Apijet: A Cgi/A* Based TA Allocation System

Thomas Lin and **Terry Koo**
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
410 Memorial Drive, Cambridge, MA 02139
{tlin, maestro}@mit.edu

ABSTRACT

Apijet is an AI application developed to optimally pair teaching assistants and courses at MIT. Apijet uses a web interface for human interaction and an A* search to create optimal pairings. Teaching assistants and professors sign in to Apijet online and enter their preferences and requirements concerning teaching. A search can be run with the click of a mouse. In creating pairings, Apijet uses A* to optimize the total suitability of TAs for assignments. This paper discusses the resource allocation problem and describes how Apijet works. Apijet is an example of how AI search methods can solve real world problems.

1. INTRODUCTION

After experiencing the great TA-ship of our Teaching Assistant Jimmy Lin, we felt an urge to develop a TA Allocation system so that as many other students as possible could have wonderful student-TA experiences like we are having in our MIT Artificial Intelligence (6.034) class. Apijet stands for "Allocation Program Inspired by Jimmy (lin)'s Extraordinary Teaching." This program is dedicated to Jimmy Lin, the greatest 6.034 TA we've ever had.

The creation of Apijet involved several stages. First, we had to decide on how we were going to get information from potential TAs, and professors requesting TAs, so that we could pair them together optimally. We eventually decided on writing a Common Gateway Interface (CGI) script so that TAs and professors could enter their preferences online. Then, we had to decide on a method for pairing the TAs and professors based upon the preferences they entered. We chose to implement an A* search to handle this.

This paper will discuss the general resource allocation problem, the specific application of the problem in allocating teaching assistants for classes, and also give a detailed overview of our Apijet system along with its strengths and weaknesses.

2. THE RESOURCE ALLOCATION PROBLEM

The more specific class under which our TA Allocating system falls under is the resource allocation problem.

2.1. The General Problem

The resource allocation problem is when there are limited resources to distribute and it needs to be decided how these resources will be distributed. One example of the resource allocation problem is the problem of deciding when each airplane can use the stands at an airport. The limited resources are the stands at the airport, and the stands need to be properly paired up with airplanes to ensure smooth operation of the airport.

Without a computer program, this task is something that would require a human operator with several years of experience [1]. However, with an AI computer program like HKAI's SAS, a "highly optimized allocation plan" can be "produced in reasonable time, (ensure) that all operational constraints are considered all the time, and (perform) problem solving in real-time or close to real-time." [1]

Computer programs can often be used to solve large resource allocation problems like that since they can perform very quick searches. Another example of this is the AMC barrel allocator, which can schedule 1000 airplane missions and 5000 flights "from scratch in less than 20 seconds on a Pentium II 400 MHz." [2]

2.2 The Specific Problem

One version of the resource allocation problem is the personnel allocation problem. In the personnel allocation problem, the limited resource being allocated is personnel. There are applicants with unique qualifications and preferences, and positions with different requirements and preferences, that need to be assigned to each other intelligently.

The TA Allocation problem is a local (at MIT) example of a personnel allocation problem. The applicants are Teaching Assistants (TAs) who need to be assigned to classes in the most optimal configuration.

Each semester, MIT offers hundreds of classes that need TAs. It can prove to be time-consuming and difficult for a human being to match up all the TAs with the classes since there are so many factors involved in each assignment and there are so many assignments. We felt that this was an area in which a computer program could make everybody's life easier. A computer program could do the pairings, and ensure that the best possible TAs are assigned to each class. The importance of assigning appropriate TAs to a class cannot be underestimated; oftentimes the TA assignments play a major role in influencing how well students learn the material and how happy everybody is with the course.

2.3 Why the British Museum Search Fails

One method of solving the TA allocation problem is to write a program to enumerate every TA-class combination possible, and then determine which of these combinations is the best. This method of exhaustive searching is affectionately known as the "British Museum" search. The British Museum search is not appropriate for solving the TA Allocation problem since it has an extremely high time-cost. Space costs can also be a factor if the British Museum search is done in a breadth first manner, or if all solutions are kept track of.

