Improvements on a Time Slot Allocation Algorithm

Lewin Gan Allen Xiao Edwin Liao

1. MOTIVATION

At the beginning of every semester, HKN has the task of scheduling its officers to tutoring time/location slots based on each person's slot preferences. This is similar to the General Assignment Problem in that each slot (task) must be assigned one or more tutors (agents). Unlike the General Assignment Problem (for which each agent is assigned up to its task capacity), we are enforcing that each officer is assigned to exactly 2 slots.

The previous scheduling algorithm used linear programming to find some feasible fractional solution, then improved on it with an inefficient hill-climbing algorithm that consisted of trying 3-swaps between all 3-tuples of tutor-slot assignments. This algorithm's documentation claimed that it could take up to 10 hours, and suggested that it be run overnight.

2. FEATURES

To sign up as a tutor, each tutor fills out a web page listing the following:

- Time slot availabilities and preference levels
- Preferred office (Soda or Cory) for each time slot
- Courses they are comfortable tutoring
- Whether adjacent tutoring slots are preferred

This last preference prevents our problem from being modeled as a linear program, unless we have constraints that are exponential in number relative to the number of slots.

We also have committee members: non-officer tutors who are assigned 1 slot instead of 2.

3. THE ALGORITHM

We assume that there are enough tutor assignments such that each slot will be assigned at least one tutor. While there exist unfilled tutoring slots, we use maximum weight matching to match each tutor to one slot, then remove all assigned slots. Once all slots have been filled, we bring back all slots and repeat this process until each tutor had their tutoring capacity satisfied.

During each iteration of this algorithm, each tutor is only assigned one slot. Furthermore, we can change edge weights between iterations. This allows us to take into consideration each tutor's adjacent slot preference, as well as prevent tutors from being assigned to concurrent time slots.

Our algorithm greedily applies maximum weight matching for each iteration. Let's say our algorithm finds some matching with value X in its first iteration. All subsequent iterations will find a matching with value $\leq X$. In the worst case scenario, the first iteration will find a matching with some value X and all other iterations will find matchings with value 0. Thus, we have a k-approximate algorithm. In our case, each officer tutors for 2 hours, so $k = 2$.

4. EXPERIMENTAL RESULTS

We were unable to obtain past semesters' tutor preferences, so we wrote a dataset generator that randomly generates a set of fake tutors with fake preferences. We compared our maximum weight matching program's output values to that of an integer linear programming library we found online.[1] We believe that the ILP library code finds an optimal to our problem (minus the adjacency slot preferences, since the adjacency preference is difficult to encode in a linear program).

We used the following weight function for our experiments:

- Assign a weight of -1 to any tutor-slot pairings that would make a tutor be assigned to two simultaneous time slots.
- Add 10 to the weight of a pairing where the tutor either has no adjacency preference, or the tutor prefers adjacent slots and is assigned adjacent slots.
- Add 10 to the weight of a pairing where the tutor is available during this time slot; add 20 instead if the tutor prefers this time slot.
- Add $x \in [1, 5]$ to the weight of a pairing depending on how much the tutor prefers to tutor in the office of this time slot.

Using Spring 2013's tutoring preferences, we found that our maximum weight matching program performed almost exactly as well as the ILP library (total value of 732 vs. 733). On randomly generated datasets [2], the maximum weight matching program usually does 5-10% better than the ILP library. The most notable exception to this was on the dataset that had exactly 30 officers (2 slots per officer) and 60 time/location slots. Since there was exactly 1 tutor

[1] Code Readme

[2] See test/exp_results.txt and test/DatasetGenerator.py for more information on how datasets were randomly generated

assigned to each tutoring slot, the maximum weight matching program's greedy nature caused it to pick a matching that was relatively worse than usual (over 5% worse than the ILP library's solution).

5. FUTURE PLANS

There are a few potential bugs with the maximum weight matching program that require further investigation:

- On some runs, the maximum weight matching program will assign a tutor to a time slot that he/she cannot make, whereas the linear programming library will assign all tutors to slots for which they are available. This is either caused by a bug in our algorithm, or we need to fine tune our weight function.
- When run on the Spring 2013 dataset, the maximum weight matching program's solution seems to consider adjacent slot preferences just as much as the ILP library; that is, not at all. Either our code has a bug, or (more likely) the Spring 2013 dataset's tutors were too restrictive in listing their available times.[3]

[3]Many tutors listed only 3-5 available time slots. One particular officer listed only 1 available time slot, even though we had to assign him to 2 tutoring slots. We took this officer out.