**An Integer Programming Formulation to Solve “Pet Detective”**

Duane Mattern, mattern@SampledSystems.com

Sampled Systems, LLC

**Abstract**

I needed to get up to speed on the state-of-the-art in optimization and software solvers, so I used an easier problem as an aid to my learning process. The problem I chose to solve is the Pet Detective [1] game on Lumosity. I’ve played the game several times, but I got to a level that became difficult to solve quickly. I thought I could come up with a solver, (*not automated*), and learn about optimization tools in the process. The approach I took was as an integer programming problem, using previously published formulations of the capacitive pickup-and-delivery, vehicle routing problem [2]. The purpose of this article is to share this experience as an educational tool. All files are available on GitHub [16]. An example and formulations coded in AMPL and PuLP as included in the Appendix.

**Keyword words and abbreviations**

IP Integer Programming LP Linear Programming

PDP Pickup and Delivery Problem VRP Vehicle Routing Problem

CVRP Capacitated Vehicle Routing Problem TSP Traveling Salesman Problem

**Tools**

There are a ton of great tools out there for solving linear programming problems. I cycled through a number of them including LINGO [3], AMPL IDE, and RStudio [4], before settling in on SolverStudio [5,6] which is an add-in for Excel. Microsoft’s Excel has its own built-in solver, but SolverStudio surpass that capability. SolverStudio supports several different modeling languages, (PuLP, COIN-OR, AMPL, GMPL, GAMS, Gurobi, CMPL, SimPy, and Python programming). SolverStudio also supports multiple solvers built into the various packages, including the web-based NEOS platform [7]. In the following, I use AMPL[8] and PuLP [9,10] as modeling languages. SolverStudio is a fantastic educational tool and greatly facilitated and accelerated my learning process.

**The Pet Detective Problem**

|  |  |
| --- | --- |
|  | Pet Detective is a shortest route determination game. A vehicle is driven over a grid-like map to pick-up pets at various locations and to deliver those pets to their specific “homes” on the map. The vehicle is constrained to a maximum carrying capacity of four pets at a time, (which isn’t relevant to the 3-pet problem shown on the left). The goal of the game is to figure out a route to deliver all of the pets with the shortest total distance. There is typically more than one optimal solution. The desired route length is provided as a goal to achieve, shown as “7” in the easy example game grid on the left. |

**Formulation of the Optimization Problem**

A “formulation” describes how the problem is modeled in order to present it to the “solver”. I followed the formulation in Lu & Dessouky’s 2002 paper [11], modifying it for a single vehicle and some other tweaks specific to this problem. The problem is viewed as a network with a set of nodes and arcs between the nodes. This approach works but it leaves out information as discussed later in the conclusions. The formulation shown below follows most of the notation used in references [11,12].

A path through all of the nodes, called a “tour”, is considered to be a Hamiltonian cycle [13] where each node in the graph is visited just once. Subtour elimination (MTZ, DFJ) [14] is built-in to the formulation [11]. Multiple “visits: to a single node are avoided by defining “a visit” to a node to only occur only if either a pickup or delivery is made. Passing through a node without a drop-off or pickup is not considered as “a visit”. The distance matrix (cost) is setup such that it only considers the shortest distance between nodes.

|  |  |  |
| --- | --- | --- |
|  | In the example on the left, there are 3 pickups, 3 drop-offs, and one depot for a total of 7 nodes. The route starts at node 7, the depot, and then ends at the last drop-off node. The return to the starting node is added without cost, (distance=0). |  |

There are **n** pets or “customers”. Each customer has a pickup and delivery site. Denode Nplus, **N+r** = { 1, 2, …n} as the set of nodes making up the pickup requests and Nminus, **N-r** = { 1+n, 2+n, …2n} as the set of the delivery nodes (homes). Then the set of all service stops is Nr = union between N+r  and N-r. There is only a single vehicle is this problem and it has a single starting point (the “depot”). We assign the depot to node number 2n+1. So the set of all of the nodes is indicated by the set “**N**”, (with no subscript). In the Pet Detective game, the game ends at the last home and there is no requirement to return to the starting location. In our formulation, the cycle requires a return to the depot, but we only use a single designation for this starting and ending node. The cost associated with the last move back to the depot is handled by assigning it a distance of zero. The vehicle capacity, denoted as **CAP**, is limited to 4 passengers. The problem is formulated as an integer programming problem as follows:

**Decision Variables (solver output)**

The decision variables are binary from the set {0, 1}.

