

Coordination of Student Project Allocation

Dimitar Kazakov
Computer Science Department
University of York

October 31, 2001

1 Introduction

This paper presents an overview of the process of final-year project allocation in the Computer Science (CS) Department of the University of York, the previously employed approaches to the coordination and automation of the process, and, finally, a new solution which has been designed and implemented to alleviate some of the existing problems. The document can be used to provide a better understanding of the issues involved in the coordination of project allocation, either in order to use the existing systems or to implement one on one's own. Also, all parties involved can benefit from a better understanding of the responsibilities of and constraints on the other participants in the process.

The author's involvement in the student project allocation started as he volunteered to stand in for the then ill project coordinator, an appointment, which soon had to be made permanent. After two years of experience with the inherited software used in the allocation, the author went clearly beyond the call of duty, and proposed to re-design the system, and supervise its reimplementation. The proposal was accepted, and an undergraduate three-month internship funded to support it. The actual implementation was carried out between July and and September 2001, but many of the ideas used had fostered in the author's mind for much longer.

Most of the undergraduate courses taught in the CS Department include a requirement to undertake a project in their final year. For some courses (Mathematics and Computer Science, MMath) the project is optional. Students may choose to have two projects, one in each of the third and fourth year, if they are on the MMath course, or they may be required to do so, if enrolled in the Computer Science and Software Engineering (MEng) course [Eva01].

The total number of students eligible for a project has rapidly risen in the past few years [Fis], and is well above 150 at present. Each year, all eligible members of staff are asked to submit a number of such proposals, which, after a process of peer reviewing and corrections, are brought to students' attention. It is in the students' best interest that they should be offered a wide range of project proposals, from which they can choose the one that is appropriate for

their course, and is of their liking. As a result, supervisors are encouraged to submit more proposals than their expected supervision loading of 7–8 students. The process of assigning projects to students has to meet several requirements: (1) assign students to projects which meet their course requirements, (2) give all students a fair chance to apply for a project of their choice, (3) balance supervisors’ loading, and, (4) allow supervisors to rank students and/or projects. Additional issues, such as data protection, must also be taken into account.

The final-year projects are worth one-third of all credits in the year, which, along with the weight of 5/9 (resp. 7/16) that the third (resp. fourth) year has in the formation of the average mark for the entire course, makes the suitable allocation of these projects a matter of prime importance for the students’ overall success in the course. Moreover, the students’ learning experience is clearly influenced by the right choice of project, which has to attract the students’ interest, and be suitable with respect to the options they have taken, and their performance in the subjects related to the project area. Finally, as the supervisor’s rôle also includes dealing with other academic and personal issues, the choice of project has implications that go beyond project supervision to influence all aspects of the supervisee’s academic life and performance.

There are also other project allocation issues which, although not directly related to the learning process, can influence the students’ learning experience through the amount of additional workload they put on the academic and secretarial staff in the department. Among these issues are the time required to prepare and submit a set of project proposals, the ability efficiently to coordinate the—potentially iterative—process of peer reviewing and subsequent updates, and the ease with which all project proposals are combined and distributed to students.

2 History of Project Allocation Approaches

In this section, the two known approaches used in the CS Department for the coordination of project allocation until academic year 1999/2000 will be described and their relative merits discussed.

2.1 The pre-1998/99 case

Until 1997/98, projects were allocated in a simple process in which members of staff submitted their proposals as a computer file of “*any conceivable format*” [Cou01] to one of the Departmental secretaries, whose responsibility was to compile all proposals into a single document, and distribute a printed copy of it to all students eligible for a project. There was a designated period of time later in the year, when students could discuss projects with supervisors. At the end of this period, students were issued with a form by the signing of which a supervisor would confirm his/her agreement to supervise the given student on the project named in the form [Run01]. This process required both student

and supervisor to be present when the form was signed, and was irreversible, as neither part could withdraw from that written agreement.

The pros and cons of this approach could be summarised in the following way:

- ⊕ Student and supervisor are guaranteed to meet and discuss the project, so no random allocation is possible.
- ⊕ Knowledge of pre-requisites can be checked and enforced by supervisor.
- ⊕ Supervisors can submit proposals in their preferred word-processor format.
- ⊖ Supervisors cannot meet all interested students before making a commitment, as it is impractical to ask all students to come to each supervisor twice. Therefore, allocation is done on the “first-come, first-served” basis. This is particularly problematic when the allocation period starts at a different time for students on different courses.¹
- ⊖ Once proposals are submitted and published, there is no easy way to inform students about any changes in them. This can be a problem in a situation when a proposal requiring collaboration with non-academic entities loses their support and has to be withdrawn.
- ⊖ The processing of project proposals in various formats puts a lot of strain on the secretarial staff.

2.2 1998/99 and 1999/2000: A WWW-based allocation database

As the number of students requiring projects doubled between the academic years 1997/98 and 1998/99, a step was taken to move from old, paper-based system of project allocation to one based on Web technology. The motivation behind the design and implementation of the resulting project allocation database are described in the report of A. Fisher [Fis], who single-handedly approached that difficult task.

