CU Boulder Fall 2018

CSCI 3202: Introduction to Artificial Intelligence

Practicum

Members: Krishna Dholakiya, Orgil Sugar

Making a venture capital 'Demo Day' more time-efficient, as a constraint-solving problem.

Before this semester started, I (Krish) worked with a small venture capital firm to solve the following problem. They had just completed a summer accelerator with 11 portfolio companies, and wanted to see if they could automate scheduling Q&A sessions between those companies and the investors in attendance at their summer 'demo day'. The investors were also able to rank their top 5 companies based on the demos, and the tool would incorporate that preference into the scheduling.

Back then, my solution was fairly brute-force (essentially went through investors, checked what the highest-ranking 'unscheduled' startup on their list was, and scheduled them). This practicum seemed like an excuse to revisit the problem and see if it could be solved any smarter.

Methods

Survey of options:

The following were methods we researched as potential solutions.

- Ford Fulkerson method for maxflow[1]
    - running the algorithm on a bipartite graph – consisting of startup nodes in one set and investors with their available hours in another set – would give

[1] https://www.geeksforgeeks.org/maximum-bipartite-matching/

us the most optimal set of connections (based on the weights of the edges between the nodes)

- Dinic's algorithm for maxflow[2]
    - Dinic's algorithm is a more efficient version of Ford-Fulkerson, in contexts where the graph is densely connected
- Stable marriage problem[3]
    - This is a similar approach to the max-flow algorithms mentioned, except it'd be more flexible in incorporating the investors' preferences
- Hopcroft-Karp algorithm[4]
    - This is another bipartite matching algorithm, this time we found it specifically prescribed (in a few places) as a good 'scheduling' algorithm, as it tends to be more efficient than the other bipartite algorithms mentioned + it tries to make the maximum number of matches (we want the maximum number of possible 'pairings' between startups/investors)
- A* search-based scheduling algorithm[5]
    - We found an interesting IEEE paper that proposed a way in which you could use the A* search algorithm for meeting scheduling specifically
- Graph coloring with backtracking[6]
    - This one was familiar from our time in this course – since we've already used recursive backtracking to tackle constraint-solving problems – and would allow us to color nodes with a small set of 'colors', such that no two connected nodes have the same color

Ultimately, in evaluating our options, it boiled down to whether we wanted to treat this problem as a bipartite matching problem (in which 'connecting' startup nodes with investor nodes would be our 'scheduling'), or as a graph coloring problem that was less

---

[2] https://www.geeksforgeeks.org/dinics-algorithm-maximum-flow/
[3] https://www.geeksforgeeks.org/stable-marriage-problem/
[4] https://stackoverflow.com/a/11144216
[5] https://ieeexplore.ieee.org/document/1273357
[6] https://www.geeksforgeeks.org/m-coloring-problem-backtracking-5/

about ‘matching’ and more explicitly about scheduling. Since graph coloring/backtracking were things we learned in this class, we went with that.

The method we went with:

To recap, our objective was to take three things as input:

- A set of startups
- A set of investors, with those investors’ top *n* preferred startups
- The amount of time available for this event / the minimum time in a time slot

and give back a schedule that ensures that all (or most) investors are able to meet their preferred startups for Q&A (while letting startups do Q&As in parallel instead of one at a time).

Since we went with graph coloring, our first step was to figure out how to contextualize our variables in a graph coloring context. It made sense to make startups the nodes, and to constrain based on investor preference by drawing edges between startups that had the same investor preferring them. The ‘colors’ in this context would be the time slot when a particular startup would hold their Q&A. For example, if Warren Buffet was interested in attending both Facebook and Google’s Q&As, we’d draw an edge between Facebook and Google on the graph, and then when graph coloring was applied to such a graph, Facebook and Google would have to be assigned different time slots so Warren Buffet could be in attendance for both.

Our methodology was as follows. We first needed a sample dataset of startups, investors, and those investors’ rankings of those startups -- we built a function with which you could randomly generate such a dataset, as well as tweak the quantity of those three things. We then needed to be able to draw the graph, such that those investor rankings were incorporated into where edges were drawn.

Once we had the data and the corresponding graph, it was a matter of applying a traditional recursive backtracking algorithm to most optimally assign startups time slots, based on the constraint of no investor being forced to choose between the startups they want to attend. As we'll discuss in our results section in more depth, that constraint is fairly strict (and not the most robust for real-life scenarios), and so we also needed to be able to loosen that constraint (which means we needed to draw fewer edges in the graph).