**Objective Function (Cost)**

Minimize ,

where

The cost matrix is required to be symmetric in this formulation. Since the solution is a Hamiltonian cycle, starting and ending at the depot, the costs between all of the drop-off or home nodes and the depot were forced to be zero, symmetrically. Thus the calculated cost matches that of the puzzle. There is no cost to return to the depot.

**Parameters**

In the AMPL and PuLP models:

“Nodes” is used to describe the set of all nodes { N }

“NReduced” is used to describe the union {}.

“Pickup” is used to describe the set {}.

“solve\_result” is a text field to display the status of the solver output.

“Total\_Cost” is the scalar value calculated from the objective function.

**Internal Variable**

With zero initial passengers, the number of passengers in the vehicle when leaving node “j” is:

zout j = Gj +

This value is constrained to the capacity of the vehicle, but only needs to be tested for violation during a passenger pickup, not during dropoff. The variable **zout** is calculated and displayed as part of the solution, but it is calculated from **b** and is not a “decision” variable.

**Constraints ( subject to )**

In the following, I’m using the equation numbering from reference [11], but I skip the equation numbers that are not being used.

**Basic network constraints**

One entry and one exit per node. This will appear in the solution “**x**” matrix as one nonzero entry per row and one nonzero entry per column, as shown in the Appendix.

|  |  |  |
| --- | --- | --- |
|  | Summation of the columns per each row | (1) |
|  | Summation of the rows per each column | (2) |

**Copy constraints** (*get the subscripts right. A/B means contained in set A, but not in the set B.)*

|  |  |
| --- | --- |
| for all arc(i,j) N /{arc(2n+1) and kN / {i} | (3) |
| for all arc(i,j) N /{arc(2n+1) and kN / {i} | (4) |
| xi,j ≤ bi,j for all arc(i,j) N | (5) |

The copy constraints, equations 3&4, are such that if node i is immediately before node j in the tour, (xi,j=1), then these equations force bk,i = bk,j for all k N and k≠i. Equation 5 makes sure that b(i,j) is set if x(i,j) is set.

**Diagonals**: The matrix diagonal terms are all zero for both **x** and **b**. EQ 5 above enforces this for **x**.

|  |  |
| --- | --- |
| bi,i = 0 for all i N | (6) |

**Priors**

In the tour, the drop-off node will never occur before the corresponding pickup node (equation 7) and the pickup node always occurs before the corresponding drop-off node, (equation 8).

|  |  |
| --- | --- |
| b n+i, i = 0 for all i | (7) |
| b i, n+i = 1 for all i | (8) |

**Capacity Constraint**

Since we only have one vehicle in the Pet Detective problem, we skip equation (9) and jump to equation (10) for the vehicle carrying capacity. We’ve already mention the internal variable zout j, which is the equal to the number of passengers/pets in the vehicle upon departing node j. Note the use of the set , since the capacity only needs to be tested when a customer is picked up.

|  |  |
| --- | --- |
| zout j = Gj + ≤ m for all i where m = 4 in Pet Detective. | (10) |

**First/Start (departing the depot)**

This constraint says that the first pickup won’t be from a drop-off node. Note the use of the set . The second equation is redundant with equation 5.

|  |  |
| --- | --- |
| b2n+1,i = 1 Start for all i , last row cells = 1 | (11) |
| x2n+1,n+i = 0 First for all i , |  |

**Last/End (returning to the depot)**

This constraint indicates that the last move, returning back to the depot, will not be from a pickup node. Note the use of the set . The second equation is redundant with equation 5 and is probably is removed during pre-solve.

|  |  |
| --- | --- |
| bi,2n+1 = 1 End for all i last column values = 1 | (14) |
| xi,2n+1 = 0 Last for all i |  |

We differ from reference [11] on equations 11 and 14 because we are only using a single node for both the first and last node. The last row/column of the b-matrix may be unnecessary, since it is pre-known with this formulation.

**Constraints for Speed Improvements**

Reference [11] included a number of constraints to increase the solver speed. I attempted all but one, the Transfer Prior constraint, which was complicated to implement. Independently, I had success with all of them, but when combined, some of the constraints were no longer of value, so I ended up using a subset and commenting out the rest. Only those used are listed below.