Let's say that there are 15 TAs applying for positions in 10 classes. If each class only wanted one TA, this would result in a search tree with branching factor 15 and depth 10 (or branching factor 10 and depth 15 if implemented differently). A search tree with branching factor 15 and depth 10 has 15^10 potential paths. 15^10 is over 576 billion, so a British Museum search would look through 576 billion possible solutions before returning a value. Implementing such a search in reasonable (for the scenario) time would require a very fast computer. Since we do not have such fast computers to work with for implementing our system, the British Museum search would take an unreasonable amount of time to complete.

If there are 15 classes requesting TAs, then the depth becomes 15 and the number of possible solutions increases many-fold, to the point where computers that can run the British Museum search in reasonable time are yet to be developed.

3 TECHNICAL IDEAS AND PRECEDENTS

In developing our program, we made choices concerning how to approach individual sub-problems of the main problem. This section describes the two main modules of our system, and how it came to be that we decided to use them.

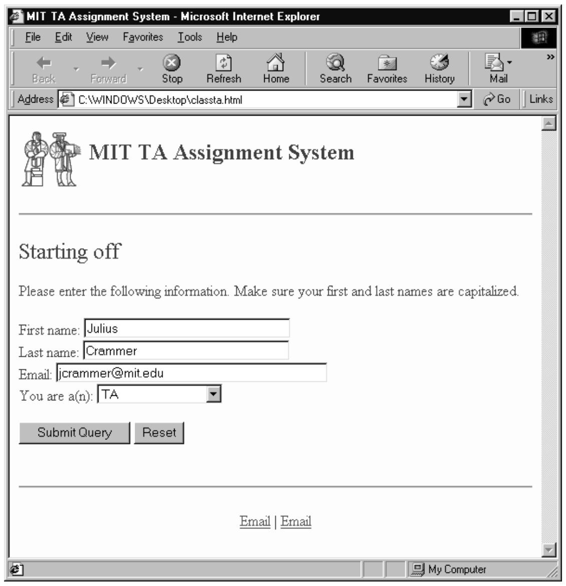
3.1 Modules Within Apijet

Apijet has two main modules: a web interface (the CGI module), and the code for solving the TA Allocation problem (the A* module). We came up for the idea for the web interface because in the real world, an easy method is needed to get information from TAs and professors. At MIT, the internet is already used for many kinds of student information processing and computer use is widely and freely available on campus. A web interface seemed like the perfect solution.

The second main module within our system is the code for optimally pairing TAs with the classes that needs TAs. The second module is implemented in MIT Scheme, and can also be run directly off the internet using CGI.

3.2 The CGI Module

The basic structure of the web interface can be divided into input and output web pages. An example of such a web page is Figure 1, which shows the "sign-in" page. The input interface allows a TA or professor to fill out an electronic survey form. This form asks TAs questions pertaining to their interests and academic record, and asks professors their requirements and preferences for a TA. The output interface allows a "Sysadmin" user to log in using a password, review the data entered thus far, and run the TA-assigning engine if desired.


MIT TA Assignment System - Microsoft Internet Explorer

Address C:\WINDOWS\Desktop\classta.html

MIT TA Assignment System

Starting off

Please enter the following information. Make sure your first and last names are capitalized.

First name: Julius
Last name: Crammer
Email: jcrammer@mit.edu
You are a(n): TA

Submit Query | Reset

Email | Email

File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Go Links
My Computer

Figure 1. The "sign-in" page for Apijet.

The web pages are not static, but designed on the fly, using the CGI protocol. Briefly, CGI is a system that allows dynamic generation of web pages through the execution of server-side programs. Although the common choice for CGI is Perl, any language can be used to create CGI programs; the only requirement is that the program, when executed, outputs a web page to stdout. When a client makes a request to view a CGI file, the server, instead of transferring the CGI file itself, executes the CGI file and redirects the output to the client.

Apijet's CGI programs were written from scratch in a variant of Scheme known as RScheme. The reason for this decision was mostly pragmatic; RScheme provides the ability to compile Scheme code into a binary executable without which the usage of Scheme for CGI would have been made much more difficult. Barring server failure, the web-based interface is viewable at:

http://boot.res.cmu.edu/~terrykoo/cgi-bin/classta.cgi

(and also linked to from http://ta.thomaslin.com)

Note that the current web-based interface is compatible only with the Internet Explorer browser.

