Executive Summary

Optimal allocation is a well-known problem that many scientists attempt to solve for precise allocation with minimum cost. Applying such method with constraint satisfaction has become a demand for various domains; one of which is academic bodies. From academic perspective, optimal allocation would be required in many application areas like allocation of: projects to students, reviewers to conference papers, markers to projects, students to personal tutors, class room allocation and many more useful applications.

This project applies the method of optimal allocation with constraint satisfaction through employing Answer Set Programming (ASP). The project was motivated by the demand of the department of computer science in the University of Bristol to have an optimum solution to allocate projects to students as well as allocating markers to projects. The allocation of projects to students has been achieved optimally through constraint programming using ASP. While the process of allocating markers to projects required more advanced technologies to extract knowledge from markers profiles in order to achieve optimum allocation to projects. This is why a text mining approach has become a real necessity.

The developed system has integrated a textual matching approach through employing SubSift[1], a text mining application with open access web service Application Programming Interface (API). This project applies SubSift to rank markers for most suitability to mark projects. ASP is then applied for final allocation of markers by optimising the process toward selecting the projects with best SubSift rank. ASP algorithm runs double optimisation processes for selecting best two markers.

This project has exploited the power of SubSift to enhance student experience. It adopts the same text matching features to suggest most suitable projects for students during the selection process. This makes the project a unique one by combining constraint satisfaction and textual matching in one application.

The project:
- Provides a web-based application for the entire student projects' cycle, from project selection to final marker assignment.
- Develops a generalised API for applying SubSift textual matching between any two groups of items, either by supplying the item content or a bookmark to its content. The new API is an interface to SubSift API.
- Develops an algorithm for optimal allocation by minimising rank among all selections with constraint satisfaction. The algorithm was adapted from the bottleneck algorithm proposed by (Proll, 1972).

[1] Available from http://subsift.ilrt.bris.ac.uk

Acknowledgments

I would like to thank Dr Oliver Ray, the project supervisor, for his encouragement and helpful guidance throughout this project. I also would like to thank my personal tutor Dr Tim Kovacs for his valuable advices. Thanks to Mr. Simon Price for his support in providing the access to SubSift web services API. Special thanks goes to my family, for their love, understanding and continuous support.

Table of Contents

Table of Figures

1 Introduction

Optimisation is a branch in mathematics that is mainly based on minimisation of some cost function. Many areas, specifically science and engineering, have found the way to applying optimisation for efficient allocation of resources. As one of possible application fields, academic institutions find optimisation a solution to many allocation requirements. Allocating projects to students and markers is a complex process that any university needs to address optimally. Department of Computer Science in the University of Bristol has this requirement and proposed this project to become a real application after completion.

A variety of algorithms currently exist for optimally solving the allocation problem, but they lack the proper load balancing consideration. This project will adopt an optimal algorithm for allocating projects to students, and markers to projects with respect to the constraints, preferences, and the loads on each supervisor and marker. The algorithm will be implemented using a declarative programming language Answer Set Programming ASP.

In the new system, students will be required to provide their preferences of projects by ranking their selections. Similarly, markers are required to be ranked for each project based on their expertise. This requirement wouldn't be possible to achieve without some text mining technique. Text mining approaches, like SubSift, are researched in this project to find the best way for textually matching projects descriptions with markers academic profiles.

The following section provides more light on the aims and objectives of this project, followed by a description of the structure of this report.

1.1 Aims and Objectives

Referring to its pervasive nature, the optimal allocation problem is required to be solved in an automated way for many application areas. Having a general application that can be used to allocate any sort of sampling is a real ambition. The aim of this project is to develop a powerful and comprehensive system that can be used for solving any allocation problem. It aims at empowering such system with fully customised features so users can declare applicable constraint. Of course this aim requires achieving an optimal allocation algorithm that can be adopted with constraint satisfaction.

To achieve this ambition, this project will start by developing an automated system for optimal allocation in one real-world application area. Assigning projects to students in the University of Bristol is a good start pilot for achieving

the aim. This allocation requires taking into account the loads on each supervisor, and the students' preferences over projects. Moreover, assigning markers to projects is another interesting objective that the project intends to accomplish.

This project will employ a diversity of advanced technologies to achieve optimal solutions to these two problems; i.e. student-project and project-marker allocation. Student-project allocation with satisfying pre-set constraints can be achieved through using one of constraint programming languages like ASP. Project-marker allocation problem needs more attention to markers expertise and hence requires text mining techniques to extract knowledge from markers profiles. The project will explore opportunities for adopting available powerful text mining tools; one of which is SubSift. At the end, projects need to be assigned to the right students, and markers to the right projects.

The project will take the opportunity to utilise SubSift power to aid student search process. It intends to find the relevancy between student preferences; i.e. search key words, and the proposed projects' descriptions, in order to make suggestion of best fit projects. This will make the student decision making process for selecting a project much easier and more efficient.

All these facilities can be provided through a tailored and secured graphical user interface with intuitive steps that can be easily used by all parties; i.e. project coordinators, supervisors, markers, and students.

1.2 Report Structure

This report is structured to aid the reader in following the project story from the idea till it comes to life and evaluation. After this introduction, the report explains the motivation of this project then it moves to the background chapter. After reviewing the literature for the required knowledge to work on this project, it moves to discuss the methods used for implementing the project. This is followed by analysis the results and evaluation of the achievements. Finally, the project is concluded and opportunities for further work are outlined. Following is a summary of the content of each of these main chapters:

Motivation: This chapter highlights the motivation of the project by reviewing a real-world allocation problem and discussing how it is being solved. It focuses on the disadvantages of the current way of solving the problem and emphasises the insisting need for having an optimal solution.

Background: This chapter starts with defining the problem type and discussing its computational complexity. It moves then to the subject of optimisation and reviews a range of optimal allocation algorithms from the literature. The reader is then directed to reviewing a constraint programming language that can be used

for solving such problems; ASP, through real-world scenarios. An innovative method for text mining through SubSift application is then discussed and alternative text mining techniques are outlined.

Methods: Starting with explaining the implementation methodology employed, this chapter goes through the project implementation life cycle. The cycle starts with planning for a system to solve the problems of allocating projects to students and markers to projects. Planning involves: system objects, the new process, system architecture, designing the optimal allocation algorithm and using a toy example to try it. Planning emphasises also on text mining design and selecting the proper technologies to implement the system. Planning section ends by reviewing the final planned model.

This chapter moves then to discuss the implementation stage by describing how each part of the architecture is built and how SubSift and ASP where used to implement the optimal allocation. The chapter ends by providing a guide for deploying the system in a real production environment.

Results: In this chapter, a number of scenarios are sketched and results of using the system to apply each scenario are analysed.

Evaluation: A number of methods for evaluating the developed system are discussed in this chapter to review to which extent the objectives are achieved. These methods include measuring the optimality of the solutions generated by the system, measuring the cost of the process, and examining user satisfaction level.

Conclusion and future work: The project is concluded in this chapter and opportunities for future work for improving the system are discussed.

2 Motivation

This project was motivated by the insisting demand of having an automated solution for assigning projects to students in the University of Bristol. The current system consumes considerable staff time and effort attempting to satisfy students, as well as load balancing on supervisors and markers. To clearly understand the motivation behind this project, the current system is discussed in the context of one academic year; i.e. year 2011/2012.

In this year, 182 projects were published as shown in Figure 1. The figure illustrates the current front-end system that the department implements. This system is very simple and only provides information about published projects. Around 120 students were required to go through the list manually one by one and read the description regarding any project which they think it is suitable for them to be able to select a project.

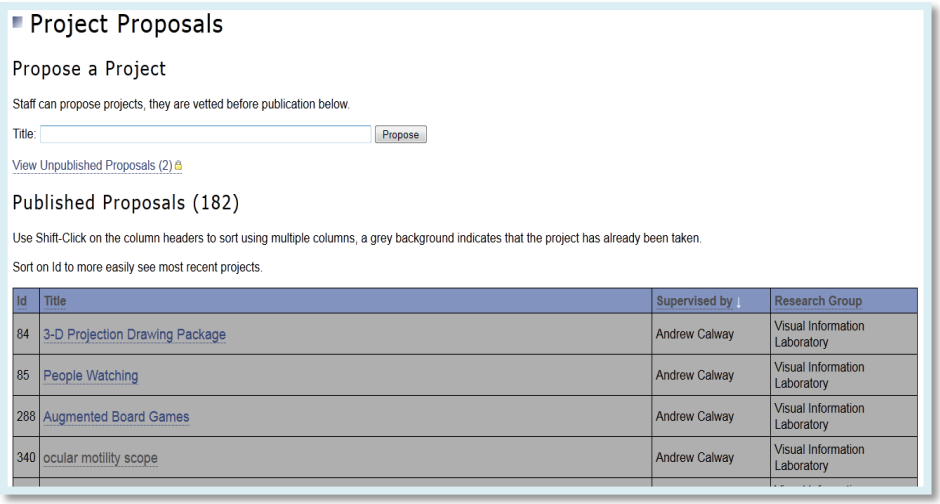


Figure 1- project proposals page
Captured from the University of Bristol department of Computer Science website

Although the current front-end system provides some sorting options, it is not sufficient to find about projects by just sorting the list. Searching and exploring such a long list, apart from the amount of time each student needs to spend, the process itself confuses students. This confusion might arise when someone at the early steps finds a project but still will continue searching because he thinks there might be another project 'somewhere in the list' that is more appropriate for him. One can imagine how much effort required for a student to just go through the description of each project in this list where still they require to research externally about some.

The list of projects needs to be reduced in reasonable way and only brings those projects which are best suitable for any individual student, based on the set

of criteria and preferences each student provide. The whole process of the current student project selection is shown by the flowchart in Figure 2.

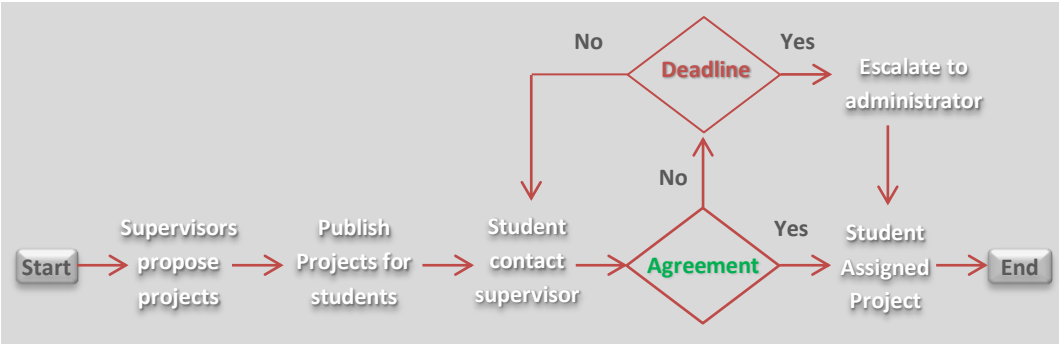


Figure 2 - process flowchart of current system

When a student manages to choose a project, he/she is required to make an arrangement for a meeting with potential supervisor. If the student manages to convince the supervisor during the discussion that he/she is the right one for the proposed project, he is assigned that project. However, there is a possibility that other students have already chosen the same project and are waiting for supervisor's decision or meeting. For such cases, current system does not show the queue on a particular project, then student needs to repeat search for another project. After all if agreed by supervisor on a particular project for a student, the process of allocation is performed manually by the project coordinator or the supervisor.

Issues are not only available in the project-student allocation process, but extend to the process of allocating markers as well. The projects coordinators need to allocate markers by hand and consider any conditions to resolve conflicts. One big concern that coordinators have is to address the issue of balancing loads on supervisors and markers.

As a result, there is an insisting need for a system that addresses all issues with the current system by employing a variety of technologies that will result in an intelligent system. Such system needs to implement the appropriate algorithm to solve the allocation problem. The project needs to select the right technologies to implement this algorithm. Additionally, text mining capabilities need to be utilised for providing suggestions to students and to assign markers based on expertise. These concepts need to be explored to decide how they can be combined to serve the allocation process.

Next chapter reviews the literature and leads the reader from understanding the basic concepts, and progresses toward selecting the right technologies to better understand how the allocation problem is optimally solved.