**Valid Equalities - Vehicle Return Constraint**

Because Pet Detective only has one vehicle and because I’ chosen to only have one node for both the starting and end location, this constraint was reduced to making the last column in the b-matrix all ones, except for the diagonal term. It would probably be easier to code it directly instead of using the summation below, but it did improve the solution speed.

|  |  |
| --- | --- |
| for all i | (25) |

**Valid Equalities – B Equality Constraint**

This constraint is from the definition of the b-variables and did result in a significant speed improvement.

|  |  |
| --- | --- |
| bi,j + bj,i = 1 for all i | (26) |

**Known Objective**

Pet Detective provides the target objective, so there is no reason to search beyond that. Most solvers will search until they concluded the optimal has been reached, but sometimes this takes time for the solver to come to this conclusion. The following provided a small decrease in the solution time by avoiding search beyond the target objective.

>= KNOWN\_OBJ

**Automating the Process**

My reason for learning SolverStudio was to get up to speed on optimization tools for another project. Whereas I could have generated an image parser to automate the generation of the distance/cost matrix from the Pet Detective grid, I chose not to spend my time on this task. As such, the most tedious part of this “solver” is the manual generation of the distance/cost matrix. Errors in this matrix, particularly those that make the matrix asymmetric, will vastly increase the runtime. Long runtimes are an indication of a user typo error.

**The Solution**

The AMPL NEOS server and the PuLP solver using the Coin-OR Branch and Cut (CBC) solver worked well, but there are a lot of configuration options. I experimented with a few of these with no conclusive result, except for using an initial feasible solution. PuLP does not have a direct way to command an initial feasible solution [16] (i.e. “mipstart”). As an alternative, I got in the habit of commenting out the “solve” command in PuLP and just using SolverStudio+PuLP to generate the CBC model file, which I would then “*import”* into the command-line version of CBC. This allowed me to a) set “*verbose 1*” to get more feedback and to b) use “mipstart” to define an initial feasible condition. As an example, when solving PD11, I used the solution for PD10 as a feasible solution to PD11 (which it is not) and this considerably reduced the solution time. I don’t know why this occurred, but this is something I would like to look into further in the future.

This CVRP formulation of Pet Detective works, but it is not particularly fast. A table of the solution parameters, including speeds for a set of puzzles from 3 to 11 passengers is shown below for both the AMPL formulation using the NEOS server. The larger problems with 10 and 11 pets took in excess of 2 hours to solve. Note that Pet10 was harder to solve than Pet11.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table 1 AMPL Performance Data (CAP=4)** | | | | | | | | |
| **# Pets (n)** | **# of Nodes (2n+1)** | **Time (sec)** | **adjusted binary variables** | **constraints** | | | | |
| **# linear** | **# non-zeros** | **# equality** | **# inequality** | **# non-zero** |
| 3 | 7 | 0.03 | 57 | 255 | 696 | 38 | 217 | 30 |
| 4 | 9 | 0.24 | 112 | 698 | 1978 | 66 | 632 | 56 |
| 5 | 11 | 0.16 | 180 | 1467 | 4120 | 102 | 1365 | 90 |
| 6 | 13 | 1 | 258 | 2673 | 7788 | 146 | 2527 | 132 |
| 7 | 15 | 300 | 357 | 4405 | 12726 | 198 | 4207 | 182 |
| 8 | 17 | 4425 | 472 | 6763 | 19888 | 258 | 6505 | 240 |
| 9 | 19 | 3632 | 603 | 9840 | 29016 | 326 | 9514 | 306 |
| 10 | 21 | 13145 | 750 | 13733 | 40580 | 402 | 13331 | 380 |
| 11 | 23 | 10700 | 913 | 18538 | 54868 | 486 | 18052 | 462 |

A table of the solution parameters for the PuLP formulation using a Dell Precision T5500 workstation with Xeon processors, running the CBC process at “High” priority under Windows 10. I don’t know the reason for different times between SolverStudio and the same model in CBC.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Table 2 PuLP Formulation Solved using CBC (64-bit) on PC/Windows10** | | | | | | |
| **# Pets (n)** | **# of Nodes (2n+1)** | **Time (sec)**  **raw** | **Time (sec) w/ cmdline** | **Enum Nodes** | **Iterations** | **Comment: Execution in SolverStudio and using the command line version of CBC** |
| 3 | 7 | 0.7 | - |  |  | Run in Excel using SolverStudio+PuLP |
| 4 | 9 | 1.1 | - |  |  | “ |
| 5 | 11 | 1.5 | - |  |  | “ |
| 6 | 13 | 2.4 | - |  |  | “ |
| 7 | 15 | 8.6 | - |  |  | “ |
| 8 | 17 | 269 | 852 | 4901 | 1263020 | “ |
| 9 | 19 | 4808 | 9365 | 47199 | 16067299 | Both SolverStudio and CBC |
| 10 | 21 | 24133 |  | 56483 | 32975824 | Run from CBC Command line |
| 11 | 23 |  | 2939 | 1663 | 1799546 | Run from CBC Command line  Feasible solution initialized using mipstart |