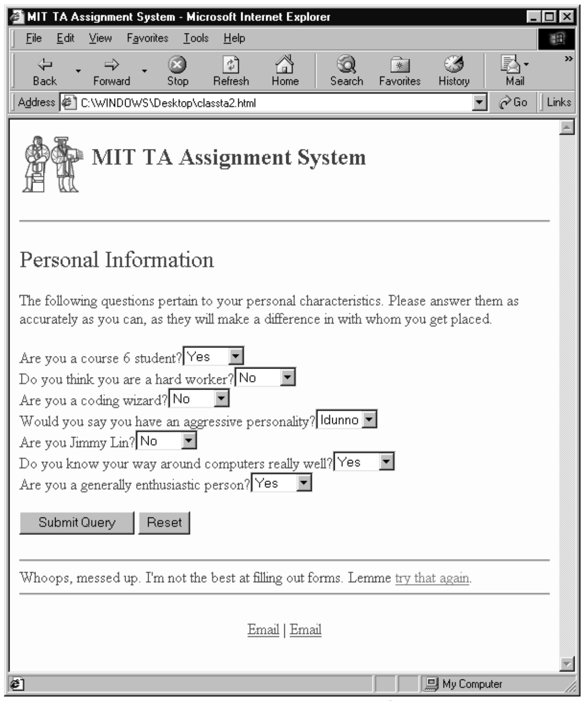
3.2.1 Usage of The Web-Based Interface

Ideally, a set date for a run of the assignment program would be made public and during the intervening time, TAs and professors would log into the web page and submit their information forms. When they finish entering data, each TA or professor's information is processed into the format of an assertion-line (i.e. a line of *list-of-assertions*) and saved into a .scm (Scheme) file at a special location on the server. Later, at the indicated time, the Sysadmin will log in and run the matching program; the matching program loads in all the saved .scm files and combines them into a list-of-assertions, which the path-length calculator and search program are executed with.

3.2.1 Precedents for the Web-Based Interface

At MIT, the 6.034 Artificial Intelligence class has recently started using an online web tutor to help students learn the material in the course. This web tutor is interactive. It provides the students with problems to solve, lecture material, and in some cases, also hints on how to solve specific problems.

The 6.034 online tutor served as a precedent for the Apijet system since it showed us that online interaction could be made fairly painless. Another system implemented at MIT which is similar to Apijet is the MIT Student Information Services site, which runs physical education and housing lotteries online for students. Students sign in and enter their information, and a search is conducted at a later date.


MIT TA Assignment System - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Address C:\WINDOWS\Desktop\classta2.html
Go
Links
My Computer

MIT TA Assignment System

Personal Information

The following questions pertain to your personal characteristics. Please answer them as accurately as you can, as they will make a difference in with whom you get placed.

Are you a course 6 student? Yes
Do you think you are a hard worker? No
Are you a coding wizard? No
Would you say you have an aggressive personality? Idunno
Are you Jimmy Lin? No
Do you know your way around computers really well? Yes
Are you a generally enthusiastic person? Yes

Submit Query | Reset

Whoops, messed up. I'm not the best at filling out forms. Lemme try that again.

Email | Email

Figure 2. The "TA personality" page for Apijet

3.2.2 Technical Choices

There were several main points that led us to choose a CGI implementation for the user-interaction part of Apijet.

One potential real-world problem in dealing with resource allocation problems is having only incomplete or vague information about the resources. We felt that by using an internet interface, we could ensure that the system had a complete set of information about each of the TAs, since the TA forms are not submitted until the TA has filled out all the necessary information.

Also, another reason to use the internet interface is that it allows us to "standardize" some of the TA personality traits. For example, if we had just asked TAs to describe their personality, the system would have received a wide range of input (i.e. "I love rabbits," "I am very smart," or "I am a member of Mensa") Perhaps the system would not be able to recognize all the terms the TAs use to describe themselves. However, by using a web interface we can restrict the personality traits on which a TA can comment on.