2.2.1 Design Outline

The system is composed of three principal components:

1. A set of static Web pages on which each supervisor publishes the list of his/her project proposals. There is one page per supervisor, who is responsible for its maintenance.
2. A plain-text database with a front-end interface based on dynamic HTML. The interface is used by the project coordinator to add projects and students to the database. It is also used by students to select and rank the

¹There was such precedent involving the MScIP course.

projects of their choice. Finally, supervisors use it to see the list of students who have expressed interest in each of their projects, and rank those students according to their preference.

3. A project allocation procedure based on the *stable marriages* algorithm, which is implemented in a mixture of C and Awk, and runs on the Departmental Web server.

The screenshot shows a web browser window with the URL <http://www.cs.york.ac.uk/projects/pad/addproj.html>. The page title is "Add a vetted project to database".

Form fields:

- Username:
- Project Code:
- Title:

Course selection table:

Course	
BEng/BSc CS	<input checked="" type="checkbox"/>
BEng/BSc CS Sandwich	<input type="checkbox"/>
BA Info Eng Sandwich	<input type="checkbox"/>
MEng PR3	<input type="checkbox"/>
BSc CS/Maths	<input checked="" type="checkbox"/>
BSc CS/Maths Sandwich	<input type="checkbox"/>
BA/BSc ITBML	<input type="checkbox"/>
BA/BSc ITBML Sandwich	<input type="checkbox"/>
MMath PR3	<input type="checkbox"/>
MEng PR4	<input type="checkbox"/>
MMath PR4	<input type="checkbox"/>
MSc IP	<input checked="" type="checkbox"/>
AdvMSc SCSE	<input type="checkbox"/>
AdvMSc SWE	<input type="checkbox"/>

Buttons:

Figure 1: Interface for adding projects

A closer look shows that as supervisors maintain the Web pages of their own proposals, it is easy to maintain and update them; on the other hand, no changes in these pages can be easily enforced, and they are likely to be in various formats. The database stores only a minimum of information about each proposal: project code and title, supervisor, and a numerical key encoding the range of degree courses for which the project is suitable. Figure 1 shows the database interface used to add a project and one possible entry. The degree course key mentioned is automatically computed according to the boxes ticked in the table at the bottom of the form, so, for instance, the combination of CS, CS/Math and MScIP courses is stored as 0049. In theory, it is possible to edit existing project entries (see Figure 2), however, the degree course key would have to be computed by hand and entered directly, as a string of digits. Of course, one can also delete the project altogether and add it again using the form in Figure 1.

There is another form that can be used by the Project Coordinator (PC) to add students (see Figure 3), but, in fact, the vast majority of students' entries are extracted automatically from the departmental database 'sv'. The form in Figure 4 serves the supervisors to rank the students who have selected

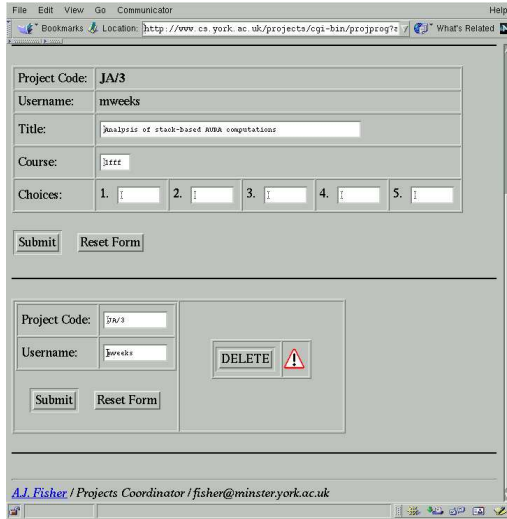


Figure 2: Interface for editing projects

one of their projects; a similar form is used by students to rank up to five projects of their choice. Both students and supervisors can change their selection throughout the allocation process.

2.2.2 Stable Marriages Algorithm

The third component of the project allocation system, the allocation procedure requires a more detailed explanation. It is based, as mentioned, on the Stable Marriages (SM) algorithm [Knu97]. The algorithm itself will not be discussed here, but the basic idea will be illustrated, along with the main difficulties related to its use.

The task of the SM algorithm is to match members from two different lists (men and women, students and projects, etc.) according to the preferences expressed by each of the lists' members. In its simplest form, the algorithm considers two lists of equal size, and each member of a given list provides explicit ranking of all members of the opposite list. This setup is shown in Figure 5.

A couple (X, Y) forms a stable assignment (or "marriage"), if neither X nor Y has ranked more highly another partner, who would rather be with X (or Y) than with his/her present partner. The algorithm halt condition requires that the above is valid for all couples. Here is the same statement in semi-logic notation:

$$\forall(X, Y), (\exists(W, Z), X \text{ ranks } Z \text{ above } Y \text{ and } Z \text{ ranks } X \text{ above } W) \text{ and } (\exists(Z, W), Y \text{ ranks } Z \text{ above } X \text{ and } Z \text{ ranks } Y \text{ above } W) \quad (1)$$