3 Background

This project deals with a problem which is compounded in nature. On the one hand, the problem of allocating projects to students requires that preferences are decided by students. On the other hand, projects-markers allocation requires the preferences to be derived first - based on items descriptions – before the assignment can be decided.

Having realised that the project consists of various components requires gaining knowledge from different disciplines to address the requirements in each component area. These disciplines include optimal allocation, constraint programming, and text mining.

In addition to discussing all these disciplines, it is important to understand the problem in hand and how complex it is. This will help deciding on the best technologies to solve it effectively. In the following sections, computational complexity is discussed in more detail in addition to the mentioned disciplines.

3.1 Computational Complexity

In theoretical computer science and mathematics, computational problems[1] are classified into complexity classes based on their difficulty; i.e. how much work they require to be solved (Tovey, 2002). Complexity classes are studied in the context of *languages* rather than problems it covers. Thus, any problem can be represented in some language with fixed alphabet in order to understand its complexity (Papadimitriou, 2003). For example, a decision problem like determining if an integer n is a prime number has the possible output alphabet $\{1, 0\}$ to denote *yes* or *no* respectively.

Most common computational problems can be solved by using polynomial-time algorithms; i.e. algorithms like Turing machines, (Arora & Barak, 2009)). Problems of this kind lie under class P. Other problems are not amenable to being solved in polynomial-time, those comes under complexity class NP, for *nondeterministic polynomial*. An example of such problems is the Travelling Salesman Problem (Womersley, 2001). Other complexity classes beyond NP include PSPACE and EXPSPACE (Arora & Barak, 2009).

There is a naming system for NP-family problems. The most commonly known names include NP-complete and NP-hard. Interestingly, not all problems that belong to these classes are NP. For instance, An NP-hard problem is at least as hard as any other NP problem, but is not necessarily to be NP. While an NP-

[1] A problem that can be solved by a computer

complete problem is both NP and NP-hard (Tovey, 2002). Figure 3 illustrates some of the complexity classes showing the relationships between them. Formal definitions of all of these classes are discussed in (Arora & Barak, 2009).

At this point, let us return back to our project problem and find out how complex it is. It is obvious that this project problem does not belong to **P** class as it cannot be solved in polynomial time. Given that NP-hard problems may include optimisation problems in addition to decision and search problems, and as the problem in hand is an optimisation problem, we can say that it can be classified under **NP**-hard class. To prove that this problem is NP-hard, reduction can be used to reduce a known NP-hard problem to our problem. Reducing a problem X to another problem Y is achieved by describing an algorithm to solve X assuming there already exist an algorithm for solving Y. Reduction is discussed in more detail in (Arora & Barak, 2009). Other NP-hardness proof techniques are explained thoroughly in (Tovey, 2002).

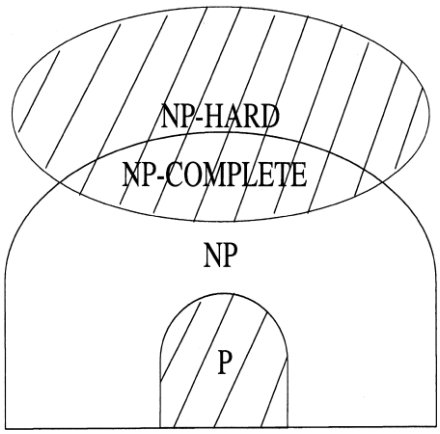


Figure 3 – Computational complexity classes relationships, from (Tovey, 2002)

Realising that the computational complexity of this problem is NP-hard would provide more confidence in researching the literature to find out how this problem would be solved.

3.2 Optimisation and Optimal Allocation Algorithms

There are many applications where it is required to find the best solution among possible solutions using algorithms in theoretical computer science and applied mathematics. Such problems are referred to as optimisation problems. Optimisation problems are NP-hard in complexity as mentioned in the previous section. Also referred to as Max-Min Problem (KO & LIN, 1995), an optimisation problem would contain at least one quantity that can be altered in order to optimise another quantity. It is called Min-Max as the optimum is normally achieved by maximising or minimising some quantities.

A special case of such optimisation problems is when the problem of interest encompasses two sets of items that are required to be allocated to each other in a way that some function is optimised; i.e. like minimising cost function. This is called Optimal Allocation and normally performed on discrete sets. This project concerns with achieving an optimal allocation of students and projects as well as between markers and projects with some constraint to be applied like load balancing.

There have been many efforts to solve this type of problems and this resulted in a variety of algorithms for optimal allocation in diversity of fields. More

advanced techniques may be required to be adopted with these algorithms to achieve a complete solution that considers constraint satisfaction. The following sections discuss examples of these algorithms in addition to related concepts like load balancing.

3.2.1 Bottleneck assignment algorithm

One solution for the allocation problem was proposed by (Proll, 1972) by using the bottleneck assignment functions. This method was used for assigning projects to students. In this scheme, the assignment was performed by taking into consideration students preferences. Thus, every student is asked to choose a specific number of projects from available list of proposed projects, and the selected projects list should be ranked by the student according to his/her preferences. Then the ranked project lists are collected and a student-project matrix is constructed such that an intersection of each student and project in this matrix will be filled by the student ranking number for that project (e.g. 1, 2,… or, any other valid ranking).

For the purpose of applying the bottleneck algorithm, this matrix should be converted into a square matrix. Usually, the number of projects are more than the number of students. Therefore, extra rows need to be added and should be filled by number 1. For any project that is not ranked by a student, the entry will be equated to *L+1*; where *L* is the number of projects ranked by this student. Now the collected data is ready for applying the algorithm. The main rule (Proll, 1972) applied to solve this problem is:

$$\text{Optimal total rank of assigned projects} = \min \max_{1 \le i \le M} \sum_{j=1}^{N} r_{ij}\, x_{ij}$$

$$\text{where } x_{ij} = \begin{cases} 1 & \text{if student } i \text{ is assigned to project } j \\ 0 & \text{otherwise} \end{cases}$$

and r_{ij} is student i rank of project j.

$$r_{ij} = \begin{cases} k & \text{if student i ranks project j as } k^{th} \\ L+1 & \text{if student i does not rank project j} \end{cases}$$

$$(i = 1, 2, 3, \dots M, \qquad j = 1, 2, 3, \dots N)$$

M is the total number of students and *N* is the total number of projects. *L* is the number of projects ranked by a student as mentioned earlier.

If the number of projects is more than the number of students, which is the case usually, the remaining entries are assigned as 1 to complete the square matrix as indicated below;

$$r_{ij} = 1\ (i = M+1, \dots, N, j = 1, 2, \dots, N)$$

This is a min-max problem (KO & LIN, 1995) and it means that the worst case (maximum cost) total sum of the assigned projects' ranks is minimised as much as possible.

In this project, similar approach can be applied where students provide their lists of projects ranked based on their preferences. The collected students ranked lists will be configured in a matrix to apply the algorithm with the min-max rule. The first step in this technique is to assign students to their first choices, and check if solution is possible. If so, the process terminates, otherwise the next preferred choice is assigned to students, and the process continues until an optimal solution is found.

3.2.2 Assignment using choice lists

Assigning items to people is one application area where optimal allocation would be required. This is closely analogous to the project allocation problem in this project. In (Wilson, 1977), three types of assigning items to people were introduced; stable, bottleneck, and numerical. These approaches are based on the assignment functions examined by (Gardenfors, 1973).

The general procedure is that each person is required to provide a list of his/her items ranked according to preference. Then a one-to-one relation is formed between people and items, so that each item is assigned to one person only and vice versa, based on some criteria. In this project, although the allocation of projects to students is a one-to-one assignment, projects and markers allocation is not completely a one to one assignment; i.e. a project needs to be marked by more than one marker, and each marker may have more than one project for marking. However, choice lists approach can be adapted so more items can be assigned to each person. One of the assignment functions used in this paper is explained below.

Stable functions

This type of assignment functions - also referred to as *Stable Marriage Assignment* - considers the preferences of both sides of the assignment. The assignment is said to be stable if there is no case of other unassigned pair with both sides prefer each other. In other words, if we have two lists P and X assigned to each other, this assignment function is stable if and only if there is no p_n and p_m in P that are assigned to x_r and x_t from X respectively where p_n prefers x_t and p_m prefers x_r. *Extended stability* holds if and only if there is no better assignment result that can be obtained by permutation of assigned items.

For the Stable marriage assignment function, there are four requirements that should be satisfied; (i) the outcome should be sensible and fair, (ii) a solution must exit, (iii) the number of output solutions should be small except for unusual cases, and (iv) obtaining the solution should be free of complexity with no much computation requirement.

Other researchers, such as (Gale & Shapley, 1962), adopted the same concept for different kind of allocation; i.e. applicant to college allocation in this case. They defined the stable applicant to college assignment as following:

> *"An assignment of applicants to colleges will be called unstable if there are two applicants α and β who are assigned to colleges A, B, respectively, although β prefers A to 'B' and 'A' prefers β to α".*

Applying (Gale & Shapley, 1962) definition of unstable assignments to the students projects allocation requires slight variation, as in our case the projects do not have preferences, but we can assume that each project requires particular expertise, so we can call the expertise requirements as project preferences. For example:

Assume projects P = {p1, p2} are required to be assigned to students S = {s1, s2}. If the output of the assignment matching is: (p1, s1), and (p2, s2), while s2 prefers p1 to p2, and p1 prefers s2 to s1. This assignment is said to be unstable. Obviously, an assignment is stable if it does not satisfy (Gale & Shapley, 1962) definition above. For instance from the above example, the assignment (p1, s2) is stable.

3.2.3 Load-balanced resource allocation

Although resource allocation algorithms are applied extensively in many application areas, few take the consideration of load balancing. To best of my knowledge, most available load balancing consideration is applied with no preference consideration. Examples include allocating radio frequencies to network channels (Fischer, et al., 2007), and allocating tasks to distributed processing sites. In both cases, neither channels nor processors have preferences. They are assigned based on balancing load among them.

Processing sites (servers) in a distributed network are allocated tasks based on their loads (Lu & Carey, 1986). In this paper, load L on each site s_i corresponds to the number of tasks NT waiting to be executed at this site, thus:

$$L(s_i) = NT(s_i) \quad (1)$$

To evaluate the task distribution balance of the whole system, the *load unbalance factor* LUF indicates the difference of loads between sites. With n sites, LUF can be defined as following:

$$LUF = \max_{j \geq 1, k \leq n} \left| L(s_j) - L(s_k) \right| \quad (2)$$

Using (1) and (2), the variance of the load distribution becomes:

$$LUF = \frac{\sum_{j=1}^{n} (L(s_j) - \bar{L})^2}{n} = \frac{\sum_{j=1}^{n} (NT(s_j) - \overline{NT})^2}{n}$$

Where $\overline{NT}$ is the average number of tasks of each site after allocation.

Based on above optimisation measure, tasks are allocated to sites.

The unbalance factor measure LUF can be applied in this project by balancing loads on supervisors and markers such that the variance on each is as small as possible.

In another context, (Ryabokon, et al., 2012) studies an optimal allocation of papers to reviewers. Adopting a similar concept as the unbalance factor, where workloads on reviewers are considered by setting a balancing criterion. This criterion is applied such that lower and upper workload boundaries are configured. The boundaries are initialised to the average number of papers and gradually altered to find the smallest workload variance.

Load balancing requirement is very common in industrial fields. In (Ricca, et al., 2011), workers in a seaport are required to serve ships in a way that load is balanced fairly. Since an ASP solution is implemented to address the seaport requirements, this application is explained in more detail in the next section.

After researching the literature for available optimal allocation algorithms, it is important now to understand the appropriate programming language to code such algorithms reliably with all constraint considerations applied.

3.3 ASP for optimal allocation

Constraint programming is intended for solving problems (expressed by variables) with constraint defined over them. Solving such problems requires assigning values to these variables with all constraints satisfied (Gavanelli & Rossi, 2010). ASP is a constraint programming language that uses finite domain constraints.

ASP is a declarative logic programming paradigm that can be used efficiently for modelling and solving optimisation problems particularly those that require weighted solutions (Çakmak, et al., 2009). The problem in this project involves: (i) solving assignments of equal preference (indifference), (ii) assigning markers to more than one project and vice versa, and (iii) refusing particular assignment (Incomplete List). This makes ASP a suitable candidate (Lifschitz,

2008) for solving the project allocation problem with high performance and less development effort.