We restricted the personality traits to a few we felt were most important for the situation. Currently, the system will ask the TA whether they are a course 6 student, if they are a hard worker, if they are a coding wizard, whether they have an aggressive personality, whether they are Jimmy Lin, whether they know their way around computers very well, and whether they are generally enthusiastic. Then, the program will ask professors which ones of those specific traits they require or prefer their TAs to have. Figure 2 shows our "TA Personality Selection" page.

At first, we were considering implementing a Java interface since there are several programs (primarily Skij and Kawa) which can run both Scheme and Java. However, we later decided that CGI would be a better choice.

A CGI interface saves the user some loading time, and allows for the easy use of html input forms. Also, CGI often uses fewer resources on the end-user's computer than Java.

3.3 The A* Module

The A* Module runs an A* search. In the A* search that we used, if there are more TAs than classes, then individual nodes of the search are TAs. A TA assignment at the first level of path length is equivalent to assigning that TA to the first class. A TA assignment at the second level of path length is equivalent to assigning that TA to the second class. The search terminates after the path length has reached the number of classes (when every class has been assigned a TA). When there are more assignments than TAs, it uses assignments as the nodes and TAs as the depth levels to continue ensuring optimal results.

The sample path (Jack Melissa Alyssa S) is equivalent to the TA assignment of Alyssa to the first class (i.e. 6AAA), Melissa to the second class (i.e. 6BBB), and Jack to the third class (i.e. 6CCC). The specific ordering of the classes is determined within the program. The search tree generated in this problem is similar to what would be generated by the map (with TAs as nodes) shown in Figure 3. The point of the search is to generate the shortest possible path (in terms of the path-length function) that contains the number of nodes equal to the number of TAs.

There are three main functions (each with their own sub-functions) within the A* program. The first function takes the description of TAs and subjects and turns it into a list containing numerical assignments for how well suited each TA will be for each class. The second function takes this list and sorts it by standard deviation. The third function takes this sorted list, makes a decision about what search method to run based on the size and standard deviations of the list, and then runs an appropriate kind of search to pair the TAs with the subjects. The A* module outputs the list of TAs paired with subjects using these three functions.

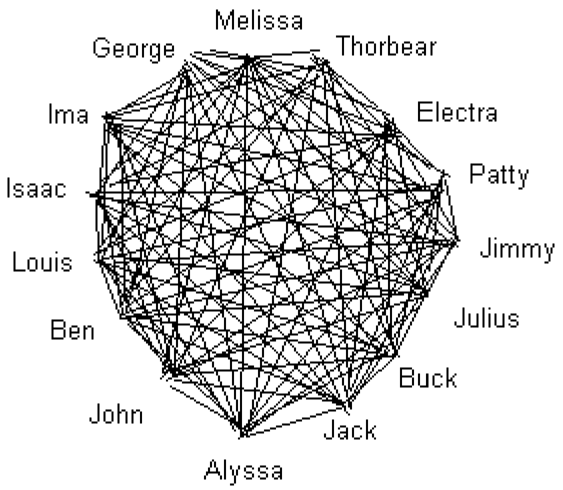


Figure 3. Map-equivalent representation of the problem

3.3.1 Converting and Sorting the Assertion List

Professor and TA input preferences/requirements into the first module of the system, and all this information is then converted into a table of path-length values for our A* search. Currently the suitability of a TA for an assignment is calculated as a value from 0 (worst) to 100 (best). It is calculated using the following formula.

- 62 points for meeting a professor's requirements, further broken down into (45% for meeting the specific grade requirements, 5% for meeting the grade preferences, 45% for meeting the personality requirements, and 5% for meeting the personality preferences.)
- 26 points for meeting a TA's preferences
- 12 points for how well the student has performed in the class in the past.