The cases in Figure 5 illustrate the principle. Case 1 represents a set of sta-

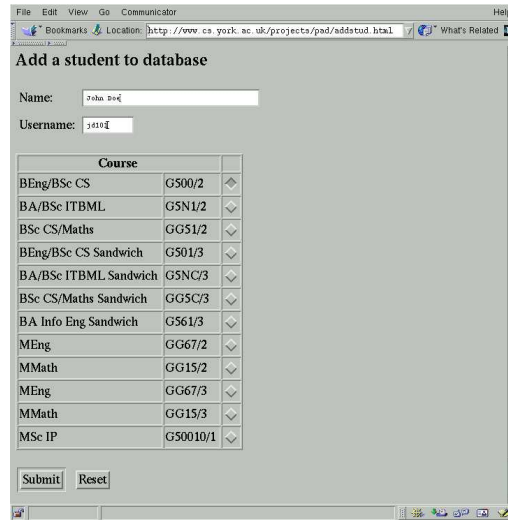


Figure 3: Interface for adding students

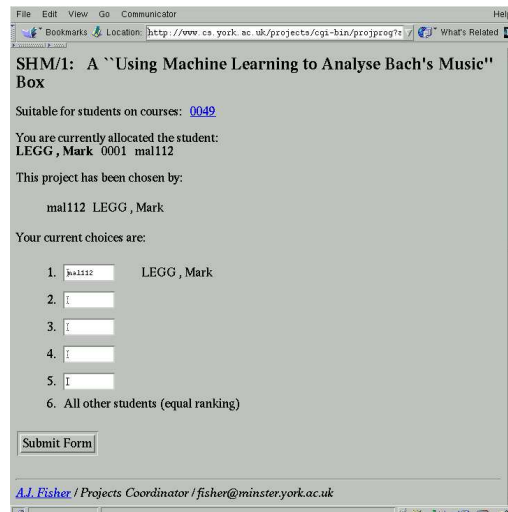


Figure 4: Interface for ranking students

ble marriages—it will be left to the attentive reader to verify this statement. In Case 2, Student 3 has swapped the ranks of Project 2 and Project 3, yet the overall matching remains unchanged. The student is attempting to have Project 3 assigned. However, his current assignment remains stable as (the supervisor of) Project 3 has ranked Student 3 below its current match. Swapping the ranks of Student 2 and Student 3 in the list of Project 3, creates the necessary condition for a change in the overall matching, as there is now a potential couple, (Student 3, Project 3), in which both members would prefer to be matched with each other, rather than with their current partners. The new stable matching is shown in Case 3.

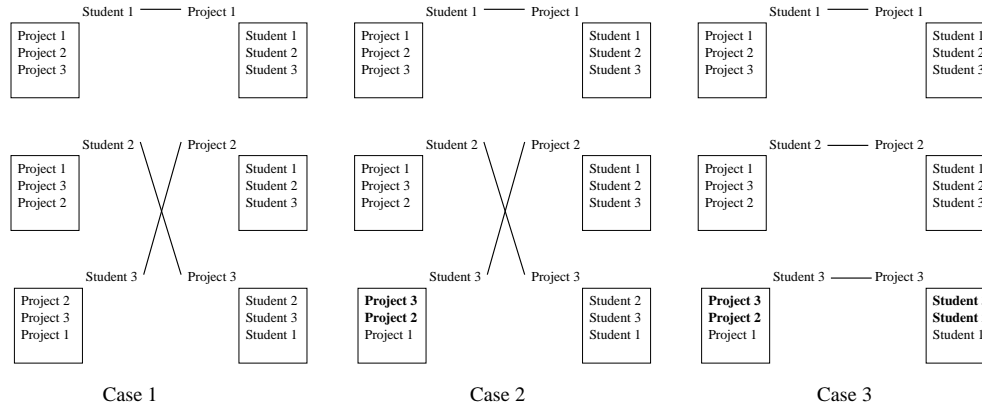


Figure 5: Stable marriages principle

A comparison between the above SM assignment problem and the needs of the project allocation setup show several additional requirements that the latter brings in.

1. *In practice, there are more projects than students.* This issue is easily dealt with by adding “dummy” entries to the list of students, so that its length is equal to the one of the project list.
2. *It is impractical to request students to provide an explicit ranking for all 200+ project proposals.* The solution provided is to keep the explicitly ranked projects at the top of their list, and to rank below all remaining projects according to some appropriate criteria.
3. *The algorithm should be able to cope with cases in which a supervisor or student has not made any selections.* The solution is as above.
4. *Not all projects are suitable to all degree courses.* One of the above mentioned “appropriate criteria” is to rank unsuitable projects at the very bottom of the student’s list.

Table 1: 1998/99 and 1999/2000 project allocation procedure

Allocation stage	Coordinator	Supervisors	Students
Call for project proposals	Receives proposals' URLs, and confirmations about vetted proposals	Submit proposals, perform peer reviewing and vet proposals	Not involved
Project selection & allocation	Publishes a Web page with links to all vetted proposals; adds projects and students to database, and opens it to supervisors and students	For each project, rank students who have selected it	Rank up to five projects
Final project allocation	Closes database; publishes final allocation	Not involved	Not involved

2.2.3 Timing

The timing of the whole process is displayed in Table 1. When the database is initially opened, it displays a random matching. Each time a student or supervisor edits his/her entry, the SM algorithm is executed and the new allocation displayed. Once the database is closed, the allocation becomes final, and is published. In practice, this was followed by another step, which is not shown in the table. As the SM algorithm used did not guarantee even loading of all supervisors, some of the allocations were usually renegotiated between supervisors and PC, and changes in the allocation made by hand. This process required the PC's personal involvement in sometimes lengthy negotiations, and was not helped by the fact that the traditional closing date for selections coincided with the beginning of Easter holidays.