The main idea in ASP is to solve a problem instance *I* by reducing it to compute stable models (solutions) (Eiter, 2008). Figure 4 outlines an overview of the process to produce stable models in ASP.

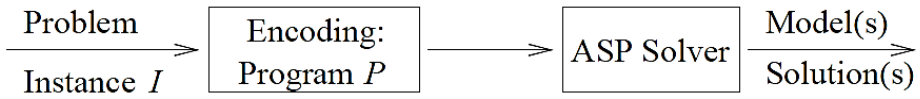


Figure 4 - ASP process overview, from (Eiter, 2008)

In recent years, ASP has been widely used for solving such problems like optimal resources allocation (Leite, et al., 2009) and reviewers to submission papers allocation. While other applications include assigning teams for serving ships in a port by taking into consideration assignment constraints and fairness of workload (Ricca, et al., 2011).

The following sections discuss the reviewer assignment and the team allocation problems using ASP.

3.3.1 Reviewer Assignment Problem using ASP

Conference submissions often require a thorough attention to appropriately select the right professionals to review the papers. Using its highly expressive nature, ASP has been adopted by (Ryabokon, et al., 2012) for assigning papers to reviewers based on their expertise. The problem is encoded by using the following main ASP predicates:

```
reviewer(revId)
paper(papId)
```

where `revId` and `papId` are reviewer and paper identifiers respectively.

Reviewers suitability for assignment to papers is determined by taking two factors into consideration: reviewer expertise level with regard to a paper, and the reviewer preference bidding.

Expertise is encoded by the following predicate:

```
paperExp(papId,revId,exp)
```

used for determining the level of expertise of reviewer `revId` with regard to paper `papId`

where `exp` may take any of the following values:

exp value	Level of expertise
0	Conflict
1	Low
2	moderate
3	High

Note:
conflict (value 0) may arise when this reviewer is also an author of this paper or biased due to some conditions like being in the same academic body.

Expertise values are specified by automatically examining reviewer expertise based on their published works. The paper applies SPARQL queries (explained in the following section) for extracting and matching expertise to papers.

Reviewers biddings for papers are expressed in the form:

```
reviewerBid(revId,papId,bid)
```

where `bid` may take one of the following values:

bid value	Preference level
0	Conflict
1	No bid
2	Weak
3	Strong

The paper/reviewer assignment pairs are represented by the predicate:

```
assign(papId,revId)
```

These pairs are obtained such that both predefined hard and soft constraints are satisfied either in favour of reviewers or papers. The assignment should be a stable marriage where no better optimal assignment exist. Both stability and workload are considered by setting a balancing criterion. This criterion is applied such that lower and upper workload boundaries are configured. The boundaries are initialised to the average number of papers and gradually altered to find the smallest workload variance.

The following section demonstrates another example of using ASP in solving allocation problems.

3.3.2 Team Allocation Problem using ASP

ASP has been found to be the optimal technique for allocating workers to serving ships in the Gioia Tauro seaport; the largest trans-shipment terminal in the Mediterranean coast (Ricca, et al., 2011). The seaport has predefined business scenarios and constraints that relates to work nature, availability of workers, skills, fairness of workloads, and work turnover between duty shifts. Regardless of the specific details about these constraints, ASP has been adopted to encode the problem using a variety of predicates some of them are outlined in table 1.

Table 1 - Some Team Allocation problem predictaes

Requirement	Predicate	Comments
Skills association	`hasSkill(employee, skill)`	
Availability	`absent(employee, shift)`	
Manual exclusion	`manuallyExcluded(employee, shift)`	
Shifts duration	`shift(shiftId, duration)`	
Shift sequence	`previousShift(shift, shift')`	`shift'` precedes `shift`
Resource requirement	`neededEmployees(shift, skill, n)`	`n` is the number of employees required
Past allocation statistics	`workedWeeklyHours(employee, hours)` `workedDailyHours(employee, hours)`	Hours worked up to current allocation
Overtime	`workedWeekOvertimeHours(employee, hours)`	

Maximum hours constraints	`dailyHours(hours)`
	`weekHours(hours)`
	`weekOvertime(hours)`	

The problem is then solved by specifying some rules based on the given input and by considering the predefined constraints. The following rules from (Ricca, et al., 2011) are some rules used for assigning workers:

```
canBeAssigned(Emloyee,Shift,Skill) :-
  hasSkill(Emloyee,Skill), neededEmployees(Shift,Skill,_),
  not absent(Emloyee,Shift), not manuallyExcluded(Emloyee,Shift),
  not exceedTimeLimit(Employee,Shift).
```

Above rule is used then in another rule (below) for the actual assignment:

```
assign(Employee,Shift,Skill) v notAssign(Employee,Shift,Skill) :-
  canBeAssigned(Employee,Shift,Skill).
```

Additionally, some filtering constraints are applied on assignments violating team constraints.

The following rule is used for balancing workload among employees:

```
prefByFairness(Employee1, Employee2,Shift,Skill) :-
       fairGap(MaxGap),
       workedWeeklyHours(Employee 1,Workedhours1),
       workedWeeklyHours(Employee 2, Workedhours 2),
       canBeAssigned(Employee 1, Shift, Skill),
       canBeAssigned(Employee 2, Shift, Skill),
       Workedhours 1 + MaxGap < Workedhours 2.
```

The paper addresses the inconsistencies that might occur as a result of combining the rules by prioritising constraints into a modified version of the program. The modified version is used only when no admissible solutions are found.

These two applications and other experiences in literature demonstrate that ASP provide acceptable solutions of most common allocation problems.

3.4 SubSift for textual matching

For optimal allocation of markers and projects, they need to be assigned based on information about specific characteristics. This information may include marker's expertise and project research fields. Additionally, students can be offered suitable projects based on information about their specialisation to assist them in making the decision. Such information is mostly unstructured in nature and might be stored in textual databases like text documents, publications, and other text resources. Retrieving and analysing this kind of information requires a specialised process like text mining.

Text mining refers to the process of extracting interesting information (patterns) and identifying relationships from unstructured text-based sources. The process is performed by employing computer learning techniques to automatically detect, analyse and extract high quality (non-trivial) information from text data sources. Text mining is sometimes referred to as Knowledge Discovery in Textual Databases (KDT) (Feldman & Dagan, 1995).

The interdisciplinary nature of text mining research area brings together insights from many fields including: natural languages, machine learning, data mining, and information retrieval (Feldman & Sanger, 2007). For instance, Information Retrieval is performed on text-based document databases for retrieving and organising the extracted information (Han & Kamber, 2000).

Text mining algorithms work on specific representations of text documents rather than the documents themselves; and those representations are feature-based. The most important features that are used to represent text documents include: characters (individual component like letters and numerals), words, terms (single or multi-word phrases), and concepts (where larger syntactical text units are categorised and related to concept identifiers) (Feldman & Sanger, 2007). It is important to have an intuition about these features in order to understand how text mining approaches operate.

There are various techniques for text mining available in the literature ranging from querying languages to full text mining applications. One of the querying languages that can be used to extract and match information from diverse data sources (like web text documents) is SPARQL[1]. SPARQL (Prud'hommeaux & Seaborne, 2008) is a query language used for querying RDF

[1] W3C recommendation on http://www.w3.org/TR/rdf-sparql-query

based documents (Resource Description Framework RDF is a W3C specification family for data interchange on the web)[1].

From querying languages to text mining applications, SubSift is one free-open source application that can be tailored to serve a spectrum of text mining situations. The following section provides more detail about SubSift followed by a section discussing alternate text mining approaches.

3.4.1 SubSift

Submission Sifting (SubSift[2]) is an innovative text mining application developed by the Intelligent Systems group at the University of Bristol. It was developed originally for matching reviewers and papers for an academic conference (Price, et al., 2010). The application discovers similarities between papers and reviewers by referring to textual sources like papers abstracts and online bibliography databases. SubSift ranks the resulted matches (assignments or paper-reviewer pairs) which would provide significant assistance for conference chairs to decide the final allocation process.

SubSift is a collection of REpresentational State Transfer (REST) web services (Feng, et al., 2009) provided as open source and freely available. Thus, SubSift requires no installation efforts and can be used through its API. Figure 6 shows the system architecture of SubSift.

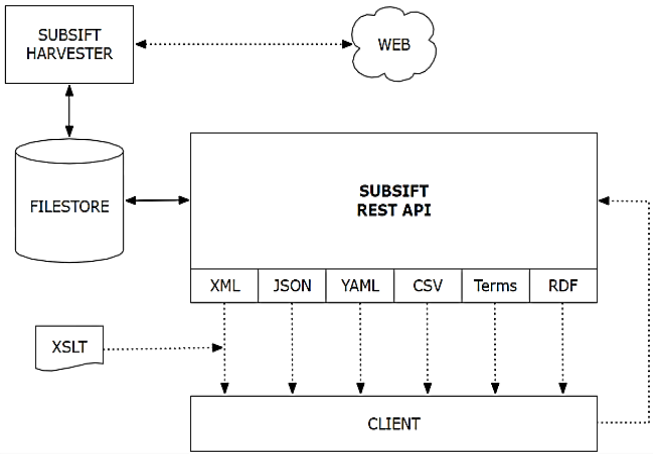


Figure 5 - SubSift Architecture
From (Price, et al., 2010)

The main theoretical concept of SubSift for profiling and matching text is based on the information retrieval algorithm: *vector space model*. This model discovers similarity between a text query and documents by applying a weighting scheme that considers term frequencies in documents. SubSift performs profiling and matching by applying this scheme but with considering both matching documents for term frequencies calculation as following:

$$tf_{ij} = \frac{n_{ij}}{\sum_k n_{kj}}, \quad idf_i = \log_2\left(\frac{|D_1 \cup D_2|}{df_i}\right), \quad tf.idf_{ij} = tf_{ij} \times idf_i$$

[1] http://www.w3.org/RDF
[2] Available from http://subsift.ilrt.bris.ac.uk

Where tf_{ij} is the frequency n_{ij} of term t_i in document d_j divided by the total number of terms in this document. idf_i is the inverse document frequency which is based on the number of documents df_i from both document collections (D_1 and D_2) containing this term. The weight of each term is then obtained by multiplying the two values. All the weights or scores of terms in a document are form document *profile*. Profiles of one collection (e.g. submissions) is matched up with all profiles of the other collection (e.g. reviewers). A final score is calculated for each match with details about terms contributions towards this score. Final scores determines the rank of matches and hence leads to the most suitable assignment. Figure 7 shows the folder organisation of SubSift processes and how it is modelled based on the filing system.

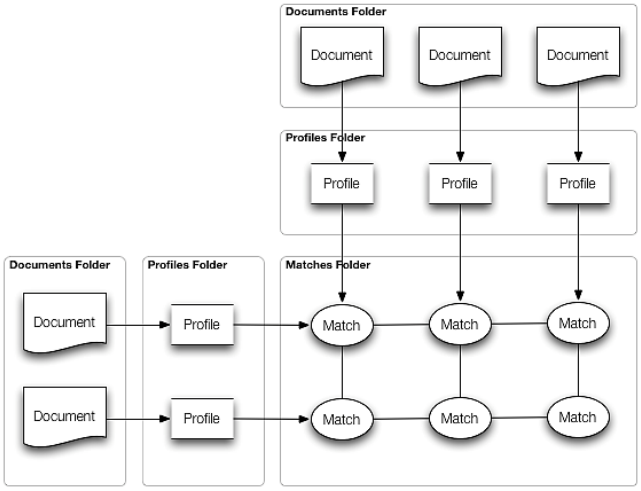


Figure 6 – SubSift folder structure and matching pairs of documents profiles from http://subsift.ilrt.bris.ac.uk

The original SubSift software was first applied - as mentioned - for conference submission sifting workflow in order to assign submissions to reviewers. Other promising SubSift use cases are discussed in (Price, et al., 2010) like finding the most suitable professional for a specific task or topic, researcher bookmarks lists profiling, and other useful applications. Figure 8 shows the original SubSift workflow design pattern.