**Conclusion**

In most instances, a human is able to solve the puzzle faster than this formulation, (particularly if you consider the manual generation of the distance matrix). I believe this is due to the fact that this formulation leaves out details about the grid and just looks at the puzzle as a network. You don’t get

|  |  |
| --- | --- |
|  | any information about overlaying routes for multiple passengers with this formulation. I know that when I solve the puzzle manually, I estimate a solution moment as to whether the main flow of the solution needs to be clockwise or counter clockwise and I look for redundant paths. For example, “2-1-4” and 10-9-12 are patterns that fit the problem on the left. This information about the grid and overlaying paths is not visible in this formulation. None-the-less, it is interesting to see the linear programming solver achieve a valid solution. |

There may be better formulations for this particular problem, but it has served its purpose and allowed me to get up to speed on optimization tools like AMPL, PuLP, and SolverStudio.

**SolverStudio**

I love SolverStudio. It’s a great tool and greatly facilitated my learning process. I was able to figure out most things by looking at the provided examples and searching the internet, including modifying the PuLP source to save a log file [17]. When I didn’t get enough debug information from SolverStudio, I would just jump to the AMPL IDE to run the model. Similarly, I would run the PuLP model in SolverStudio with the command “solve” commented out and then go to the appropriate Windows folder at: %TEMP%/SolverStudio<random> to extract the raw model file which I could them import into the command line version of CBC. And being able to switch to the NEOS server for large problem is a huge benefit.

The only drawback that I see to SolverStudio, is that unlike most other environments (AMPL & LINGO), it combines the data and model into a single Excel .xslx file, which: a) results in duplicate models for various sets of data; and b) as a non-text file, it does not lend itself to source differencing, typical of version control. I don’t think I would use SolverStudio for production work without addressing this particular shortcoming, but as an education tool, it is hard to beat.

**References**

1. <https://www.lumosity.com/press/news/en-blog/2017/5/17/pet-detective-behind-the-game>, “Pet Detective: Behind the Game”, Ethan Kennerly, [FineGameDesign.com](file:///C:\temp\PetDetective\docs\FineGameDesign.com).
2. <https://en.wikipedia.org/wiki/Vehicle_routing_problem>, “Vehicle routing problem”, Wikipedia.
3. <http://www.lindo.com/index.php/products/lingo-and-optimization-modeling>, LINGO Optimization Modeling Software, LINDO System Inc.
4. [https://www.rstudio.com/products/rstudio](https://www.rstudio.com/products/rstudio/)/, Integrated Development Environment for the “R” programming language.
5. <https://solverstudio.org/>, SolverStudio is an add-in for Excel that allows you to build and solve optimization models using PuLP, COOPR/Pyomo, AMPL, GMPL, GAMS, Gurobi, CMPL, SimPy, and Python.
6. <http://pubsonline.informs.org/doi/pdf/10.1287/ited.2013.0112>, “SolverStudio: A New Tool for Better Optimisation and Simulation Modelling in Excel”, Andrew J. Mason, Informs Transactions on Education, Vol 14, No. 1, Sept. 2013, pp 45-52, ISSN 1532-0545.
7. <https://neos-server.org/neos/index.html>, NEOS Server: free internet-based service for solving numerical optimization problems.
8. [AMPL: A Modeling Language for Mathematical Programming](http://ampl.com/resources/the-ampl-book/), Robert Fourer, David M. Gay, and Brian W. Kernighan, 2nd edition, ISBN 0-534-38809-4.
9. <http://ojs.pythonpapers.org/index.php/tppm/article/download/111/112>, An Introduction to pulp for Python Programmers, Stuart Mitchell, The Python Paper Monograph, Vol. 1 (2009).
10. <http://www.optimization-online.org/DB_FILE/2011/09/3178.pdf>, PuLP: A Linear Programming Toolkit for Python, Stuart Mitchell, Michael O’Sullivan, Iain Dunning, Sept 5, 2011.
11. <http://www-bcf.usc.edu/~maged/publications/MultiplePickup.pdf>, An Exact Algorithm for the Multiple Vehicle Pickup and Delivery Problem, Quan Lu, Maged Dessouky, Dec. 12, 2002.
12. <http://www-bcf.usc.edu/~maged/publications/pdcongestion.pdf>, Xiaoqing Wang, Maged Dessouky, Fernando Ordonez, circa 2014.
13. <https://en.wikipedia.org/wiki/Hamiltonian_path>, Hamiltonian path.
14. [Applied Integer Programming: Modeling and Solution](http://www.wiley.com/WileyCDA/WileyTitle/productCd-0470373067.html), DerSan Chen, Robert G. Batson, Yu Dang, ISBM: 978-0-470-37306-4, section 6.4 “Formulating Asymmetric TSP”, p 142-145.
15. <https://github.com/PointOnePA/SolverStudio-Examples>, source files on GitHub.com.
16. <https://groups.google.com/forum/#!topic/pulp-or-discuss/GC6Ytid_4lI>, paraphrased: “pulp does not support …an easy way to create a mip start”, “list of strings that you want to pass cbc as a commandline option”, Stuart Mitchell, 3/15/2015.
17. <https://stackoverflow.com/questions/26642029/writing-coin-or-cbc-log-file>, “Writing COIN-OR CBC Log File using PuLP.