Results

As the code in our Jupyter Notebook demonstrates, we were able to use recursive backtracking to solve this problem, tested with randomly generated data at the proportions of the actual event (11 portfolio companies, 35 investors present). However, we found two corollary experiments to be worthwhile, since these types of events are rarely predictable in terms of numbers/growth etc:

- How scaling the ***no. of investors***, ***no. of startups***, and the ***no. of startups that investors could list preference for*** would impact the solvability + number of time slots required to make the problem solvable.
- At scales at which the problem was no longer solvable (within a 2 hour time period, where time slots must at least be 15 minutes): ***how could we loosen the degree to which we constrain the problem, such that we don't completely throw out investor preference but still allow for some conflicts?***

Scale

The following table describes how increases in the no. of investors, the no. of startups, and the no. of startups investors prefer, impacted whether they were solvable + the number of time slots required. We 'grew' the no. of investors way more, since in the real-life circumstance of a VC demo day, that number growing is far more likely/significant than the no. of startups in a particular cohort.

No. of startups	No. of investors	Time Slots
11	35	4
11	60	7
11	85	7
11	110	8
11	200	Couldn’t be solved.
20	35	3
30	35	3
40	35	3

What this seems to demonstrate (pretty logically) is that as you increase the number of investors, the more time slots you require. Additionally, given that time slots must be at least 15 minutes within the 2 hour window, there’s a limit to how many investors this works with (it didn’t work with 200).

Also, when we adjust the # of startups the investors could list as preferred (increasing it from 2), it stopped being solvable (the number of edges grows non-linearly with it).

To summarize: the simple version of our algorithm struggles with investors scaling beyond a certain number, as well as the investors being able to list more than 2 preferred startups. The logical way to try to deal with this seemed to be ‘loosening’ how constrained our graph was.

Reducing Constraint Strictness

The simple version disallows startups from being slotted with any other startup, when there exists an investor that prefers both of them; it does so by drawing an edge between any such two startups. To ‘loosen’ such a constraint, we’d have to decrease the certainty at which an edge would be drawn in such a conflict. The default is with 100% probability: we experimented with decrements of 10% from there.

Variable being scaled	**Degree of constraint (probability of drawing an edge)**	**No. of time slots**
200 investors	100%	Can’t be solved
200 investors	90%	8
200 investors	80%	6
200 investors	70%	5
200 investors	60%	5
200 investors	50%	5
5 startup-preferences	100%	Can’t be solved
5 startup-preferences	90%	Can’t be solved
5 startup-preferences	80%	7
5 startup-preferences	70%	5
5 startup-preferences	60%	5

5 startup-preferences	50%	4

As is evident, at least within our test data, loosening our constraints does eventually solve the problems of scale: the green highlighted cells are what we'd consider 'optimal' degrees of constraint since the problems suddenly become solvable at their respective scales. If we decided that we wanted even fewer time slots (to increase the duration of one time slot), we could further reduce the degree of constraint.

This 'degree of constraint' parameter – when applied to a more unpredictable real-life application of our tool – ultimately lets us analyze the scale of the event and determine how much 'constrainedness' we can afford while still solving the problem.

Discussion/Conclusions

Treating this as a constraint-solving problem seemed fairly logical at the outset, and worked pretty well. There was an initial concern that something like recursive backtracking – at least in it's most common form – wouldn't be flexible enough to allow anything but a 'perfect' solution, in which no startups that were in the same investor's preferences could hold their Q&A sessions in parallel (so investors could attend every single one they were interested in). In a real-life circumstance, such as a demo day like this, 'no solution' is simply not an option -- if human-managed, the fix would be to randomly make a few investors have to compromise so that a solution was possible at all.

We essentially used random chance to introduce entropy and increase flexibility/solvability: in the future, it'd possibly be worth figuring out a more 'informed' way to draw the edges and swapping our our edge-drawing function with that. It'd also be worthwhile to automate the process of figuring out the 'best' degree of constraint, instead of currently requiring a human assessment of all the metrics to make that

decision. Also, in generating our test data in the future, we could account for ‘power laws’: it’s possible that in the wild, 80% of the investors would prefer 20% of the startups, which would introduce new challenges worth exploring.

In the pursuit of matching human flexibility/compromise, our exploration of the ‘degree of constraint’ parameter in determining the constraints of the system was valuable. We learned to assess whether a solution scales well in different real-life proportions, as well as how to introduce human-like flexibility into systems that default to being either optimally solved or not solvable at all.