Path length is 100 minus the suitability value, which gives low-suitability pairings a higher path length. The formula we used is based on what we thought would yield a good numerical reflection of the quality of a TA-class pairing. Notice that the way the suitability formula is currently set up, professor preferences cannot out-weight professor requirements unless the professor states 10 personality or grade requirements. Our program currently does not allow the professor to set 10 personality requirements (since they can only choose from 7), and it is highly unlikely that a professor will require certain grades in 10 prior classes before accepting someone as a TA.

The second function sorts the path-length table by standard deviation. This result is then reversed, so that classes with the lowest standard deviations of TA suitability numbers (a class has a low standard deviation if most of the TAs have similar suitability numbers) are looked at first and classes with the highest standard deviations for TA suitability numbers are looked at last. We found that organizing the table in this manner generally yielded the quickest searches since the program will know early on using the distance-to-goal measurement, whether assigning a TA to a class with low standard deviation is a bad idea since the TA could be better suited for another class later on the list.

*3.3.2 Searching the table using A**

The point of our A* search is to minimize the total path length and thus maximize our "TA suitability points" and yield the absolute most desirable assignment state based on what is in the assertions database. The admissible heuristic/distance to goal at any given node is an underestimate of the minimum path-length required to reach the goal from that node using nodes that haven't been assigned yet.

A rule which A* inherently does with our model is assigning each TA to one class at most. It will not assign a TA to two classes, since that would mean visiting the same node (TA) twice in a given path down a search tree.

3.3.3 Supporting Multiple TAs

The program can accommodate professors seeking multiple TAs. After every class requesting a TA has been assigned a TA the program will go back and edit the assertions list appropriately, removing any TAs that have already been assigned and removing any classes that have been assigned the number of TAs they requested. Then, the search is run again with the new set of assertions. This is repeated until the assertions list contains either no more TAs to assign, or no more classes requesting TAs.

We felt that assigning every class at least one TA before assigning any classes more than one TA would be the fairest arrangement. We would not want a situation where a professor complained that his class was assigned no TAs while another class was assigned two TAs.

3.3.4 Precedents for the Search

The particular problem which was a useful precedent for our system was the problem of getting from one point on a map to another point. We felt that this problem was a good reflection of how a computer search is able to get from one state (the start state) to another (the finish state). So, we translated our resource allocation problem into a problem that would be similar to the map-path problem. The map-path problem has been demonstrated to be most effectively be solved by the A* search method.

From our 6.034 Artificial Intelligence class, we learned several ideas which proved to be useful in approaching the search part of the TA Allocation problem. Specifically, we learned about how effective searches were in comparison to each other, and about the many benefits of an A* search.

An A* search generates optimal paths. An A* search with a good distance-to-goal underestimate is able to search in the right direction toward the goal.

We chose A* over such searches as Best-First and Depth-First since we were looking to optimize the TA-Class pairings, and A* is an optimal search while Best-First and Depth-First are not. It is very important to create optimal or near-optimal pairings in this problem, since good results are often crucial towards the success and happiness of students, TAs, and professors over entire semesters. We chose A* over the Branch and Bound optimal search because A* employs a distance-to-goal measure which ensures that the program is searching in the right "direction" towards the goal.

3.3.5 Technical Choices for the Search

The most difficult technical choice for the search was deciding which kind of search to use in the first place. We could have either approached the problem as a Constraint Satisfaction Problem (CSP), or as a general search problem.

We chose to approach the problem as a general search problem as opposed to a constraint satisfaction problem because a large part of pairing TAs with classes/professors is matching preferences. If you implement a rule-based system with constraints, then you will often run into data which cannot satisfy all constraints. In that case, you will need to gradually relax the constraints. We felt that approaching the problem as a general search problem solved that problem since when we view this problem as a general search problem, we are guaranteed a solution and never have to go back and change constraints.

The A* method which we used works very similarly to methods used in solving constraint satisfaction problems.

- The A* method implements the rule that no TA can be assigned twice naturally, since that would mean visiting a single node twice in the search.
- If our TA-suitability function is modified so that only professor requirements count towards the suitability and meeting all the requirements yields a suitability of 1 while not meeting all the requirements yields a suitability of 0, then our search essentially becomes a straight CSP search where to goal is to find an answer without violating any of the requirements.
- By adding the TA suitability calculator, what the program does is take not only requirements, but also preferences into account.
- Forward Checking in CSP search checks to make sure that all constraints can still be satisfied from a given position during the search. Our distance-to-goal measurement does a similar thing, by giving positions that violate requirements a high distance-to-goal measurement which ensures that they won't be considered until all other options have been exhausted.