**Files**

All of the files for this investigation are available on GitHub [15].

**Example Problem with Cost/Distance Matrix (PetD8)**

|  |  |
| --- | --- |
| **PetD8** contains 8 pets to be picked up and delivered to 8 homes for a total of 16 service nodes and 17 nodes total when including the depot. The pet nodes are numbered first, 1-8, and the homes are numbered 9-16, with the “depot” or starting/end point labeled 17.  The vehicle capacity (CAP) is 4 pets. The goal objective is 29. The pet nodes (pickup) have a value G=+1, whereas the home nodes (drop-off) have a value G=(-1). There is more than one optimal solution, but one optimal solution is the numbered order shown to the right. | 17: 7 6 14 2 1 4 15 10 9 8 5 3 16 13 12 11 :17 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **N** | **17** |  |  | M | 8 |  |  | KNOWN\_OBJ | | | 29 |  | **CAP** | 4 |  |  |  |
| **Nodes** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | **17** |
| **NReduced** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |  |
| **Pickup** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |  |
| **G** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |  |
| **zout** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **cost** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 1 | 0 | 1 | 4 | 1 | 4 | 3 | 4 | 7 | 6 | 3 | 5 | 8 | 7 | 3 | 2 | 5 | 4 |
| 2 | 1 | 0 | 3 | 2 | 3 | 2 | 3 | 6 | 5 | 4 | 4 | 7 | 6 | 2 | 3 | 4 | 3 |
| 3 | 4 | 3 | 0 | 5 | 2 | 5 | 6 | 3 | 4 | 7 | 5 | 4 | 3 | 5 | 6 | 1 | 6 |
| 4 | 1 | 2 | 5 | 0 | 5 | 2 | 3 | 6 | 5 | 2 | 4 | 7 | 8 | 4 | 1 | 6 | 5 |
| 5 | 4 | 3 | 2 | 5 | 0 | 5 | 6 | 5 | 6 | 7 | 7 | 6 | 5 | 5 | 6 | 3 | 6 |
| 6 | 3 | 2 | 5 | 2 | 5 | 0 | 1 | 4 | 3 | 2 | 2 | 5 | 6 | 2 | 1 | 6 | 3 |
| **7** | 4 | 3 | 6 | 3 | 6 | 1 | 0 | 3 | 2 | 1 | 1 | 4 | 5 | 3 | 2 | 7 | 2 |
| **8** | 7 | 6 | 3 | 6 | 5 | 4 | 3 | 0 | 1 | 4 | 2 | 1 | 2 | 4 | 5 | 4 | 3 |
| **9** | 6 | 5 | 4 | 5 | 6 | 3 | 2 | 1 | 0 | 3 | 1 | 2 | 3 | 3 | 4 | 5 | 0 |
| **10** | 3 | 4 | 7 | 2 | 7 | 2 | 1 | 4 | 3 | 0 | 2 | 5 | 6 | 4 | 1 | 8 | 0 |
| **11** | 5 | 4 | 5 | 4 | 7 | 2 | 1 | 2 | 1 | 2 | 0 | 3 | 4 | 2 | 3 | 6 | 0 |
| **12** | 8 | 7 | 4 | 7 | 6 | 5 | 4 | 1 | 2 | 5 | 3 | 0 | 1 | 5 | 6 | 3 | 0 |
| **13** | 7 | 6 | 3 | 8 | 5 | 6 | 5 | 2 | 3 | 6 | 4 | 1 | 0 | 6 | 7 | 2 | 0 |
| **14** | 3 | 2 | 5 | 4 | 5 | 2 | 3 | 4 | 3 | 4 | 2 | 5 | 6 | 0 | 3 | 6 | 0 |
| **15** | 2 | 3 | 6 | 1 | 6 | 1 | 2 | 5 | 4 | 1 | 3 | 6 | 7 | 3 | 0 | 7 | 0 |
| **16** | 5 | 4 | 1 | 6 | 3 | 6 | 7 | 4 | 5 | 8 | 6 | 3 | 2 | 6 | 7 | 0 | 0 |
| **17** | 4 | 3 | 6 | 5 | 6 | 3 | 2 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Note the zero values in the violet squares force the return to depot cost to zero while keeping the cost matrix symmetric.