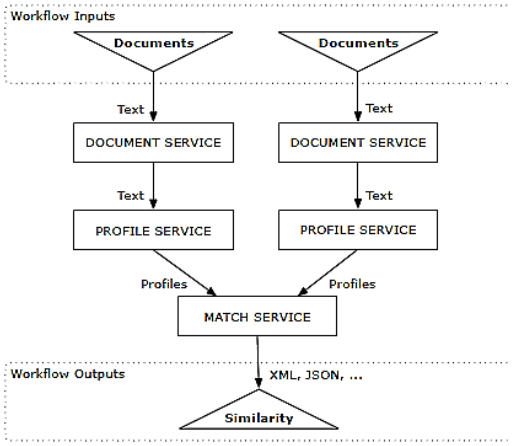


Figure 7 - Original SubSift workflow design pattern From (Price, et al., 2010)

SubSift is a suitable tool that can be integrated in this project to efficiently assign projects to markers based on their expertise. Moreover, it can be utilised for offering students a list of suitable projects to support their decision.

The following section discusses some alternate text mining approaches available in the literature.

3.4.2 Other text mining approaches

In addition to the vector space model (VSM) approach; i.e. *tf-idf* approach discussed in SubSift section, there are other application fields that apply other techniques to perform text mining. Text Categorisation (or TC) is one method for classifying text based on assigning labels to documents. TC is solved using one of the following inference types: Inductive (estimating a model using training data), or Transductive (estimating a label of the new data) (Berry & Castellanos, 2008).

Clustering is another technique used in text mining where algorithm like *k-means* or *principle components* are applied to cluster text based on similarity in features. Clustering tasks and algorithms are discussed in (Feldman & Sanger, 2007).

As mentioned earlier, text mining involves information retrieval. Information retrieval or sometimes called *Information Extraction (IE)* is another technique in text mining. IE using vector space model VSM is examined in (Berry & Castellanos, 2008). Thus, SubSift is one approach that is considered to be IE.

Understanding the variety of text mining techniques would assist in selecting and implementing the right tools for applying textual matching in this project. At this point, I can say that we have reviewed the required knowledge for implementing such system efficiently and achieve the pre-set objectives.

4 Methods

This project comprises multiple components that are required to be designed and implemented carefully to work successfully in one integrated framework. To make this story more interesting, this chapter is structured in relation to the actual work performed. It starts with a brief description about the project life cycle methodology adopted including an explanation of each individual stage in the cycle. Each stage is then expanded to show all work performed.

4.1 Implementation methodology

Microsoft Solutions Framework (MSF) is a milestone-driven iterative methodology that combines the principles of waterfall as well as spiral models. MSF is selected to be the methodology for implementing this project. Figure 9 illustrates the various phases of this methodology. The iterative characteristic allows flexible feedback of processes which provides more creativity.

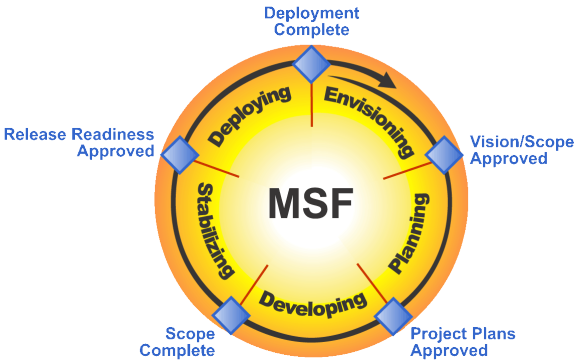


Figure 8 - Microsoft Solutions Framework, from http://technet.microsoft.com

The life cycle starts with the envisioning phase where project vision and scope are identified and project feasibility is studied. Planning phase involves detailed planning for the algorithm to be used, constraint programming using ASP, text mining with SubSift, and user interface designs. Developing phase then starts to implement the planned designs. After completing the development stage, stabilizing is performed to test the system collectively. The system then is deployed and moved to production. Although this project has not been deployed yet, deployment phase section will provide the necessary guidance for deploying such system to production environment.

Why iterative?

There might be situations where some requirements are changed during the life cycle. In this project, those changes may include changing existing or adopting new constraints. Other changes may extend to adding totally new functionalities to the system like the optimal allocation of time slots for project marking process. Despite of its type, any change needs to start a new sub-cycle of MSF.

This means that its feasibility needs to be studied and if feasible, it gets added to the project scope. Then it needs to be planned and implemented in the context of the main project life cycle.

For this project, envisioning phase is already covered in the previous chapters having project objectives and scope well understood at this stage. For the remaining phases, we will use different names so to be more expressive or specific to the project.

This mapping might not be complete, as some tasks are partially mapped to the tasks in the newly named phase. The idea is to adopt the iterative mile-stone methodology rather than applying MSF literally .

4.2 Planning for optimal allocation

Good planning is the key to succeed in any project. Planning the various components in this project would require understanding objects involved and outlining the new process to get a better overall insight of the requirements. Additionally, system architecture needs to be sketched to show where each component rests and its relation to other components in the system.

To solve the main problem; i.e. the optimal allocation, the algorithm to be adopted needs to be designed with all requirements considered. And before it can be implemented, it is good to try any algorithm using a toy program. Moving to the other part of the main problem, markers-projects allocation would require a good planned text mining approach.

This chapter goes through all these plans and ends the series with planning the technologies for implementing them.

4.2.1 System Objects

Apart from other features that it aims to provide, this project is mainly concerned with allocating projects to students and markers to projects. It is important at a very early stage to identify the objects involved and how they are related to each other. This will have great impact on how the system is designed and implemented. For this, a high level UML class diagram would provide necessary intuition of the various objects involved. Figure 10 shows the main entities in this project.

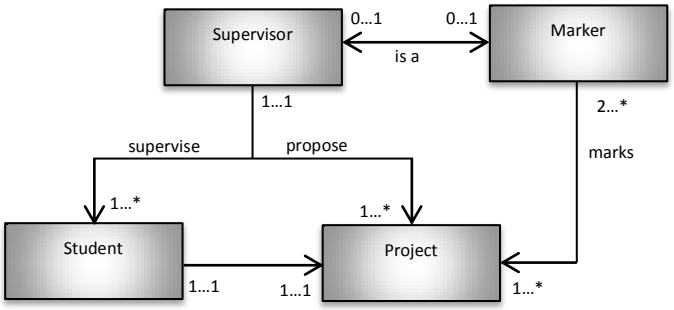


Figure 9 – High level UML diagram

The UML diagram shows that this project has four strong entities, namely supervisor, marker, student, and project. For this system, the relationships between these entities would be as follows:

- Each supervisor may be a marker or not and vice versa;
- Any supervisor may propose one or more projects, and each project is proposed by only one supervisor;
- Each student is supervised by one supervisor and any supervisor supervises one or more students;
- A marker may mark one project or more and each project would be marked by two or more markers.

The maximum number (denoted by * in Figure 10) for any relationship can be customised. In addition to these entities, there is another entity "Coordinator" which has a relationship with each of the above entities. This entity is left out as it is the controller of the whole system rather than being part of the process.

4.2.2 The new process

Developing software applications may require adapting the underlying business process where applicable so users involved can easily shift to using the new system. Of course this will depend on the type of the developed software and the motivation of building it. In this project, the processes of assigning projects to students and markers to projects needs to be modified to address the limitations in the current system.

For easy comparison and to figure out what changes are made, the current and the new processes for allocating projects to students are sketched in figures 11 and 12.

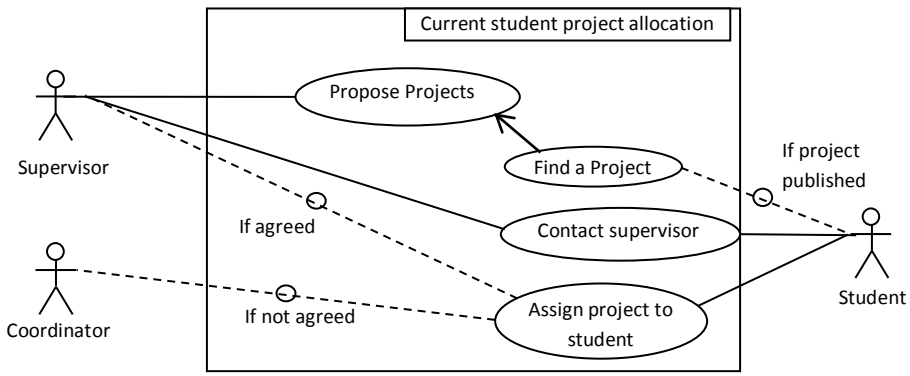


Figure 10 – Use Case Diagram of the current student project allocation process

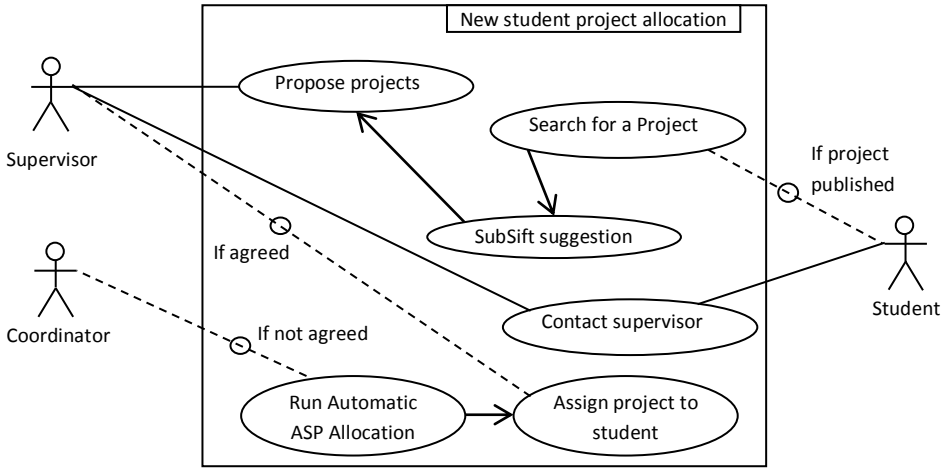


Figure 11 - Use Case Diagram for the new student project allocation process

Similarly, figures 13 and 14 illustrate use case diagrams for the project marker allocation process, current and new respectively.

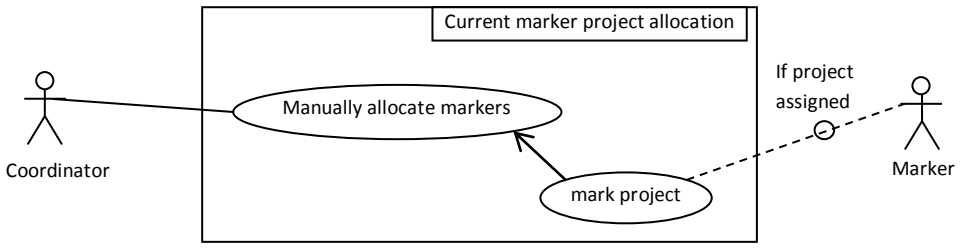


Figure 12 - Use Case Diagram for the current marker project allocation

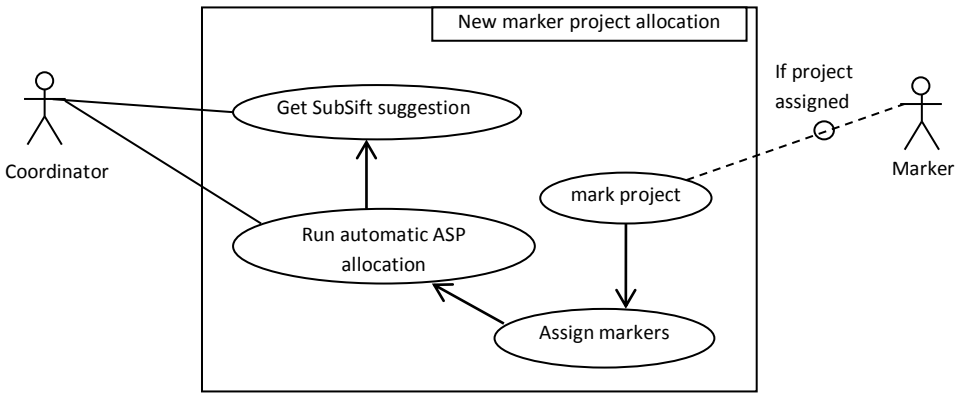


Figure 13 - Use Case Diagram for the new marker project allocation

4.2.3 System Architecture

The project of optimal allocation is designed to operate a web-based software which is comprised of three main layers as shown in Figure 15. These three layers are Front-end, Controlling, and Back-end. Layers and their components' planned specifications are described in more detail below.