4 RESULTS AND REMINISCENCES

4.1 Does the Program Produce Optimal Results?

Our program produces optimal TA/class assignments for our path length function. Our path length function tells us how desirable it is to assign a TA to a class. The desirability is rated from 0 to 100, with 0 meaning least desirable and 100 meaning most desirable. The program then conducts a search to pair the TAs and classes in a way such that the total desirability is maximized. Our path length function, however, has not been scientifically proven to yield exact "desirability," which is in itself a subjective term.

4.2 How does running time increase with problem size?

As A* is an optimal search, running time increases exponentially with problem size. However, while the worst case run time is exponential, the best case run time is linear. The path-length and distance-to-goal functions in A* are used to try to goad the run time towards the best case run time, and away from the worst case run time.

So far we have been getting very positive results (sometimes 0.1 second search results) based on data similar to sample data sets provided for us by faculty of our Artificial Intelligence class. A* is likely to take a long time if many TAs are assigned the same path-length in the same classes, or if some TA is best suited for every class (in which case the distance-to-goal underestimate is less effective).

The program is currently not optimized to deal with cases where it needs to assign an extremely large number of TAs. For example, if the program was used for pairing every TA at MIT, it would take a long time to run. Until the program is optimized to support such large data sets, we suggest that it be used in assigning TAs in smaller data sets. For instance, it could be used to assign TAs who want to teach low-level math classes, or TAs who want to teach a certain subset of classes in the Electrical Engineering and Computer Science Department.

4.3 How is running time consumed?

Most of the run time is consumed in searching the list. The searching procedure is the only one that is exponential time in the worst case scenario. If you have x (where $x > y$) TAs and y classes requesting TAs, then the worst case search time is approximately a linear (depending on how many TAs each class requests) multiple of x ^ y. The program is currently set to display progress (in A* terms, "paths expanded") in the search as it is searching. If this function were turned off, some search time could be conserved.

5. FUTURE WORK

There are several additional features that may be coded into future versions of Apijet. For example,

- Apijet can currently understand if a TA says that they are "hard-working" and the professor requests someone "hard-working." However, Apijet will not know not to assign a "lazy" TA to a professor requesting a "hard-working" TA. In essence, if the TA enters that they are "lazy," it has the same effect as if the TA had chosen that they weren't "lazy" or "hard-working." The ability to distinguish that may be included in future versions.

- Also, Apijet currently runs the same search method (A*) on all data sets. In future versions, Apijet may run different searches on the data depending upon the data size and the standard deviation values. For example, if the future version of Apijet is asked to pair together all of MIT, it might run a best-first search on the data instead of an A* search so that it can accomplish the search in a reasonable amount of time. Or, it might seek probably approximately correct (PAC) results. [3]

6. CONTRIBUTIONS

This paper discusses the resource allocation problem and the TA allocation problem. Then it discusses Apijet, which is an implemented system for solving the TA Allocation problem. It describes how we came up with the methods used in Apijet, and how we implemented the methods. We hope that the methods used in constructing Apijet can also be of use in constructing other similar applications.

REFERENCES

1. Chun, A.H.W., Chan, S.H.C., Tsang, F.M.F, and Yeung, D.W.M., HKAI SAS: A Constraint-Based Airport Stand Allocation System Developed with Software Components. *Eleventh Conference on Innovative Applications of Artificial Intelligence,* Orlando, Florida, 1999
2. Becker, M.A., and Smith, S.F., Mixed-Initiative Resource Management: The AMC Barrel Allocator. *Fifth International Conference on Artificial Intelligence Planning Systems*, Breckenridge, CO, 2000
3. Gratch, J, Chien, S., and DeJong, G. Improving Learning Performance Through Rational Resource Allocation. *Eleventh National Conference on Artificial Intelligence*, Seattle, Washington, 1994