The solution to the PET8 problem is presented below. The size of the matrix provides an idea of the number of binary variables for which a solution must be obtained. Some of the constrained cells are also shown colored with a blue background. All of the values in b and x are binary.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **b** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | **1** |
| 2 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | **1** |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | **1** |
| 4 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | **1** |
| 5 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | **1** |
| 6 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **1** |
| **7** | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **1** |
| **8** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | **1** |
| **9** | **0** | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| **10** | 0 | **0** | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| **11** | 0 | 0 | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **12** | 0 | 0 | 0 | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| **13** | 0 | 0 | 0 | 0 | **0** | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| **14** | 1 | 1 | 1 | 1 | 1 | **0** | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| **15** | 0 | 0 | 1 | 0 | 1 | 0 | **0** | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| **16** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **0** | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| **17** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | 0 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **x** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **0** |
| 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **0** |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | **0** |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | **0** |
| 5 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **0** |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | **0** |
| **7** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **0** |
| **8** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **0** |
| **9** | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **10** | 0 | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **11** | 0 | 0 | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** |
| **12** | 0 | 0 | 0 | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **13** | 0 | 0 | 0 | 0 | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **14** | 0 | 1 | 0 | 0 | 0 | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **15** | 0 | 0 | 0 | 0 | 0 | 0 | **0** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **16** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **0** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **17** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **solve\_result** |  | **Total\_Cost** |  | **Order** |
| solved |  | 29 |  | **17 7 6 14 2 1 4 15 10 9 8 5 3 16 13 12 11 17** | |

**AMPL MODEL EQUATIONS**

set Nodes ordered; # All Nodes

set Nreduced ordered; # Nodes{1:2M}

set Pickup ordered; # Nodes{1:M}

param N integer; # all node count

param M integer; # pickup count

param CAP integer;

param KNOWN\_OBJ integer;

param G{Nreduced} integer;

param zout{Nreduced} integer;

param cost {Nodes,Nodes} >= 0; # Costs

# Decision Variables --------------------------------------------------

# x contains the (0,1) binary variable to define the arcs used

var x {i in Nodes, j in Nodes} binary ;

# b contains the (0,1) binary variable to define if node i occurs before j

var b {i in Nodes, j in Nodes} binary ;

# zout is not a decision variable. It’s calculated from the resulting G[j],B[i,j]

# OBJECTIVE -----------------------------------------------------------------

# The objective function is added

minimize Total\_Cost:

sum {i in Nodes, j in Nodes} cost[i,j] \* x[i,j];

# CONSTRAINTS ---------------------------------------------------------------

#1 Basic Network, one entry, one exit per node

subject to LinksOutPerNode{k in Nodes}:

sum {j in Nodes} x[j,k] = 1;

#2

subject to LinksIn\_PerNode{k in Nodes}:

sum {i in Nodes} x[k,i] = 1;

#Copy Constraints. If x[i,j]=1, then node i isimmediately before node j,

#3 and b[k,i]=b[k,j] for all k, where k<>i

# Does not include DEPOT

subject to copy1{i in Nreduced, j in Nreduced, k in Nreduced: k<>i }:

b[k,i] <= b[k,j] + (1-x[i,j]);

#4

subject to copy2{i in Nreduced, j in Nreduced, k in Nreduced: k<>i and k<>j }:

b[k,j] <= b[k,i] + (1-x[i,j]);

#5

subject to copy3{i in Nreduced, j in Nreduced: i<>j} :

x[i,j] <= b[i,j];

#6 Diagonals

subject to DiagonalsB{i in Nodes}:

b[i,i] = 0;

#not useful

subject to DiagonalsX{i in Nodes}:

x[i,i] = 0;

#7 PRIOR CONSTRAINTS, pickup before delivery

subject to Prior1{i in Pickup}:

b[M+i,i] = 0;

#8 PRIOR Pickup must be before delivery

subject to Prior2{i in Pickup}:

b[i,M+i] = 1;

#10 CAPACITY CONSTRAINT tested at pickup location only

subject to Capacity{j in Pickup}:

sum {i in Nreduced} b[i,j]\*G[i] <= CAP-G[j];

#11 FIRST node won’t be a drop-off

subject to First{i in Pickup}:

x[N,M+i] == 0;

#14 LAST node won’t be from a pickup

subject to Last{i in Pickup}:

x[i,N] == 0;

#Extra since Depot is start and end.