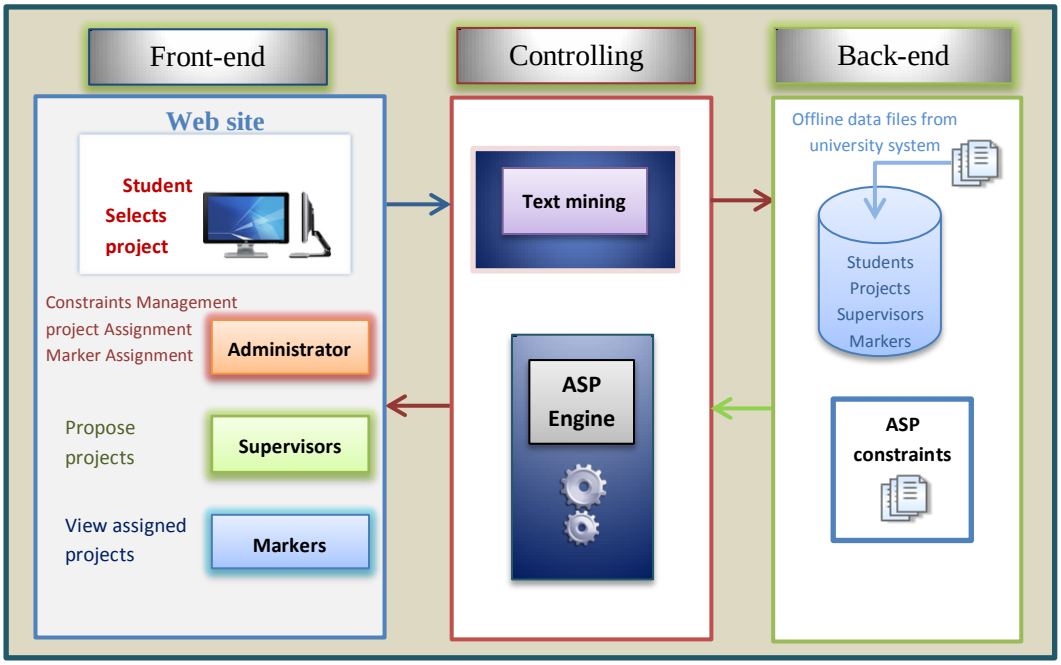


Figure 14 – Project system architecture

Front-end Specification

The Front-end layer is the top level of the application which is a part of the system that users can interact directly with the system through visual interface and web pages.

Front-end layer will be constructed to serve different users such as students, administrator, supervisors, and markers through a friendly user interface environment. Each of these users will have a page to perform their tasks.

Student Workspace

The students workspace will contain several features for helping students on project selection. Apart from friendly environment interface, the system will suggest a short list of projects which are most suitable to the current student based on student interests.

The system will provide students the options of categorising and filtering the list of projects based on the project type or skills required via choosing the preferred filters from dropdown lists or type-in the required search terms. The system should allow searches that involve a combination of two or more filters.

Students will be able to distinguish the queue on each project during arranging his preferred project list by displaying the number of students having these projects in their lists.

The system will perform the filtering and categorising of projects by applying information retrieval and text mining techniques. Details can be found in text mining design section.

Administrator Workspace

The system will provide the administrator (who is normally the projects coordinator) a user friendly workspace for the entire process of project allocation and system management. Through easy and well-defined steps, the administrator will be able to control and regulate the whole process of assigning projects and manipulate constraints. The system will support the process of assigning projects to students in fully automated approach by just monitoring the workflows as the system will perform entire process ranging from collecting students preferences projects lists and performing the algorithm for projects assignment.

Similarly, the process of assigning markers to projects will be performed automatically through integration with text mining tools to discover the markers expertise and projects areas for assigning projects to markers. For details on information retrieval refer to the text mining design section.

Supervisor and Marker Workspace

The supervisor workspace is dedicated for proposing projects, and will be monitored and updated by supervisors. The system will demonstrate up-to-date information regarding each project to supervisors such as those displayed for the administrator except that the supervisor can only access his/her list of projects. In addition, supervisors will be able to track queues on their projects, with the facility to access tailored information about students in these queues. Supervisors will be required to enter relevant and sufficient information about each project to enrich the content for text mining accuracy. For data extraction techniques refer to text mining design section.

The marker workspace will enable each marker to view a list of projects assigned to him/her. The interface of the page will be more like a timetable for marking projects schedules. A marker can view all relevant information such as the number of projects he/she should mark with exact date and time, in addition, information about students and projects with links to submitted students documents.

Controlling specification

The Controlling layer or middle tier is the logical layer between the Front-end (user interface) and the Back-end (data bases) layers.

The main application processes encompass in this layer are the process of applying text mining techniques like SubSift and performing the assignment process by using Answer Set Programming. Additionally, this layer will include data retrieval interface that performs necessary database access operations. Designing text mining and ASP are explained in later sections.

Back-end specification

The Back-end components of the proposed project are the system database (where required information for fulfilling the process of allocation is stored regarding all parties), web data sources (references to markers publications), and files or any other repository allocated by university for storing all ASP constraints on projects allocation problem. Data from university database can be obtained as offline text files and in turn can be imported to the system internal database.

ASP constraints is the main component for storing ASP programs to be run by ASP engine in the Controlling layer. ASP design is discussed in a later section.

4.2.4 Algorithm Design

"Optimal Allocation" is the key concept (service) that this project argues to provide. In front of all available optimal allocation algorithms, and specifically those discussed in the Background chapter, one would have to find the appropriate algorithm for the problem in hand and adapt it if necessary.

Although does not consider load balancing, the Bottleneck algorithm seems very suitable for the problem. It addresses the preferences requirement efficiently and produces assignments of lowest cost.

To demonstrate how this algorithm can solve the problem, let us work out a simple example as following:

Assuming that the number of students m = 3, projects n = 4, and student preferred list size l = 2
(i.e. students should select and rank two projects out of 4)

M = { s1, s2, s3} N = { p1, p2, p3, p4}

Ls1 = { p1, p3} Ls2 = { p3, p4} Ls3 = { p1, p2}

Where Ls is the preference list of the respective student.

Then the Rank Matrix (of ranks r_{ij}) becomes:

	p1	p2	p3	p4
s1	**1**	3	**2**	3
s2	3	3	**1**	**2**
s3	**1**	**2**	3	3
-	1	1	1	1

The bottleneck assignment problem is solved by finding the optimal (minimum) of the worst (maximum) total rank among assigned projects:

$$\text{Optimal total rank} = \min \max_{1 \le i \le m} \sum_{j=1}^{n} r_{ij}\, x_{ij}$$

By trying all possibilities by hand, an optimal instance of the possible solutions would be as outlined in table 2 .

Table 2 – an optimal solution using bottleneck algorithm

i	j	r_{ij}	x_{ij}	$r_{ij}\, x_{ij}$
1	1	1	1	1
1	2	3	0	0
1	3	2	0	0
1	4	3	0	0
			Min	1
2	1	3	0	0
2	2	3	0	0
2	3	1	1	1
2	4	2	0	0
			Min	1
3	1	1	0	0
3	2	2	1	2
3	3	3	0	0
3	4	3	0	0
			Min	2
			Total Sum	4

Note that the same project cannot be assigned to more than one student which reduces the possible solutions considerably.

Maximum (worst) score for each student would be 3. The idea here is to avoid these max scores as much as possible and assign the project that gives the

minimum score (projects of lower rank). Trying to find the minimum of the total maximums (i.e. 9), the optimal solution as demonstrated in the table above (having total score4) will be:

{ (s1, p1), (s2, p3), (s3, p2) }

Students s1 and s2 have been assigned to their first choice and student s3 to the second choice.

The plan is to scale this problem to a real number of students and projects. Remember that this algorithm does not consider any loads on project supervisors (or markers in case of allocating markers to projects) which is a requirement in this project. The algorithm is also not designed for allocating more items instead of one to one assignment (which is a requirement in allocating markers to projects). The following section shows how these requirements can be achieved through constraint programming using ASP.

4.2.5 A Toy ASP program

ASP is selected to be the language for applying the bottleneck algorithm. It has all the capabilities that any optimisation problem would require. For our case, all requirements that are left not designed in the selected algorithm can be designed in ASP for its constraint programming abilities.

Having the confidence that ASP is capable of handling constraints, this would release the stress at this stage and simplify the design of ASP routine. In this section, we will transform the example discussed in the previous section into a toy ASP program. This would simplify the implementation of the real ASP program and would give a good experience for using the language in real optimisation problems.

Having analysed all steps in the algorithm, we can see how each step can be coded in ASP. Projects' list, students' list, and students' rank matrix are all called ground facts in ASP and can be encoded as following:

```
project(1..4).            % define 4 projects
student(1..3).            % define 3 students

rank(1,1,1;;1,2,3;;1,3,2;;1,4,3).          % rankings of student 1
rank(2,1,3;;2,2,3;;2,3,1;;2,4,2).          % rankings of student 2
rank(3,1,1;;3,2,2;;3,3,3;;3,4,3).          % rankings of student 3
```

Now, the one-to-one assignment statement can be written as:

```
1{assign(S,P):project(P)}1 :- student(S).
```

Assigning the same project to more than one student can be restricted with the following constraint (where symbol **:-** means *not*):

```
:- assign(S1,P), assign(S2,P), S1 != S2.
```

The optimisation is then applied by finding the rank of each assigned project (`bad` predicate), and then looking for the total worst rank among all assigned projects and finally minimise the worst total quantity by using `#minimize`:

```
% student S has badness R under the assignment
bad(S,R) :- assign(S,P), rank(S,P,R).

% some student S2 is worse off than S1
worse(S1) :- bad(S1,R1), bad(S2,R2), R1<R2.

% R is score of worst off student
worst(R) :- bad(S,R), not worse(S).

totalSum(T) :- W = #sum [bad(_,R)=R].
#minimize [totalSum(T)=T].
```

Running this program in using Gringo and Clasp ASP solvers (discussed in technologies selection section), the output is:

```
clasp version 2.1.0
Reading from stdin
Solving...
Answer: 1
assign(3,2) assign(2,4) assign(1,1) bad(3,2) bad(2,2) bad(1,1)
Optimization: 5
Answer: 2
assign(3,2) assign(2,3) assign(1,1) bad(3,2) bad(2,1) bad(1,1)
Optimization: 4
OPTIMUM FOUND

Models       : 1
  Enumerated: 2
  Optimum   : yes
Optimization: 4
Time         : 0.022s (Solving: 0.02s 1st Model: 0.00s Unsat: 0.01s)
CPU Time     : 0.016s
```

The optimum found here (Answer 2) is the same optimal solution obtained from working the algorithm manually; i.e. { (s1, p1), (s2, p3), (s3, p2) }.

To ensure that the minimum total rank is obtained also with minimum individual rank, so good assignments are not selected on the account of some others, another minimisation is required:

```
#minimize [worst(R)=R].
```

This will make sure ranks selected are the best and also with best total. The output produced is:

```
Answer: 1
assign(3,2) assign(2,4) assign(1,1) bad(3,2) bad(2,2) bad(1,1)
Optimization: 2 5
Answer: 2
assign(3,2) assign(2,3) assign(1,1) bad(3,2) bad(2,1) bad(1,1)
Optimization: 2 4
OPTIMUM FOUND
```

maximum rank of any assigned project is 2,
and maximum total sum of all selected ranks is 4

First optimisation number 2 refers to the last `#minimize` added and has higher priority than the second quantity to be optimised.

This additional minimisation adds one more optimisation to the original bottleneck algorithm as following:

Table 3 - Addition to the original bottleneck algorithm

For all projects $j = 1, 2, \dots, N$

Optimal rank of any individual assigned project = $\min \max_{1 \le i \le M} r_{ij}\, x_{ij}$

There is another alternative approach that can be followed which is minimising the individual project rank without minimising the total rank. But this would give a solution of total rank 5 as optimum since the maximum is still 2; i.e. assign projects of ranks 2, 2, 1 to the three students.