3 Analysis of the WWW-based Database

By the beginning of academic year 2000/01 it was clear that the Web-based allocation database implemented by Tony Fisher, despite being a big step forward, had several shortcomings,² which needed to be addressed. Here is a list of comments compiled by the author over time, and eventually presented to and discussed at the Board of Studies in Spring 2001 [Kaz01]:

Poor GUI and confusing dialog For instance, when a supervisor makes her choices, it is not immediately clear that the `Submit` button has to be

²He would certainly have found the best way to resolve these, had he only had the chance.

pressed *separately* for the selections made *for each* of the projects. (There was a precedent during the 1999/2000 allocation when all selections but one were not submitted properly for that reason.)

Too centralised data input PC has to type in the data and make all changes. That represents a serious overhead (around 250 proposals), and leads to delays when information has to be updated. A typical scenario requiring changes in the DB is when a supervisor is convinced by a student to relax the “course” constraint, so that the student could select the project without receiving a warning, which would be issued otherwise by the system.

Better indexing required Ideally, one should be able to index and search the project proposals by course, supervisor and topic. At present, students are forced to visit the Web pages of all supervisors in the search of a project. Moreover, a detailed account of the number of projects suitable to each degree course would be helpful, as no fair allocation is possible if there is not enough project proposals for each of the courses taught.

Project withdrawal Health reasons or change of mind of an industrial partner may require the proposal withdrawal in the middle of the allocation process. The system does not feature a standard way of informing the students who have selected the project about that change. This is a handicap for these students, as it reduces the number of their choices.

Poor balance of supervisors’ loading Supervisors are encouraged to submit more proposals to ensure better variety of topics. However, the existing algorithm does not ensure that project loading is evenly spread. In last year’s project allocation process, there were cases of supervisors receiving up to 200% the average load of projects.

Student-defined projects have to be treated in a special way. Instead of asking both supervisor and student to select the project as their first choice in order to ensure the match, as it is at present, a special case should be made for this type of projects, allowing the supervisor to earmark the project as student-defined, and directly provide the student’s name.

As this type of project proposals are very time-consuming for both sides involved, I also suggest that a student-defined proposal should be binding for both sides involved, and avoid this year’s precedent, when a student had a self-defined project as one of his less favourite choices.

Insufficient openness One could discuss whether the selection and ranking of one side (e.g., students) should not be made available to the other (e.g., supervisors) to facilitate their choice. Indeed, a supervisor has made the point of saying that one might prefer selecting a student who has made the project their first choice, as it is a good proof of motivation.

At the beginning of Year 2000/01, in an attempt to concentrate more effort on the project allocation, the author agreed to provide active support in the

form of both data and discussions to Jonathan Dye, an undergraduate student, who chose the task as a topic of his final-year project [Dye01]. The student made some interesting comments about the existing system, but he decided to concentrate on the properties of the SM algorithm. The experiments carried out provided an interesting theoretical insight, but were of little immediate practical importance, apart from the fact that finding *all matchings* that satisfy the stable marriage requirements is an intractable task for a data set of the size usually used in the Department.

In his paper, Jon Dye mentions two issues, which this author identified, and discussed with the student at length.

Popular projects not being allocated Some of the most popular project proposals are sometimes not allocated at all, despite the long list of students who have selected them. There are both psychological and technological factors involved in this phenomenon. On one hand, students may not rank a popular project as their first choice, even if that is the case. The line of reasoning goes as follows: *“it is unlikely that I will get this project, and if I don’t, I won’t get allocated even my second choice, as someone else may have ranked it on top of their list, so I’d better play it safe and select as first what is really my second choice...”*.

In Tony Fisher’s implementation of the SM algorithm, the most popular projects are ranked at the bottom of a student’s list, if not explicitly selected by that student. The reason for this choice was to prevent students who have not made any selections from randomly ranking as their top projects the ones, for which there is already a lot of competition [Fis].³

Random allocations A student who has not selected any project, and who has not been selected by any supervisor, would obviously end up with a randomly allocated project. This is a positive feature of the system, as it ensures that students who need a project would be assigned one. However, there is another situation in which random allocation is more difficult to justify. If all explicit choices of a student correspond to projects, which are in great demand, it is possible that all those projects would be allocated to other students, so that the student in question would be left with a randomly allocated project.

4 2000/01: A Two-round Project Allocation

The code of the legacy system that the author had in hand contained almost no comments, and was quite idiosyncratic at places, which made it very difficult to modify.⁴ Nevertheless, some room for improvement within the existing system

³Another way of looking at the system is that it penalises supervisors submitting uninteresting projects by giving those supervisors a higher chance to be assigned random students!

⁴Indeed, facing the impossibility to recompile the source code due to a missing library, a text entry in one of the dynamically generated HTML forms had to be modified by searching the compiled binary file, and editing its string table.

was identified and explored in the project allocation in academic year 2000/01. Here is the account of the changes made at this stage as presented in May 2001 [Kaz01].