Subject to Start{i in Nreduced}:

b[N,i] == 1;

subject to End{i in Nreduced}:

b[i,N] == 1;

#Lu&Dessouky SPEED IMPROVEMENT CONSTRAINTS

#4 (eq25) Vehicle Return Constraint cuts some fractional solutions (1 vehicle, simplified)

subject to VehicleReturn:

sum {k in Nreduced} b[k,N] == N-1;

#5 (eq26) Equality Constraint from definition of the b variables

subject to Equality26{i in Nreduced, j in Nreduced: i<>j}:

b[i,j] + b[j,i] == 1;

#Limit Objective to Known Target Objective

subject to MIN\_OBJ:

sum {i in Nodes, j in Nodes} cost[i,j] \* x[i,j] >= KNOWN\_OBJ;

#===================================================================

# Process Results

#===================================================================

# Get the data from the sheet

data SheetData.dat;

option solver cbc; #switch to COIN-CMDC solver

option show\_stats 1;

#option gentimes 1; # display summary of resources used

#CPLEX OPTIONS FOR AMPL

#option mipdisplay 3; #frequency of displaying branch-bound info

#option lowerobj KNOWN\_OBJ+1; #

#option tunedisplay 3; # exhaustive printing

solve;

#solution – pull the SolverStudio variables on to the sheet

display x > Sheet;

display b > Sheet;

#solver info

display solve\_result > Sheet;

display Total\_Cost > Sheet;

#Calculate zout

let {j in Nreduced} zout[j] := G[j] + sum{i in Nreduced}round(b[i,j],0)\*G[i];

display zout > Sheet;

#EXTRA DEBUG INFO

display \_ampl\_elapsed\_time, \_ampl\_system\_time, \_ampl\_user\_time;

display \_solve\_elapsed\_time, \_solve\_system\_time, \_solve\_user\_time;

\*\*\* END OF AMPL MODEL \*\*\*

**PuLP MODELING EQUATIONS**

# Import PuLP modeler functions

from pulp import \*

# Import math functions

from math import \*

import datetime as dt

t1 = dt.datetime.now()

# Parameters

Debug = 0

# Create the 'prob' variable to hold the data

prob = LpProblem("PetProject", LpMinimize)

# Creates a list of tuples containing all the possible Nodes, skip i=j

ARCS = [(i,j) for i in Nodes for j in Nodes if i!=j ]

# Decision Variables --------------------------------------------------

# x contains the (0,1) binary variable to define the arcs used

x = LpVariable.dicts("x",(Nodes,Nodes),0,1,LpBinary)

# b contains the (0,1) binary variable to define if node i occurs before j

b = LpVariable.dicts("b",(Nodes,Nodes),0,1,LpBinary)

# z is not a decision variable. It's calculated from the resulting G[j],B[i,j]

# OBJECTIVE -----------------------------------------------------------------

# The objective function is added to 'prob' first

prob += lpSum([x[i][j]\*costs[i,j] for (i,j) in ARCS ]),"Total\_Costs"

#print prob

# CONSTRAINTS ---------------------------------------------------------------

#1 Basic Network, one entry, one exit per node

for k in Nodes:

prob += lpSum([x[j][k] for j in Nodes]) == 1,"Basic\_One\_Entry\_%d"%k #sumcols

#2

for k in Nodes:

prob += lpSum([x[k][i] for i in Nodes]) == 1,"Basic\_One\_Exit\_%d"%k

#Copy Constraints. if x[i,j]=1, then node i isimmediately before node j,

#3 and b[k,i]=b[k,j] for all k, where k<>i

for i in NReduced:

for j in NReduced:

for k in NReduced:

if k != i:

prob += b[k][i] <= b[k][j] + (1-x[i][j]),"copy1\_%d\_%d\_%d"%(i,j,k)

#4

for i in NReduced:

for j in NReduced:

for k in NReduced:

if k != i:

prob += b[k][j] <= b[k][i] + (1-x[i][j]),"copy2\_%d\_%d\_%d"%(i,j,k)

#5

for i in Nodes:

for j in Nodes:

prob += (x[i][j] <= b[i][j] ),"copy3\_%d\_%d"% (i,j)