Playing with this toy program was really interesting and provided the required ground for implementing the real ASP code. Next section plans for text mining through SubSift.

4.2.6 Text Mining design

In this project, text mining is not an allocation technology, rather it is a supporting process for making the allocation algorithm works efficiently. There are two areas in this project for implementing text mining techniques; one for assigning markers to projects, and the other for offering projects to students. The general process of text mining is illustrated in Figure 16.

In either areas, the type of information required for text mining - which are mostly stored in textual databases - are semi structured. Consequently, an automated approaches of text mining should be employed that is able to extract these type of information. SubSift is a suitable candidate that can be used to perform text mining process based on Information Extraction and all its auxiliary processes such as data pre-processing.

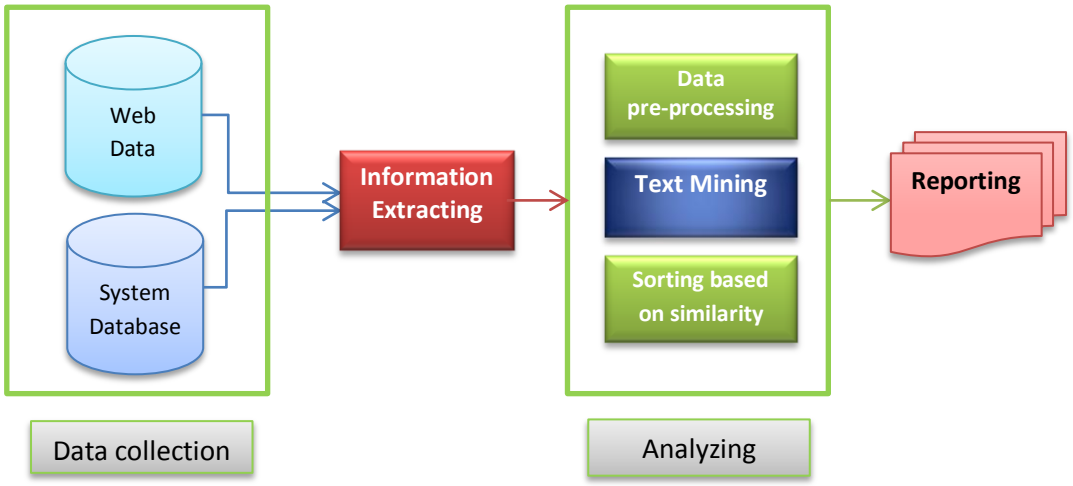


Figure 15 – Text Mining process breakdown

Designing text mining for each of the two areas are discussed below in more details.

Marker-project text matching

To make an optimal assignment of projects and markers, we have to discover the field and expertise of each marker's prior to the assignment. The information required is mostly available in text databases. Regardless of the type of information available, SubSift arranges them in a specific folder structure to perform the final matching. Figure 17 shows the folder structure plan for markers matching to projects.

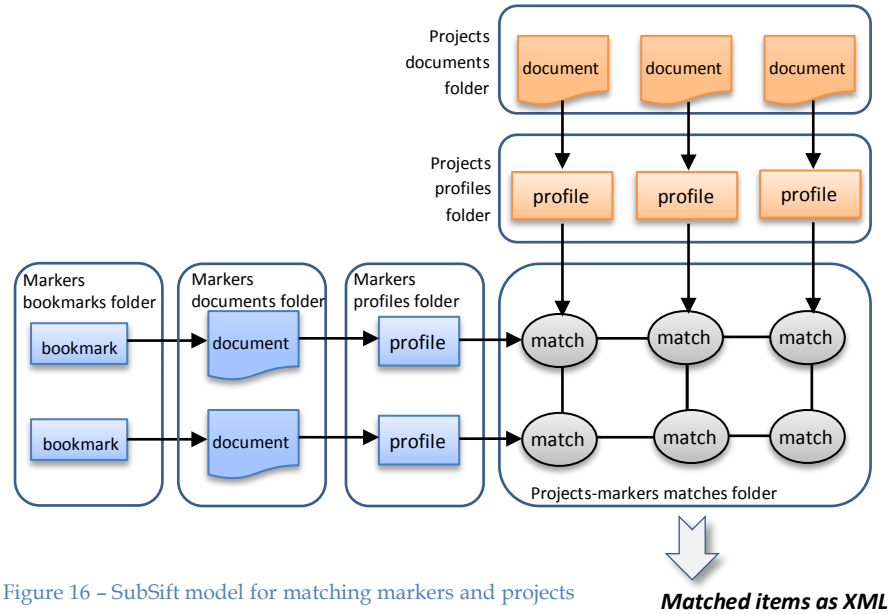


Figure 16 – SubSift model for matching markers and projects

Creation of this folder structure requires invoking the right Methods using the SubSift web services API. The sequence of these method calls for this project is planned as outlined in table 4.

Table 4 - SubSift matching plan for Markers

	Description	Method	URI format	Parameters
Step 1	Create markers bookmark folder	POST	*/bookmarks/markers*	
Step 2	Create markers bookmark items	POST	*/bookmarks/markers/items*	*items_list* the list of all markers URLs as a string
Step 3	Create markers documents folder	POST	*/documents/markers*	
Step 4	import markers document items from web pages (bookmarks)	POST	*/documents/markers/import/markers*	
Step 5	Create markers profiles	POST	*/profiles/from/markers* Or just: */profiles*	

Similarly, projects folder structure needs to be created before matching with markers can be performed. The steps for projects is quite the same as those of markers, except that projects do not require bookmarks structure. Table 5 outlines the plan for projects.

Table 5 - SubSift matching plan for Projects

	Description	Method	URI format	Parameters
Step 1	Create projects documents folder	POST	*/documents/projects*	
Step 2	Create projects documents items	POST	*/documents/projects/items*	*items_list* the list of all projects abstracts as a string
Step 3	Create projects profiles	POST	*/profiles/from/projects* Or just: */profiles*	

Folders can be named differently for each type of folders, but for simplicity, the same name is recommended to be used; i.e. all markers related folders are named as *markers*, and *projects* name is used for all projects folders.

Finally, for matching markers and projects, SubSift calculate scores (*tf.idf*) of the two profiles and provides the result in the format specified by the user.

Textual matching of markers and projects can be achieved by following the plan outlined in table 6.

Table 6 - SubSift matching plan between Markers and Projects

	Description	Method	URI format	Parameters
Step 1	Matching markers and projects Profiles	POST	*/matches/markers_projects/profiles /markers/with/projects*	
Step 2	Get Matching results	GET	*/ matches/markers_projects/items*	*profiles_id* this parameter can be given value of *projects* so to have result grouped by projects.

The final result can take one of several formats; like XML and JSON. The plan is to get the default XML formatted result.

Student-search text matching

Students project selection in the current system is not efficient as we realised, since it consumes a large amount of students' time for finding a suitable project among a long list. To help them in selecting an appropriate project based on their preferences, it is better to reduce the number of projects to be searched by students through filtering those projects which are not relevant. Therefore, we need to obtain these information from project proposal pages (project's description and abstract) for categorising them. Again, SubSift can be used in a similar way to the markers', but this time, the item to be matched with projects is the search term. Figure 18 shows the folder structure for projects-search matching.

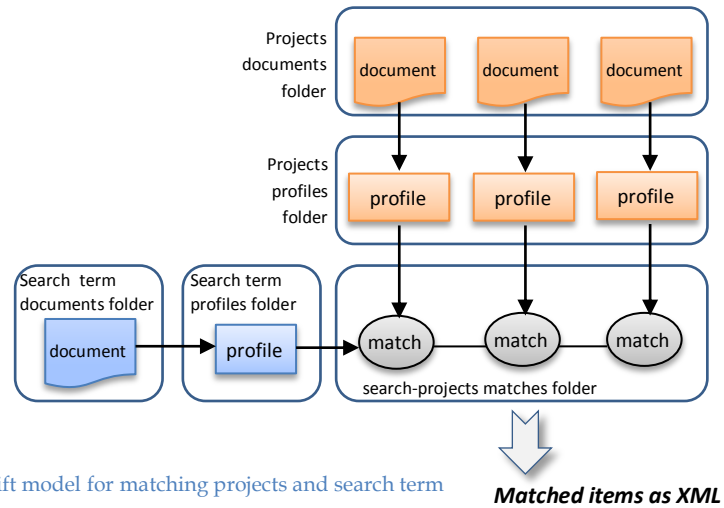


Figure 17 – SubSift model for matching projects and search term

Projects plan for SubSift matching is exactly as outlined in table 5. Interestingly, projects can be prepared only once for the use of both areas of application; i.e. with markers and with student search. For the search term, similar plan can be followed except *projects* folder name needs to be replaced

with another name like *search*. In this matching, the difference is that there is only one item to be matched to a group of items. the final matching plan is outlined in table 7.

Table 7 - SubSift matching plan between Projects and Search term

	Description	Method	URI format	Parameters
Step 1	Matching projects and search Profiles	POST	*/matches/projects_search/profiles /projects/with/search*	
Step 2	Get Matching results	GET	*/ matches/projects_search/items*	*profiles_id* this parameter can be given value of *search* so to have result grouped by search.

Important Notes on SubSift Usage

In order to be able to consume the SubSift online web services API, administrators of SubSift need to create a new user id and assign a Token to it. All URIs specified in the previous plans need to be preceded with the following:

http://subsift.ilrt.bris.ac.uk/<user-id>

And all http method requests need to be supplied by the Token so to be authenticated in SubSift web server.

There are other Methods that can be invoked for certain purposes; like calling DELETE to delete an existing folder so to create a new folder with same name. HEAD method is also another method that is useful for finding out if some processes have completed, like *import*. Although fetching text from web sites may take time, this process is performed by the SubSift harvester robot as a background task. Thus, HEAD can be used to check whether the background process has finished.

Text Mining Data Sources

Just to emphasis the source of information for text mining, we are highlighting them here in the context of their application in this project. Later when discussing system implementation, the data sources will be reviewed with all its specific detail like data format.

Information required for the purpose of filtering and categorising relevant projects to student, when they search for a project, could be extracted from the project proposal pages, or from any other data source which the administrator specifies. In addition to the abstract or short description that each supervisor provides with the proposed project, keywords entered by supervisors can also be

utilised. Subsequently, by using a tool like SubSift, we can extract information required from the project description and the keywords. Then the system will classify them, and introduces suitable list of projects to each student.

The same project information can be used for markers-project matching. Information regarding markers can be usually found from the university web site e.g. lecturer page Research interests, teaching section, or university databases such as publications database of the Department of Computer Science, and from other online published publications.

4.2.7 Technologies selection

This project combines a beautiful bunch of concepts that require a diversity of technologies for applying them. Following is an outline of the technologies to be used for each of these concepts:

Answer Set Programming	Solver: *gringo 3.0.4* for windows Output: *clasp 2.1.0*
Text mining	SubSift REST API
System framework and front-end	ASP.NET (C#) using Visual Studio 2010 (since all what a client would require to run the system is a browser)
Styles	Cascaded Style Sheets (CSS)
Data	SQL Server CSV text files

4.2.8 The final model

All plans described previously can be combined in one general overview of the expected functionalities from the new system. Figure 18 sketches all features to get a better intuition of the system.