Two rounds of project allocation In the 1999/2000 project allocation, there was a serious disbalance in the project loading per supervisor. As a result, several projects, and the students assigned to those projects, had to change supervisor. Since the choice of project may also be guided by the supervisor's personality or special skills, it was considered a better idea to give those students a second chance to select projects designed by the remaining underloaded supervisors in a second round of project allocation (see Table 2). This method was used this year with several positive results:

- ⊕ The majority of the students—around 65%— had a firm allocation by the end of the first round, and could start working on their literature review.
- ⊕ An earlier closing date for the first round meant that MScIP students could discuss alternative projects with supervisors in person, before the beginning of Easter holidays.
- ⊕ Staff who had been allocated the required number of projects in the first round were withdrawn from the database (along with their proposals), making sure that submitting more proposals is not penalised.
- ⊕ Another side effect of the two-round allocation was expected effectively to discourage staff from submitting less than the minimum number of proposals in the first round, as they had to go to the second round. Not only would they have to deal again with students at their doors actively asking for projects, but their maximum loading in this round would not be guaranteed anymore.

No random allocations Last year, random allocation was used as a solution of last resort despite the considerable amount of time and effort of several people involved. This year, contact was sought with all of the few (about 10) students, which were left with randomly allocated projects after the second round. As a result, there will be virtually no random project allocations.

AdvMSc incorporated into the system Separate treatment of undergraduate and AdvMSc projects in 1999/2000 caused problems with submitting the same proposal to the two groups of students, and counting and rebalancing the project load. This has (presumably) been solved by integrating the project allocation process for both UG and AdvMSc students. (Bioinformatics projects are still dealt with separately.)

Adding a second stage to the allocation process aimed to address two of the main sources of discontent, random allocations and disproportionate supervisors' loading. The two problems were interrelated. If a student chose

Table 2: 2000/01 project allocation procedure

Allocation stage	Coordinator	Supervisors	Students
Call for project proposals	Receives proposals' URLs, and confirmations about vetted proposals	Submit proposals, perform peer reviewing and vet proposals	Not involved
First round of project selection and allocation	Publishes a Web page with links to all vetted proposals; adds projects and students to database, and opens it to supervisors and students	For each project, rank students who have selected it	Rank up to five projects
End of first round	Closes database; publishes final allocation for all non-random allocations	Not involved	Not involved
Second round of project selection and allocation	Publishes lists of unallocated projects and students left w/o project	Rank students	Rank up to five projects
End of second round	Closes database; publishes final allocation for all non-random allocations	Not involved	Not involved
Manual allocation of remaining students	Mediates contacts	Underloaded supervisors discuss projects with the remaining students; supervisors of those students mediate contacts (or select on their behalf)	Unallocated students discuss projects with underloaded supervisors
Beginning of Autumn term	Publishes final allocation	Not involved	Not involved

projects with supervisors, who had submitted more proposals than their nominal loading, and whose projects proved popular, than it was likely that some of these supervisors would end up overloaded. At this stage, there were three options available: keep the student, assign him/her to another project or pass the project and the student allocated to that project to another, unloaded, supervisor.

There had been precedents of the latter option in the past. The argument on which this practice was based was that the characteristics and complexity of any final-year project were such that it could be supervised any member of staff, as it was general guidance rather than deep insight that was required from them. However, this position had lost ground since, and the aim was to provide students with a better chance to choose both project and supervisor.

In 1999/2000, when one-round allocation was used, there were 11 students left with random allocations, despite having made selections. There were also 7 students who did not make any selections. The supervisors' loading was rather uneven, as Table 3 shows,⁵ In comparison, the total number of students after the two-round selection was around 10. It is difficult to provide an objective comparison of the supervisors' loading between the two years, as the same fluctuations in the project loading can occur for various reasons. For instance, the most unbalanced automatic allocation in 1999/2000 had given the supervisor approximately twice the expected load, and he was extremely unhappy about it. The same extreme case in 2000/01 also corresponded to a supervisor with a double load, yet the supervisor was happy to have the extra students. Also, this supervisor only joined the project allocation in the second round, and hence did not benefit from the protective mechanism that the two-round approach provided. If one had to judge by the number of unhappy supervisors with whom the project coordinator had to deal in each of the two years, one could clearly say that the changes had been for the better. These improvements were achieved at a considerable cost. Since the database was not designed for this type of use, list of remaining projects, students and supervisors had to be re-entered manually or tools had to be implemented specifically for this purpose.

5 2001/02: A New Project Allocation Database

As mentioned in Section 1, the project allocation database was re-designed by the author, and Jamie Hodgkinson, a third-year undergraduate student, was employed over the Summer of 2001 to implement the new system. This section will present the most important features of the new design.

5.1 Project filters and en-masse selection

The first natural improvement proposed is to provide students with more flexible ways of browsing the project proposals. Students can apply three types of filter

⁵Column 4 in this table accounts for other supervisions, which are not directly assigned by this allocation database. This is the case of supervisors 12, 30 and 32.

