Phoenix: Integer Programming Using Branch and Bound

Chris Mascioli
cmasciol

April 2022

1 Introduction

For this project I used `CPLEX` (in Python) to solve the problem.

2 Model

2.1 Decision Variables

There were 3 sets of decision variables

1. A $1 \times$ n_tests vector u, where

$$u_j = \begin{cases} 1 & \text{If test } j \text{ was used} \\ 0 & \text{Otherwise} \end{cases}$$

2. A $1 \times$ n_diseases vector s, where

$$s_j = \begin{cases} 1 & \text{If disease } j \text{ was not tested for} \\ 0 & \text{Otherwise} \end{cases}$$

3. A n_diseases $\times$ n_diseases matrix G, where

$$G_{jk} = \begin{cases} 1 & \text{If disease } j \text{ is tested more than } j \text{ and } k \text{ are tested together} \\ 0 & \text{Otherwise} \end{cases}$$

2.2 Constraints

There were four constraints

1. $\sum_j s_j = 1$. At most one disease is not tested for.
2. $\sum_i A_{ij} u_i \geq 1 - s_j$. All diseases except one must be tested at least once
3. $G_{jk} + G_{kj} = 1$. Only one of the pair has to be tested more than the pair is tested.
4. $\sum_i (A_{ij} u_i - A_{ij} A_{ik} u_i) \geq G_{jk}$. The number of times the first disease disease in a pair is tested for without the second member of the pair has to be more than the pair is tested together if the G value for that pair is 1.

3 Branch and Bound

3.1 Node Selection

Both depth-first search and best-first search are implemented in the solution. Best-first search performed significantly better (almost a 2x speedup) compared to using DFS. I also tried using a combination of the two (allowing DFS to get to a certain depth for all nodes and then using best), but that didn't create any performance improvements. This is handled in the `branch` function of the solver and best-first search is implemented using `heapq`.

3.2 Variable Selection

The variables are selected with some problem-specific context in mind.

1. First, it checks if all $u_j \in \mathcal{Z}$. If they are not, it picks the one which is furthest from being integer-valued (closest to .5) and chooses to branch on that
2. Next, it checks s to see if it has a 1. If it does not, it picks the variable closest to 1 and branches on that
3. Lastly, it looks for non-integer members of G and chooses to branch on the one furthest from being integer-valued

This is the configuration that gave the best performance, other variations in both order and how to pick the variable were used and this one ended up being the best. This is implemented in the `get_next` method.

3.3 Cutting

I decided to use Python because an answer online indicated that you could access the simplex tableau using docplex (which wasn't available in Java) in order to add Gomory cuts. Unfortunately, accessing the tableau sometimes gives errors about it not being LU factorized, so generating cuts broke (and accessing it via the `get_cplex` method was undocumented so it probably isn't a particularly stable way to do things).

4 Results

The correct answer was reached on all but two instances within the 5 minute time limit (and all finished under 15 minutes). It's possible that Java would have finished all in under 5, but I chose python to do cuts and by the time I learned they didn't work, I had already implemented the full solution here. Results were extremely sensitive to the type of search (DFS vs best-first) and the way variables were selected (changing to checking s before u was around a 20% slowdown, for instance).

5 Future Directions

I wanted to try and implement something similar to [Eth+20] or [BLP20], but it just wasn't practical for the timeframe of the project. I also don't think, given the small number of instances, it would produce any particularly interesting results. I don't think the Etheve results would generalize particularly well if all of the instances weren't drawn from the same generative function (or real life process) so, depending on how the instances for our inputs are created, may not even work.

6 Logistics

I spent around 25 hours on the project. A lot of these hours were just testing different problem-specific things like order of variables to assign, BFS vs DFS vs a combination, and how to choose which variable to branch on. I wish I documented it better to capture a table to compare the results generated.

References

[BLP20] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. “Machine Learning for Combinatorial Optimization: a Methodological Tour d’Horizon”. In: *arXiv:1811.06128 [cs, stat]* (Mar. 2020). arXiv: 1811.06128. URL: `http://arxiv.org/abs/1811.06128` (visited on 04/04/2022).

[Eth+20] Marc Etheve et al. “Reinforcement Learning for Variable Selection in a Branch and Bound Algorithm”. In: *arXiv:2005.10026 [cs, stat]* 12296 (2020). arXiv: 2005.10026, pp. 176–185. DOI: `10.1007/978-3-030-58942-4_12`. URL: `http://arxiv.org/abs/2005.10026` (visited on 04/04/2022).