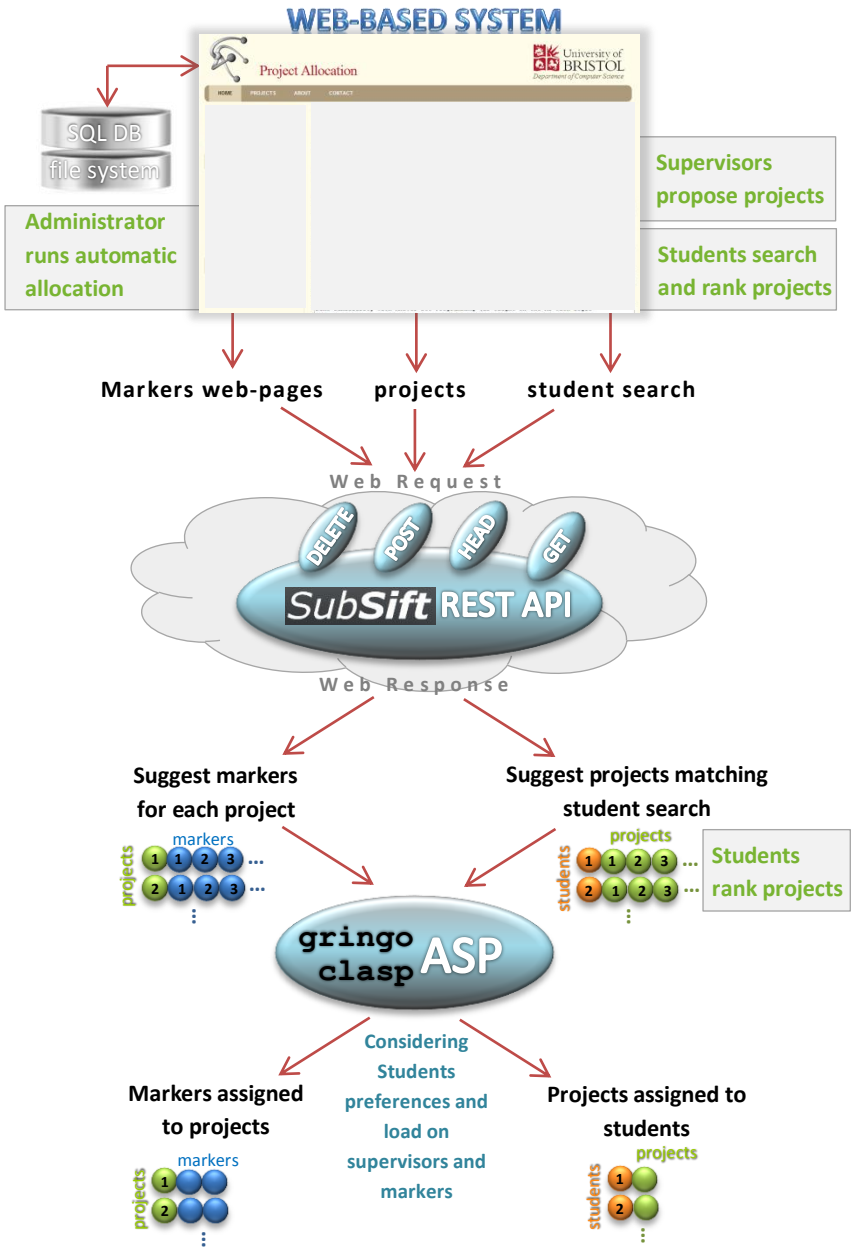


Figure 18 - System Design Overview

4.3 Building the Blocks

Software implementation phase mainly involves mapping the plans into the real system. Therefore, this stage would include building the planned system architecture first and then accommodate all components in one integrated environment. Implementation details are explained in the following sections.

4.3.1 Implementing the Architecture – Fornt-end Layer

Recall that the system architecture is composed of three layers, each of which has different components. The front-end layer provides a workspace for each of the system users. Each workspace is discussed in the sections below.

The front-end is developed in C# (a java-like object oriented programming language). Technically, each page represents a class in the code (called *code behind*) and it is associated to an HTML source code for its web interface.

Supervisor Workspace

As the first step in the whole cycle, each supervisor proposes projects through the page in figure 20. The supervisor would be recognised if the system is linked to the university login system. For this project, staff id can be selected manually from a drop-down list where this selection brings all information related to this supervisor as his name and proposed projects. Supervisors can modify projects as well as deleting unneeded entries. Projects Ids are generated automatically by the system.

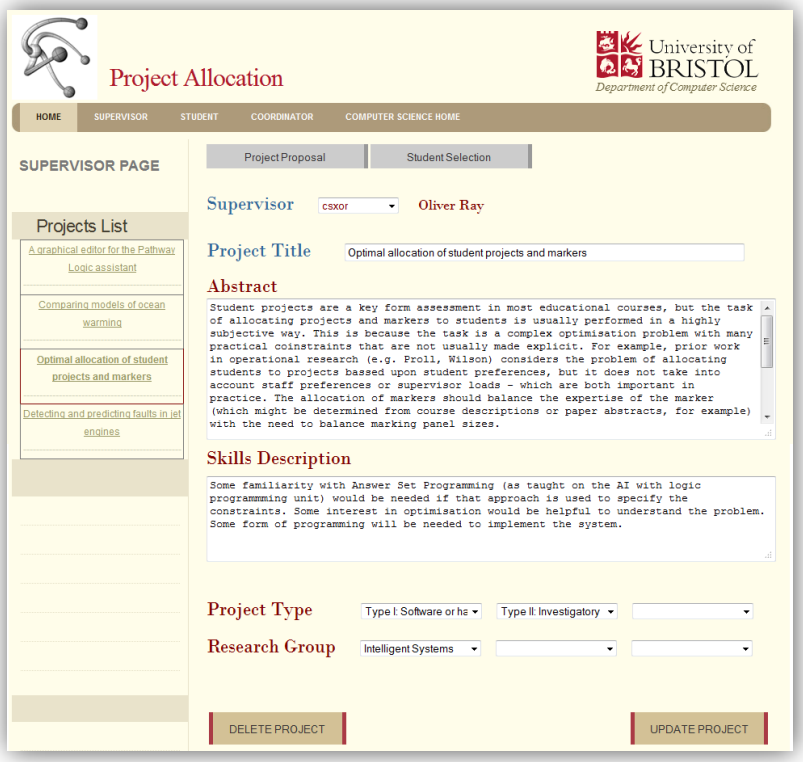


Figure 19 – Supervisor page

Student Workspace

When projects gets published, students can easily find appropriate projects without going throw a long list of projects and even with no need to open the links of any project to read their abstracts. Following is a description of how students find and get allocated a project.

Figure 21 shows students page for finding a project. First, any student selects his/her user name from a dropdown list then the system displays student full name next to the dropdown list (in reality this will be replaced with login system so the system will recognises students from their user names and passwords).

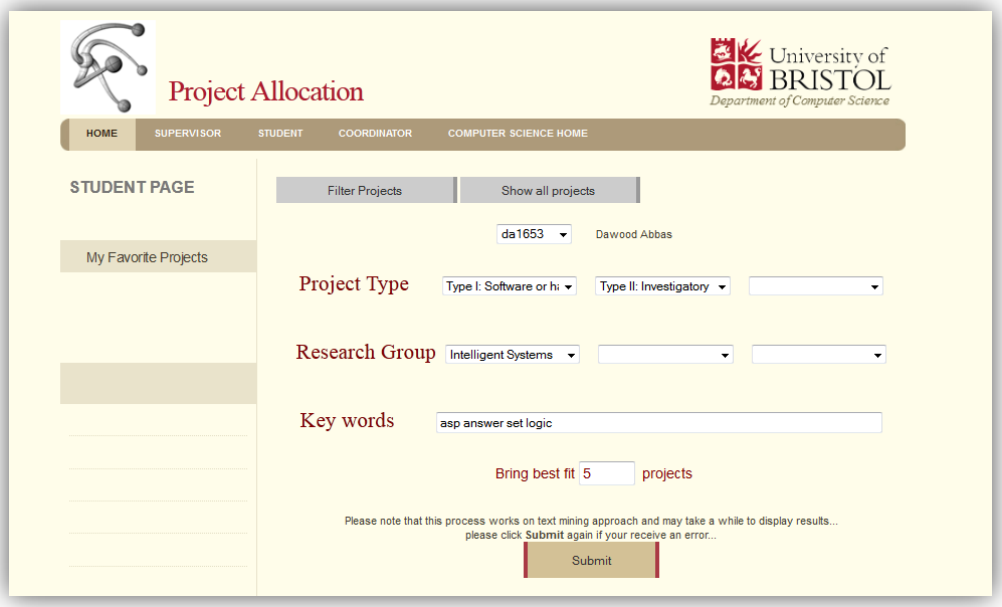


Figure 20 - Searching for projects

Student can choose from other dropdown lists to specify preferred project type and research group, followed by entering some keywords and indicating the number of projects he/she wants the system to display out of entire list of projects. Consequently, the system will discard all those projects that are not relevant to the current student.

By clicking on the 'Submit' button, the system will fetch projects that match student search terms, and projects will be displayed in ascending order which means that the first project is the most one that fits the student's search and the second is after that and so on.

The process of finding and sorting projects for any individual student has been implemented by using SubSift. SubSift implementation will be discussed later when explaining Controlling layer implementation.

Student search result will be displayed on the current page (figure 22) in such a way that the first project (which is the closest to student's search) will show both its title and abstract expanded. While other projects will be (collapsed) displaying only their titles and get expanded by clicking on their titles (one at a time). This style behaviour is called Accordion control and it is discussed in the section about Styles. This will help students for viewing projects – as mentioned - with no need for leaving the current page.

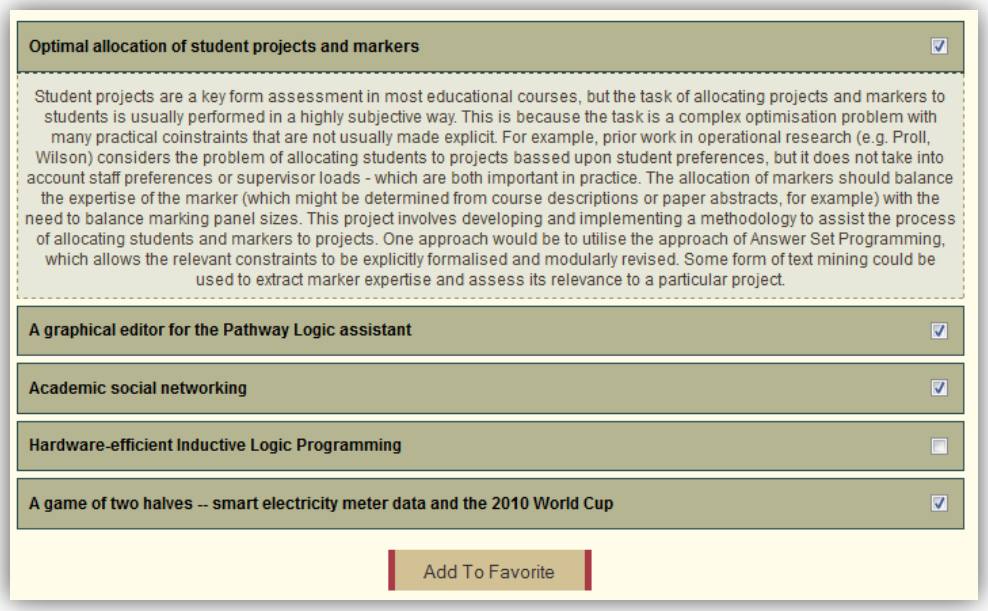


Figure 21 - SubSift Result for student projects search

By ticking any project checkbox and pressing "Add to Favorite" button a list of selected projects will be displayed on the left side of the page where the student can rank projects and submit them.

When all students submit their ranked lists of projects then the coordinator will assign each student a project (explained in the next section).

Coordinator Workspace

Project coordinator is the person responsible to coordinate the processes of assigning projects to students and markers to projects. To perform tasks related to his responsibilities, he is dedicated a page in the new system including relevant features to his tasks. Since these tasks relate totally to SubSift and ASP implementation, they are explained in separate sections thoroughly.

Getting the intuition about how the interface looks like, would help to understand the main part of this chapter; the implementation of SubSift text matching and ASP optimal allocation. These two implementations lies in the controlling layer and are discussed individually after getting a summary about the back-end layer of the architecture.

4.3.2 Implementing the Architecture – Back-end Layer

The back-end layer in the system architecture comprises the system database, data files, and ASP constraints. ASP constraints component is explained in the following section in the context of the optimisation problem solution. In this project, a light-weight relational database is developed for managing and relating

various data to each other. Although data are supplied to the system as plain text files, they are imported into database tables to achieve more control over data.

Visual Studio comes with an internal built-in SQL server called SQLEXPRESS in addition to the required tools for managing this server. Databases can be created with any integrity constraints required between tables. Relationships can also be set between tables for aggregating integrity constraints during various database operations.