Table 3: Supervisor loading in 1999/2000 (before manual corrections)

Supervisor No	Num. offered	Num. taken	Num. others	Relative loading	Expected No of projects	Projects assigned
1	8	4	0	1.00	6.09	4
2	10	4	0	1.00	6.09	4
3	10	7	0	1.00	6.09	7
4	4	2	0	0.50	3.05	2
5	9	3	2	0.70	4.26	5
6	4	3	0	1.00	6.09	3
7	5	1	0	0.10	0.61	1
8	6	5	0	1.00	6.09	5
9	6	5	0	1.00	6.09	5
10	6	2	0	0.93	5.66	2
11	9	6	0	1.00	6.09	6
12	10	10	0	1.00	6.09	10
13	10	6	0	1.00	6.09	6
14	5	3	0	0.33	1.98	3
15	2	1	0	0.10	0.61	1
16	9	4	0	0.20	1.22	4
17	7	4	0	1.00	6.09	4
18	2	1	0	0.60	3.65	1
19	8	7	0	0.92	5.60	7
20	12	7	0	0.75	4.57	7
21	13	6	0	1.00	6.09	6
22	10	5	0	1.00	6.09	5
23	9	7	0	1.00	6.09	7
24	3	3	0	1.00	6.09	3
25	10	8	0	1.00	6.09	8
26	8	8	0	1.00	6.09	8
27	9	4	0	0.92	5.60	4
28	8	5	0	1.00	6.09	5
29	2	1	0	0.48	2.92	1
30	9	9	0	0.75	4.57	9
31	5	2	0	0.50	3.05	2
32	15	12	0	1.00	6.09	12

to view selectively parts of the list of proposals.

By supervisor The filter is used to view all projects of a given supervisor.

By course The filter is used to select all projects suitable to a given degree course.

By keyword(s) In the new database, supervisors are allowed to assign keywords to each of their proposals (see Figure 6).

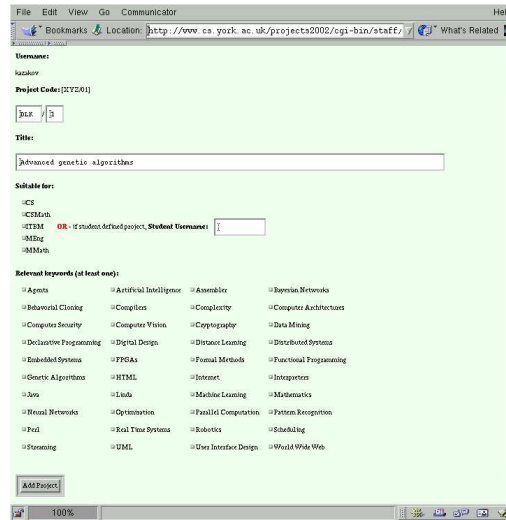


Figure 6: Interface for adding projects to the new database

The three types of filter can be combined consecutively to apply more complex constraints on the projects being displayed. Each of the selections of projects produced as a result of filtering, then can be ranked and added *en masse* to the student’s list of favourite projects, which, in turn, is automatically extended with new slots to accommodate the new entries.

This technique allows the students to rank a much greater number of projects than it would have been possible if projects had to be read and selected individually. For instance, a student can rank a few (4–5) individual projects, then add all those that are supervised by Dr J. Smith and mention the keyword “Artificial Intelligence”, followed by all other projects related to AI. A much longer list of explicit choices, of course, reduces significantly the probability of random allocation taking place.

5.2 Perfect Marriages

Another, and probably the most important, change in the way allocation is performed in the new system is based on the newly proposed notion of a *perfect*

marriage. This is simply a matching in which each of the two partners has listed the other as his/her first choice.

Many of the complaints received from students in the past two years had to deal with the fact that no allocation is certain until the database is closed. Moreover, as the allocation deadline approached, many students and supervisors, unhappy with their current allocation, were frantically changing their preferences in the hope for an improvement. All this was quite distressing for those students who were satisfied by the currently allocated project, and had to watch in despair the keen efforts of their colleagues. In addition, these last-minute changes put a lot of strain on the database, and even succeeded a few times to bring it to a halt.

This issue was addressed to an extent by the two-round allocation, as students knew they would have a second chance to select a project if they were very unsuccessful the first time. Also, the majority of students had a project assigned at the end of Round 1. However, many of them, for instance, the ones participating in perfect marriages, could have been assigned their projects immediately. This idea has been extended, and incorporated into the new database in the following way.

Initially, all students and projects are entered into the database. Unlike in the old database, adding projects is decentralised, and performed by all supervisors rather than by PC alone. Once all data is entered, the database is open for selection and allocation, which is done in three phases.

Phase 1

In this phase, students can access projects and discuss them with supervisors. Both students and supervisors can rank each other, but no allocation is performed. The phase should be of sufficient duration to provide all students with a fair chance to discuss all projects in which they are interested.

Phase 2

At the beginning of this phase, the database is examined for perfect marriages, and all such matchings are made permanent. Projects/students which/who have entered a perfect marriage, are removed from the list of choices of all other students/supervisors. This step may result in new selections popping up to the top of list of choices. Subsequently, new perfect marriages can appear. An additional condition for the creation of a perfect here is that the new top choices are part of the explicit selections made by a student, resp. supervisor. These “second-order” perfect marriages are again made permanent and the above repeated, until no more perfect marriages are left in the database.

