Iterations = 118 (25)

CPLEX> di so va -
Variable Name           Solution Value
x_2_1                         1.000000
x_42_1                        1.000000
x_3_2                         1.000000
x_4_3                         1.000000
x_5_4                         1.000000
x_6_5                         1.000000
x_7_6                         1.000000
x_8_7                         1.000000
x_9_8                         1.000000
x_10_9                        0.800000
x_25_9                        0.200000
x_12_10                       0.400000
x_25_10                       0.800000
x_12_11                       1.000000
x_23_11                       0.400000
x_24_11                       0.600000
x_13_12                       0.600000
```

```
x_14_13                          1.000000
x_17_13                          0.400000
x_15_14                          1.000000
x_16_15                          1.000000
x_17_16                          0.600000
x_18_16                          0.400000
x_18_17                          0.600000
x_23_17                          0.400000
x_19_18                          1.000000
x_20_19                          1.000000
x_21_20                          1.000000
x_22_21                          0.600000
x_28_21                          0.400000
x_23_22                          1.000000
x_28_22                          0.400000
x_24_23                          0.200000
x_25_24                          0.200000
x_27_24                          1.000000
x_26_25                          0.800000
x_27_26                          1.000000
x_28_26                          0.200000
x_29_28                          1.000000
x_30_29                          1.000000
x_31_30                          1.000000
x_32_31                          1.000000
x_33_32                          1.000000
x_34_33                          1.000000
x_35_34                          1.000000
x_36_35                          1.000000
x_37_36                          1.000000
x_38_37                          1.000000
x_39_38                          1.000000
x_40_39                          1.000000
x_41_40                          1.000000
x_42_41                          1.000000
All other variables in the range 1-861 are zero.
```

References

[1] G. Danzig, R. Fulkerson and S. Johnson, Solution of a Large Scale Travelling Salesman Problem, *Operations Research* 2, 393-410, 1954

[2] D. Applegate, R. Bixby, V. Chvátal and W. Cook, On the Solution of the Travelling Salesman Problem, *Documenta Mathematica* Extra Volume ICM 3, 645-656, 1998

[3] Record Traveling Salesman Solution, *http://www.crpc.rice.edu/CRPC/newsArchive/techweb_6_29_98.htm*, 1998

[4] M. Trick, Cuts for Special Structure, *http://mat.gsia.cmu.edu/orclass/integer/node16.htm*, 1998