CS 255 AI Coursework

Victor Morland
u1803998
Discrete Mathematics G4G1
victor.morland@warwick.ac.uk

***Abstract*—This is a report for the AI Coursework documenting in detail how each task was created and how they work. The coursework was to create a schedule from a list of modules and tutors that adhered to certain requirements and with a low cost - the basic requirements for each task being that each module (and lab) must be assigned a tutor and a slot in the schedule. This report is split up into three sections.**

1) **The first task was to find a schedule for a list of tutors and their modules over 5 days, which 5 slots to each day. No tutor could teach more than once in a day and no tutor could teach more than two modules a week.**
2) **The second task was to find a schedule for a list of tutors and their modules over 5 days, which 10 slots to each day. Which meant that every module had a lecture - worth 2 credits and a lab - worth 1 credit. A tutor could teach a maximum of 2 credits a day and a maximum of 4 credits over the week.**
3) **The third task had the same requirements as the second task but additionally there was a cost to each lecture and lab (which will be detailed further in the section) with the aim being to find the minimum cost (or as close to it as possible).**

I. Task 1

Let us refresh ourselves on the requirements for this task:

- Each module must be assigned a tutor and a slot.
- A tutor can only teach a module if the tutor's expertise topics include every topic covered by the module.
- A tutor can only teach a maximum of two different modules.
- A tutor cannot teach multiple modules in a single day.

Initially it was decided to split this task into two sub problems to make this problem more manageable.

1) Find a matching between modules and tutors (each module assigned to a valid tutor).
2) Find a schedule from this matching of modules and tutors.

Now we must show that if we find a valid matching of modules to tutors that there exists a valid schedule that can be created from this matching. This is easy to show, by first ordering the tutor, module pairs alphabetically (by the name of the tutor) and then filling the timetable horizontally starting with Monday slot 1 , Tuesday slot 1 , ... , Friday slot 5. This will always be a valid timetable as no tutor ever teaches twice on the same day they will always teach on adjacent days (if the tutor teaches two modules). All the other requirements only relate to the matching.

Not only have we shown that this problem can be split up as we want, but additionally we have found a valid schedule for any valid matching. So the only problem left to solve is to find a matching of modules to tutors.

Finding a Valid Matching

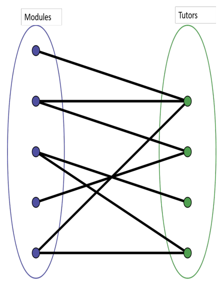


Fig. 1. Bipartite Graph of Modules to Tutors.

We will model this problem as a search problem by using a bipartite graph (observe figure 1), where one set is made up of modules and the other set is made up of tutors. There is an edge between an module and a tutor if the tutor can teach the module.

Let us define a *Covering* of the modules - for every module in the set of modules there exists exactly one edge in the graph where the module is the end point for this edge.

Now the problem is simply to find a special covering of the modules where every tutor has a maximum of degree of 2 i.e. it only teaches two modules. This can be modelled as a search tree where each child is when a module has been assigned to a tutor.

This graph was implemented with an adjacency list. To search through the search tree I used a variant of DFS based search using a specific search strategy.

The search strategy has two steps:

1) Order the modules by the number of tutors that could teach it (the fewer tutors that could teach the module the earlier this sub-tree was searched)
2) Order the tutors by the number of modules they could teach (the fewer modules that could they could teach the earlier this sub-tree was searched)

This was done to reduce the size of the search tree as quickly as possible as if a module can only be taught by one tutor then may as well assign them immediately rather than later and let that tutor be taken up assigning some other module. This makes it so the hardest choices are taken last (the choices that are most likely to be mistakes) which means that the program spends less time backtracking and so takes less time. Essentially a fail first heuristic is used here.

The second part of the strategy picks the tutor who can teach the least modules first so that it is less likely for us to end up in a situation where a tutor is teaching more than 2 credits and therefore forcing us to backtrack. Again it makes the easiest decisions as early as possible so as to reduce the size of the search tree.

The search for the matching can be summarised as following:

For all the modules in the graph pick the first valid tutor, if a module has no valid tutors backtrack and pick the next valid tutor. Do this until all modules have been assigned a tutor.

Evaluation

This technique guarantees that a matching will be found and in almost all cases will be found almost instantly as it does search through all possible matchings starting with the ones most likely to be valid. Additionally creating the schedule as discussed earlier is immediate and guarantees a valid schedule.