Then, the stable marriages algorithm is applied to produce a temporary allocation for the remaining projects. Partners whose marriage is not perfect can change their preferences. After each such change, the two steps searching for perfect, then for stable marriages, are carried out again.

A look at Figure 5 can help illustrate the idea of perfect marriages. For instance, in Case 1, (Student1, Project1) is, by definition, a perfect marriage.

Removing Project 1 from the top of the list of Student 2 means that now it can be perfectly matched with Project 3. As a result, the remaining student and project will also form a perfect marriage.⁶

An experiment with the 2000/01 data has shown that 2/3 of all projects would be immediately allocated on the basis of perfect marriages alone. This fact means that more than 65% of the students spent unnecessarily two long weeks waiting for their allocation to be published. One thing has to be made clear: the search for perfect marriages produces exactly the same allocations as the ones that the SM algorithm would have produced, had it been applied alone. The actual difference can be made more clear if it is compared to two different ways of using the SM algorithm.

SM can be used as the batch algorithm that it is, i.e., it can be run just once on the students' and supervisors' selections to produce all allocations. The problem with this approach is that students with random allocations do not get the chance to change their preferences and apply for another project. This setup would put pressure on students to examine and rank as many projects as possible before the allocation is made, which, as a result, would require from supervisors to discuss their projects with unnecessarily many students.

In comparison, in the previous allocation database, a student could change their choices with time, if the allocation of the initially ranked projects seemed unlikely. However, there is no way to avoid those random allocations that are due to changes in the database shortly before its closure.

Ideally, one would like to combine the immediate results that the single run of SM brings with the ability to add more selections, in the cases where the allocation is of poor quality (e.g., based on random ranking). The perfect marriages paradigm is an attempt to implement this combination. Allocations are gradually made permanent if they are (1) non-random and (2) represent a match that could not potentially be bettered if both partners' choices remained unchanged. As allocated projects disappear from students' lists, they can realise in time which projects are no longer available, and consider alternatives. In fact, the implementation at present guarantees that an E-mail warning is automatically sent to a student each time s/he is left with no explicit choices in the list. If a certain number of days before the allocation deadline the situation still remains unchanged, the student's current supervisor is also E-mailed, and requested to select projects on the student's behalf.

Phase 3

The database closes and all non-random allocations are made permanent. The database is reopened again with the remaining, hopefully very few students. The current supervisors of these students are E-mailed with the list of their names, and requested to advise the students and/or make a final choice on their behalf. The process ends when all allocations are made.

⁶They wouldn't, if, for instance, Project 2 had not explicitly ranked Student 3 on its list.

5.3 Tailor-made allocation constraints

The last feature, which has been added to the system, has to deal with the ability to impose additional, flexible constraints on the allocation process. The user of any fixed implementation of the project allocation can encounter a situation when an unsuitable allocation is made due to the impossibility to add the corresponding constraint to the algorithm used. Examples can include banning a particular couple of supervisee and supervisor from being matched due to a previous conflict or simply because the student, who has done a project with this supervisor in the previous year, wants to work with another one this time. Also, a handicapped student may have to choose from a narrower range of supervisors, etc. The tentative solution to this problem is based on the implementation of the stable marriages algorithm as a logic program in Prolog. A simple, yet functional, first draft of this implementation is shown in Appendix 6. It is expected that additional constraints expressed as logic predicates can easily be added to the program in the places marked inside the `propose/2` predicate. Nevertheless, a theoretical proof of the existence of a solution in this case or, at least, extensive practical tests have to be provided before the Prolog implementation of SM could be used. At present, the administrator can choose between the Prolog and the more standard and thoroughly tested Perl implementation.

6 Conclusions

The paper presents a critical analysis of a range of approaches used for the coordination of student project allocation. Rather than focussing on the automation of the process and the details of its algorithm only, an attempt is made to take into account the social dimension of the problem in the form of a student's absence or unwillingness to meet the deadlines, personal preferences, discontinued interest in projects, etc. Several of these issues have been addressed—first, on a more modest scale, in the approach used in the 2000/01 allocation, and second, in the completely re-designed new allocation approach. The latter is, in the author's judgement, a better attempt to allocate to the computer the tasks (or functions) that it is good at, while leaving room for a degree of flexibility guaranteed by human intervention, e.g., at the later stages of allocation. In this case, one could also see the parallel between the particular function allocation chosen, and the general theoretical guidelines for such allocation [She00]. Some of the computational issues involved in the implementation of the new system, such as the theoretical discussion behind the use of perfect marriages, although considered in detail by the author, have not been presented here in order to keep the document accessible to a larger range of readers.

The new system is still work in progress. The features implemented so far have to undergo more extensive tests. Ideally, these tests should involve selected students and members of staff to achieve realistic results. This document will be used as the basis for a future manual describing the new system in detail, which should help users in the trials and after, when the system is used in earnest.