#6 Diagonals

for i in Nodes:

prob += b[i][i] == 0, "DiagonalB\_%d"%i

#for i in Nodes: #(not useful)

prob += x[i][i] == 0, "DiagonalX\_%d"%i

#7 PRIOR CONSTRAINTS, pickup before delivery

for i in Pickup:

prob += b[M+i][i] == 0,"Prior1\_%d"%i

#8 PRIOR Pickup must be before dlivery

for i in Pickup:

prob += b[i][M+i] == 1,"Prior2\_%d"%i

#10 CAPACITY CONSTRAINT with vehicle initially having zero passengers

for i in NReduced:

prob += lpSum(b[j][i]\*G[j] for j in NReduced) <= CAP-G[i],"Capacity\_%d\_%d"% (i,CAP-G[i])

#11 FIRST node won't be a drop-off

for i in Pickup:

prob += x[N][M+i] == 0,"First\_%d"%i

#14 LAST node won't be from a pickup

for i in Pickup:

prob += x[i][N] == 0,"Last\_%d"%i

#Extra since Depot is start and end.

for i in NReduced:

prob += b[N][i] == 1,"Start\_%d"%i

for i in NReduced:

prob += b[i][N] == 1,"End\_%d"%i

#Lu&Dessouky SPEED IMPROVEMENT CONSTRAINTS

#1 (eq 21) Transfer Prior Constraint

# (too hard to implement because of arbitrary collection)

#2 (eq 22,23) Adjacent Prior Constraints DOESN'T HELP

#for k in Nodes:

# for i in Pickup:

# if(i != k and i+M != k):

# prob += b[k][i]+b[k][i+M] >= x[i ][k]+x[k][i], "AdjacentPriorA\_%d\_%d"%(k,i)

# prob += b[i][k]+b[i+M][k] >= x[i+M][k]+x[k][i+M],"AdjacentPriorB\_%d\_%d"%(k,i)

#3 (eq24) Pairing Prior Constraint DOESN'T HELP

#for i in Pickup:

# prob += lpSum(b[k][i] for k in Nodes)+1 <=

lpSum(b[k][i+M] for k in Nodes),"PairingPrior\_%d"%i

#4 (eq25) Vehicle Return Constraint cuts some fractional solutions (1 vehicle, simplified)

prob += lpSum([b[k][N] for k in NReduced]) == N-1,"VehicleReturn"

#5 (eq26) Equality Constraint from definition of the b variables

for i in NReduced:

for j in NReduced:

if i != j:

prob += b[i][j] + b[j][i] == 1,"Equality26\_%d\_%d"%(i,j)

#Limit Objective to Target Objective or above to truncate minimum search

prob += prob.objective >= KNOWN\_OBJ, "MIN\_OBJ"

#SOLVER ---------------------------------------------------------------

prob.writeLP("PD9.lp") # Write the problem as an LP file

prob.solve() # Solve the problem using the default solver

#prob.solve(COIN\_CMD(msg=1)) #solve using CBC with logging

#prob.solve(COIN\_CMD(msg=1, fracGap = 0.0)) #solve using CBC

#CALC TIME ---------------------------------------------------------------

t2 = dt.datetime.now()

dt = t2-t1

print 'Time: %6.3f seconds' %(dt.seconds+dt.microseconds/1e6) #modulo printing

#DISPLAY SOLUTION ---------------------------------------------------------------

solve\_result = LpStatus[prob.status]

''' write to spreadsheet '''

solve\_result = LpStatus[prob.status]

for (i,j) in ARCS:

xout[i,j] = x[i][j].varValue

for (i,j) in ARCS:

bout[i,j] = b[i][j].varValue

print("Status:", LpStatus[prob.status])

TotalCost = lpSum([x[i][j].varValue \* costs[i,j] for (i,j) in ARCS ])

print 'TotalCost = %s' % TotalCost

Total\_Cost = value(prob.objective)

#display solution X

print ' x ',

for i in Nodes:

print '%2.0f' % i,

print

for i in Nodes:

print '%2.0f ' % i,

for j in Nodes:

if i!=j:

print '%2.0f' % x[i][j].varValue ,

else:

print " 0",

print

print

#Diplay Capacity at nodes

print ' z '

for i in NReduced:

z = G[i]

for j in NReduced:

z = z + G[j]\*b[j][i].varValue

print '%2.0f'%z,

zout[i] = z #write to spreadsheet

print

\*\*\* End of PuLP Model \*\*\*

\* \* \* EOF \* \* \*