II. Task 2

Let us refresh ourselves on the requirements for this task:

- Each lecture and lab must be assigned a tutor and a slot (each module has a lecture and a lab).
- A tutor can only teach a lecture if the tutor's expertise topics include every topic covered by the module.
- A tutor can teach a lab session if their expertise topics include at least one topic covered bu the module
- A tutor can only teach a maximum of 4 credits (Modules count as 2 credits and labs as 1 credit).
- A tutor cannot teach more than 2 credits a day.

Again it was decided to split this task into two sub problems to make this problem more manageable.

1) Find a matching between lectures and tutors, and labs and tutors (each lecture assigned to a tutor and each lab assigned to a tutor for each module).
2) Find a schedule from this matching of modules and tutors.

Using a similar argument as for task 1, we can show that any matching can be created into a valid schedule. Using the same technique as in task 1 of grouping up the lecture, tutor and lab, tutor pairs by the name of the tutor and assigning them horizontally starting with Monday slot 1 , Tuesday slot 1 , ... , Friday slot 10. This works as at most a lecture teaches 4 things but they will all be one day after the other and since there are 5 days they will never teach twice in a day so they will never teach more than 2 credits a day. All the other requirements only relate to the matching. Once again this is an algorithm we can use to create the schedule once we find a matching, so all we need now is to find a valid matching.

Finding a Valid Matching

We will again model this problem as a search problem by using a bipartite graph (observe figure 1) with slight modifications to account for the labs, where one set is made up of lectures and labs and the other set is made up of tutors. There is an edge between an lecture/lab and a tutor if the tutor can teach the module. The edge has a weight of 2 if it connects to a lecture and a weight of 1 if it connects to a lab.

Again the problem is simply to find a special covering of the lectures and labs where every tutor has a maximum of weight of 4 (the sum of the weights of the edges connected the the tutor node is at most 4).

The graph was implemented with an adjacency list. Using the same search strategy as in the first task with the necessary modifications to account for labs

The search strategy has two steps:

1) Order the lectures and labs by the number of tutors that could teach it (the fewer tutors that could teach the lecture/lab the earlier this sub-tree was searched)
2) Order the tutors by the number of modules they could teach (the fewer modules that could they could teach the earlier this sub-tree was searched), no need to edit this as if a tutor can teach the lecture it can teach the lab.

The reasoning for this strategy is exactly the same as for the first task (making the easiest decisions as early as possible to reduce the size of the search tree as quickly as possible).

Evaluation

This technique guarantees that a matching will be found and in almost all cases will be found almost instantly as it does search through all possible matchings starting with the ones most likely to be valid, this task takes slightly longer on some problems due to the increased size of the search tree. Additionally creating the schedule as discussed earlier is immediate and guarantees a valid schedule.

III. Task 3

The requirements for this task are exactly the same as for the second task. However, for this task we want to minimize the cost of the schedule. The rules for the cost are defined as follows:

- To hire a tutor to teach a single lecture costs £500. If that tutor is hired for a second lecture it costs £300. If the second lecture is on the next day it costs £100 instead of £300.
- To hire a tutor to teach a single lab session costs £250. Each subsequent lab session costs £50 less (i.e. £200, £150, £125).
- If a tutor teaches two lab sessions on the same day the cost for both is halved i.e. £250 + £200 $\rightarrow$ £125 + £100.

Again it was decided to split this task into two sub problems to make this problem more manageable.

1) Find a matching between lectures and tutors, and labs and tutors (each lecture assigned to a tutor and each lab assigned to a tutor for each module).
2) Find a schedule from this matching of modules and tutors.

Finding a Optimum Matching

To minimize the cost we want to ideally pick as few tutors as possible as that will mean more tutors will teach two lectures or two labs which decreases the cost of the overall schedule. We will do this by tweaking the search strategy so that instead of picking the tutor which teaches the fewest lectures as in task 1 and 2, we will pick the tutor that teaches the most lectures. This is the only change we will make as we still need to find a matching in a reasonable time-frame.