There are also other, unpublished, features that are currently under study. One of them has to deal with the balance of supervisors' loading. The solution proposed is to remove all remaining projects of a supervisor, if s/he has entered into a number of perfect marriages equal to her/his full loading. In another attempt to facilitate remote negotiation between students and supervisors, they are shown whether they have been selected by the other party as their first choice, as a positive answer would mean that changing one's own preferences to create a perfect marriage would mean achieving a firm allocation immediately. The combination of these two techniques could encourage supervisors to be more flexible and change their initial ranking of students, as they would be attempting to reach their limit of perfect matches and get out of the system with a guaranteed standard number of projects.

All features of the system, whether, already existing or planned, have been designed with the aim to provide the students with the best possible chance to learn about research in the department by browsing through the lists of projects, which are to a large extent representative of staff members' interests, and select the project that would give them the best chance to achieve good results, and motivation to pursue these results to a successful end.

References

- [Cou01] Truda Counsell. Personal communication. Computer Science Department, University of York, 2001.
- [Dye01] Jonathan Dye. A constraint logic programming approach to the stable marriage problem and its application to student project allocation. Fourth year project supervised by Alan Frisch. Computer Science Department, University of York, June 2001.
- [Eva01] Andy Evans, editor. *Computer Science Students' Handbook*. Department of Computer Science, University of York, 26th edition, 2001.
- [Fis] A.J. Fisher. Student project allocation using the stable marriages algorithm: a web-based implementation. Department of Computer Science, University of York.
- [Kaz01] Dimitar Kazakov. Comments on the project allocation process. Presented to the CS Department Board of Studies, May 2001.
- [Knu97] Donald Ervin Knuth. *Stable marriage and its relation to other combinatorial problems: an introduction to the mathematical analysis of algorithms*. Providence, R.I., American Mathematical Society, 1997.
- [Run01] Colin Runciman. Personal communication. Computer Science Department, University of York, 2001.
- [She00] T.B. Sheridan. Function allocation: algorithm, alchemy or apostasy? *Int. J. Human-Computer Studies*, 52:203–216, 2000.

Prolog implementation of the stable marriages algorithm

```
:- dynamic engaged/2, m_pref/2.

% contains all men
man(a).
man(b).
man(c).

% contains all women
woman(m).
woman(n).
woman(o).
woman(p).

% for each man, a list of his preferences in decreasing order
m_pref(a,[m,n,o]).
m_pref(b,[n,m,o,p]).
m_pref(c,[p,m,n,o]).

% for each woman, a list of her preferences in decreasing order
w_pref(m,[b,a,c]).
w_pref(n,[c,b,a]).
w_pref(o,[a,c,b]).
w_pref(p,[a,c,b]).

% initial 'dummy' couples (Woman,noone)
engaged(m,noone).
engaged(n,noone).
engaged(o,noone).

% more facts to come
%
% whose_project(Project,Supervisor)
%
% loading(Supervisor,NoOfProjectsSoFar)

%
%
% appropriate(Project,Course1)
% appropriate(Project,Course2)

%
%
% incompatible(Supervisor,Student)
%
```

```

% self-explanatory
break_engagement(Woman,Man):-
retract(engaged(Woman,Man)).

% self-explanatory
get_engaged(Woman,Man):-
assert(engaged(Woman,Man)).

% self-explanatory
better_choice(BetterMan,WorseMan,ForWoman):-
w_pref(ForWoman,OrderedListOfMen),
to_the_left(BetterMan,WorseMan,OrderedListOfMen).

% Arg1 appears to the left of Arg2 in the list Arg3
to_the_left(BetterMan,WorseMan,[BetterMan|_]):-
!.
to_the_left(BetterMan,WorseMan,[WorseMan|_]):-
!, fail.

to_the_left(BetterMan,WorseMan,[_|OrderedListOfRemainingMen]):-
to_the_left(BetterMan,WorseMan,OrderedListOfRemainingMen).

% self-explanatory
next_best_choice(Man,NextBestWoman):-
m_pref(Man,[NextBestWoman|OrderedListOfRemainingWomen]),
retract(m_pref(Man,[NextBestWoman|OrderedListOfRemainingWomen])),
assert(m_pref(Man,OrderedListOfRemainingWomen)).

% self-explanatory
match(Man):-
repeat,
next_best_choice(Man,BestChoice),
propose(Man,BestChoice).

% self-explanatory
propose(Man,Woman):-
engaged(Woman,noone),
% additional constraints, such as max. loading per supervisor, to come here
break_engagement(Woman,noone),
get_engaged(Woman,Man).

propose(Man,Woman):-
engaged(Woman,AnotherMan),
better_choice(Man,AnotherMan,Woman),
% additional constraints, such as max. loading per supervisor, to come here
break_engagement(Woman,AnotherMan),
get_engaged(Woman,Man),
match(AnotherMan).

```

```

% resets initial couples (Woman,noone)
reset:-
retractall(engaged(_,_)),
woman(Woman),
assert(engaged(Woman,noone)),
fail.

reset.

repeat.
repeat:-
repeat.

% loop over all men
match_all:-
reset,
findall(Man,man(Man),AllMen),
match_all(AllMen).

match_all([Man|OtherMen]):-
match(Man),
% !, % do I need the cut?
match_all(OtherMen).

match_all([]).

% main predicate
do:-
match_all,
listing(engaged/2).

```