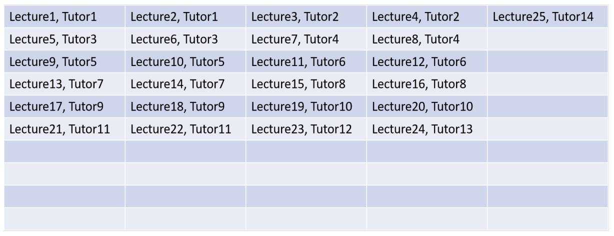
Finding a Optimum Schedule

There are two main things we want to optimise in the schedule:

1) Maximise the number of adjacent lectures (as this makes the biggest difference in terms of cost).
2) Maximise the number of labs taught by the same tutor to be on the same day.

We shall call these lecture pairs and lab pairs respectively.

Maximising Lecture Pairs



Lecture1, Tutor1	Lecture2, Tutor1	Lecture3, Tutor2	Lecture4, Tutor2	Lecture25, Tutor14
Lecture5, Tutor3	Lecture6, Tutor3	Lecture7, Tutor4	Lecture8, Tutor4	
Lecture9, Tutor5	Lecture10, Tutor5	Lecture11, Tutor6	Lecture12, Tutor6	
Lecture13, Tutor7	Lecture14, Tutor7	Lecture15, Tutor8	Lecture16, Tutor8	
Lecture17, Tutor9	Lecture18, Tutor9	Lecture19, Tutor10	Lecture20, Tutor10	
Lecture21, Tutor11	Lecture22, Tutor11	Lecture23, Tutor12	Lecture24, Tutor13	

Fig. 2. Timetable after lectures allocated

We sort the lectures in our matching into two separate lists pairs and non pairs. We then sort each list by the number of labs the tutor for the lecture teaches. The more they teach the earlier in the list they are. We then combine the two lists together. The similarly to our simple schedule in task 1 and 2 we fill in from left to right the timetable except we go from Monday slot 1 to Thursday slot 6. This has guaranteed that the cost of lectures is maximised as all lectures taught by the same tutor are adjacent. The last lecture left (as there are 25 lectures) is then put on Friday slot 1. This lecture is the one with tutor who teaches the fewest labs due to our earlier sorting. This is important as there are 9 slots left on Friday which must be filled with labs and we want to block off as few of them as possible. See figure 2.

We can get a schedule from this because from Monday to Friday there are at most 6 tutors blocked due to credits and then 1 tutor blocked off on Friday. Since there are 4 slots left on every day Monday to Thursday we can put all the labs that are blocked on one day onto the next. So now there are 9 labs left to put on Friday. These can't be blocked by the lecture on Friday as either that tutor is teaching 4 credits already or the labs have already been placed on another day as the ones blocked on Friday (a maximum of two labs) will be placed on Monday.

Maximising Lab Pairs

To maximise the number of lab pairs we want to group up the labs into pairs and non pairs (if a tutor teaches 3 labs the 2 of those labs are separated randomly and designated as a pair).We then order each list by the number of labs the tutor teaches so that the more labs they teach the earlier the tutor, lab pair is placed in the list. This again means that if a lab fails in the recursive placement of the labs in the schedule the ones most likely to fail fail as early as possible to reduce the size of the search tree and reduce the amount of backtracking required and therefore the amount of time taken to complete the schedule.

Once the labs are ordered we send it through an algorithm that assigns each lab to the first possible slot in the schedule, backtracking if there is no slot available for a lab.

Evaluation

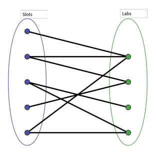


Fig. 3. Bipartite Graph of Slots to Labs.

The optimal solution for a schedule is £10,050, which is when

- 12 tutors teach 2 lectures on adjacent days
- 6 tutors teach 4 labs (with the pairs on the same days)
- 1 tutor teaches 1 lecture and 1 lab

The cost of this is:
(500 + 100) x 12 + (125 + 100 + 75 + 50) x 6 + 500 + 250 = £10,050.

My solution tended to average around £11,750 on a larger sample size which is an acceptable amount. However, the speed of my solution tended to be very fast even for some complicated schedules.
With some more time it may have been best to use another matching algorithm using an adjacency list between the slots remaining and the lab, tutor pairs remaining as shown in Figure 3 once the lectures had already been assigned. This would allow a similar algorithm to be used as for the matching of tutors to lectures and labs and may have allowed for a more optimal and faster solution with proper ordering such as the fail heuristic used in the tutor, module matching.