This system is intended to serve a number of users and is not restricted to provide read-only information. In this system, concurrent access to same data with editing rights as well as reading is a requirement. Despite all other features that an SQL server provides like data indexing, managing concurrent access would have been very difficult with text files. Additionally, ASP.NET has various nice controls that work seamlessly with SQL server like *DataGrid*. All these reasons can be added to security management that ASP.NET provides for accessing SQL databases. Connection specification is stored in the system *Web.config* configuration file and is accessed through *ConfigurationManager* class. Connection is opened using *WindowsAuthentication* mode, while in a real system, this can be replaced with an SQL server authentication by supplying a user name and password.

To make more focus on the main problem this project is trying to solve, data import from text files to database tables has been done manually; i.e. data are copied directly to database tables. Of course in a real system, a page for importing data into the system would be provided in the coordinator workspace. Database tables structure is available in appendix I.

4.3.3 SubSift implementation

SubSift text matching requires preparing the text items before they can be matched. The actual matching is performed between these text profiles. Information supplied to SubSift takes one of two forms: actual text or bookmark to text. Developing general methods to handle these two types of text would make this system applicable for any text matching problem.

As planned, the set of methods to be invoked for preparing text for matching need to be developed in ASP.NET. In this project, a general class *SubSift* for keeping all SubSift related methods is developed in C#.

For calling SubSift methods, two versions of method *SubSiftMethod* are created in this class: one for requesting SubSift without parameters, and the other with parameters.

```
public static bool SubSiftMethod(string url, string method)
```

```
public static bool SubSiftMethod(string url, string method,
                          string parameterName, string parameterValue)
```

In these methods, the class uses a *WebRequest* object for the specified URI. It sets the protocol *method*; i.e. http method, to the required method like POST for instance. The WebRequest object has headers associated with it and the Token for SubSift server authentication needs to be added to the request headers. Finally, the request gets executed by calling its *GetResponse* method. All these steps are encoded in C# as following:

```
WebRequest myReq = WebRequest.Create(url);
myReq.Headers.Add("Token", "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx");
myReq.Method = method;
WebResponse wr = myReq.GetResponse();
```

For more security, the token can be stored in the ASP.NET web application configuration file: *Web.config* and then it can be retrieved by using *ConfigurationManager* class.

In the case of having parameters to be sent with the request, a *StreamWriter* instance for the request stream is used to write the parameter with its value to the stream as following:

```
string postString = parameterName + "=" + parameterValue;
StreamWriter requestWriter = new StreamWriter(myReq.GetRequestStream());
requestWriter.Write(postString);
requestWriter.Close();
```

In some cases, it is required to read the returned result from executing the *WebRequest*, like the case in getting the matched items for instance. In this case, *GetResponse* object returned value can be stored in a *WebResponse* object. Then the response stream is stored in a *Stream* object to create a *StreamReader* for it. The reader reads the stream and returns a string (XML result in our case) of the read content. These steps are coded in method called *SubSiftGetMatchedItems* as following:

```
Stream receiveStream = wr.GetResponseStream();
StreamReader reader = new StreamReader(receiveStream, Encoding.UTF8);
string content = reader.ReadToEnd();
```

Now, knowing how to invoke SubSift methods, we can move to see how this code is generalised to serve any matching application.

Preparing SubSift Text

To prepare the text for SubSift matching, two methods are created to work for either forms: actual text or bookmark to text.

For the actual text form, the planned steps are implemented as following:

```
public static string prepareSubSiftText(string textId, string text)
```

```
// Create Documents Folder
SubSiftMethod("http://subsift.ilrt.bris.ac.uk/user/documents/"+textId,
"POST");

// Create document Items
SubSiftMethod("http://subsift.ilrt.bris.ac.uk/user/documents/"+textId+"/
items", "POST", "items_list", text);

// Create profiles
SubSiftMethod("http://subsift.ilrt.bris.ac.uk/user/profiles/"+textId+"/f
rom/"+ textId, "POST");
```

And for the bookmark form, the planned steps are implemented as following:

```
public static string prepareSubSiftBookmarkText(string textId, string
bookmarkItems)
```

```
// Create Bookmark Folder
SubSiftMethod("http://subsift.ilrt.bris.ac.uk/user/bookmarks/" + textId,
"POST");

// Create Bookmark Items
SubSiftMethod("http://subsift.ilrt.bris.ac.uk/user/bookmarks/" + textId
+ "/items", "POST", "items_list", bookmarkItems);

// Create documents folder
SubSiftMethod("http://subsift.ilrt.bris.ac.uk/user/documents/" + textId,
"POST");

// import document items from web pages (bookmarks)
SubSiftMethod("http://subsift.ilrt.bris.ac.uk/user/documents/" + textId
+ "/import/" + textId, "POST");

// create profiles
SubSiftMethod("http://subsift.ilrt.bris.ac.uk/user/profiles/" + textId,
"POST");
```

The plan for matching the two texts is generalised as following:

```
public static string match(string firstTextId, string secondTextId)
```

```
// Matching first text and second text Profiles
SubSiftMethod("http://subsift.ilrt.bris.ac.uk/user/matches/" +
 firstTextId + "_" + secondTextId + "/profiles/" + firstTextId +
"/with/" + secondTextId, "POST");

// Get Matched results
// profiles_id parameter is to select which text to group by
SubSiftGetMatchedItems("http://subsift.ilrt.bris.ac.uk/user/matches/" +
firstTextId + "_" + secondTextId + "/items?profiles_id=" + secondTextId,
"GET");
```

Everything is prepared now for using it, but how all this is used?

For markers-projects matching, it is simply as following:

```
// Prepare markers
prepareSubSiftBookmarkText("markers", markersList);

// prepare projects
prepareSubSiftText("projects", projectsText);

// match
string SubSiftResult = match("markers", "projects");
```

SubSiftResult will have the xml matching result which can be processed as required for reading. In this project, the result is loaded into an *XmlDocument* object for easier processing. With this object, xml nodes are retrieved and they are written to a file in order to be viewed later on the page.

Similarly, matching projects to student search term is performed as following:

```
// prepare projects
prepareSubSiftText("projects", projectsText);

// prepare search
prepareSubSiftText("search" , studentSearch);

// match
string SubSiftResult = match("projects", "search");
```

Only one thing left to be considered here, the format of the supplied text. There is a specific format for the text required by SubSift. For instance, *projectsText* is required to have all projects as one string with each project record in a line. Each line should be of the format <*project-id*>, <*project-text*>. For *search*, there will be one item (one line); e.g. *s1,* <*search-text*>.

Bookmarks are required to have the same format as those of projects and search, but this time the bookmark URL addresses will replace the text; i.e. <*marker-id*>,<*marker-URL*>.

All these functionalities are available as one of the features in the coordinator workspace as in figure 23.

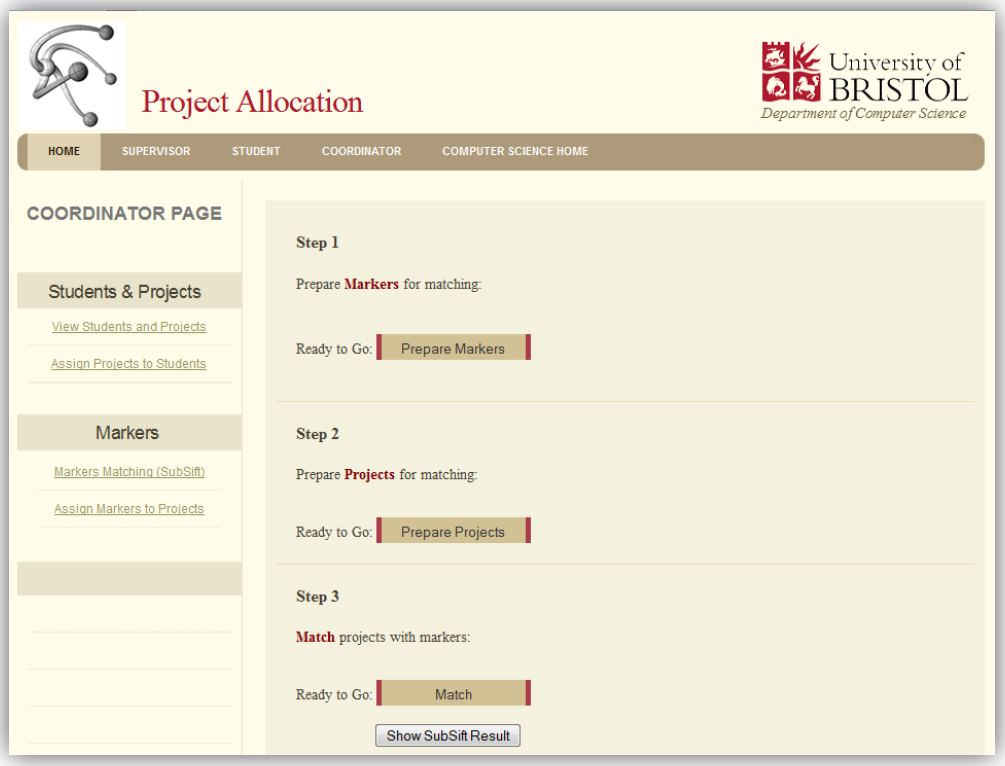


Figure 22 - Coordinator page for markers-projects matching

4.3.4 ASP Optimisation – behind the seen

The toy program can be enhanced now to build the real program for solving the optimal allocation with constraints. In addition to the condition of not exceeding the load on each marker, marker-project allocation process has the criteria of not assigning a supervisor to mark his project. Keeping these two constraints in mind, in addition to the requirement of optimising the allocations so they are closer to lower ranks (most preferred), we will go through the process of implementing the optimality.

Allocating projects to students

To run a complete version of the ASP program, all required predicates need to be created. For allocating projects to students, the predicates required are summarised in table 8:

Table 8 - Fact predicates required for allocating projects to students

	Predicate	Source
1	*rank(<student-id>, <project-id>, <rank>)*	*StudentRank* database table
2	*project(<project-id>,<supervisor-id>)*	*Project* database table
3	*student(<student-id>)*	*Student* database table
4	*staff(<staff-id>, <load>)*	*Staff* database table

All these ground facts are written to a *.pl* file. Then all other ASP code statements need to be appended to the same file. The assignment statement becomes:

```
1{assign(S,P,V):project(P,V)}1 :- student(S).
```

Some constraints need to be applied to ensure a one-to-one assignment of students to projects which will make sure no project is assigned to more than one student, and no student is assigned more than one project:

```
:- assign(S1,P,_), assign(S2,P,_), S1 != S2.
:- assign(S,P1,_), assign(S,P2,_), P1 != P2.
```

To manage load on supervisors, and having load specified for each supervisor, the assignments can be counted for each supervisor and the count should be restricted to this supervisor's pre-set load value. This will prohibit any assignment leading to the violation of this condition. Following is the constraint for managing load:

```
:- assign(_,_,V), staff(V,L), B = #count{assign(_,_,V)}, B > L.
```

As planned, we have two optimisations that we need to apply by minimisation; one is for minimising the worst rank among all assigned projects:

```
% student S has badness R under the assignment
bad(S,R) :- assign(S,P,_), rank(S,P,R).

% some student S2 is worse off than S1
worse(S1) :- bad(S1,R1), bad(S2,R2), R1<R2, S1 != S2.

% R is score of worst off student
worst(R) :- bad(S,R), not worse(S).

#minimize [worst(R)=R].
```

The other optimisation is to minimise the total rank of all assigned projects:

```
totalSum(T) :- T = #sum [bad(_,R)=R].
#minimize [totalSum(T)=T].
```

A complete version of the code for 12 students, 24 projects, and 6 supervisors is available in appendix II.

Now, the ASP *pl* file is ready for execution using both gringo and clasp. The command line statement for running the program (here is named Predicate.pl) with some clasp options for enhancing the optimisation is shown below:

```
gringo Predicate.pl | clasp --opt-value=15,15
```

Where –-opt option means that the optimum values in this program should start from the specified values; i.e. 15.

This is how it can be run in a command line program, but for integrating with our system, a method *runCommand* is developed for executing this command using *Process* object with setting its properties to start running the program in command line. The execution output is stored in a file as following:

```
process.StartInfo.Arguments = "/c gringo " + inputFile + " | clasp > "
+ outputFile;
```

The system then accesses the results file and process it using string processing operations to extract results and store them in the database file *ProjectAssignement* for easy access. The reader can refer to the relevant source code in appendix IV.

These functionalities are embedded behind a friendly user interface in the coordinator workspace as in figure 24.

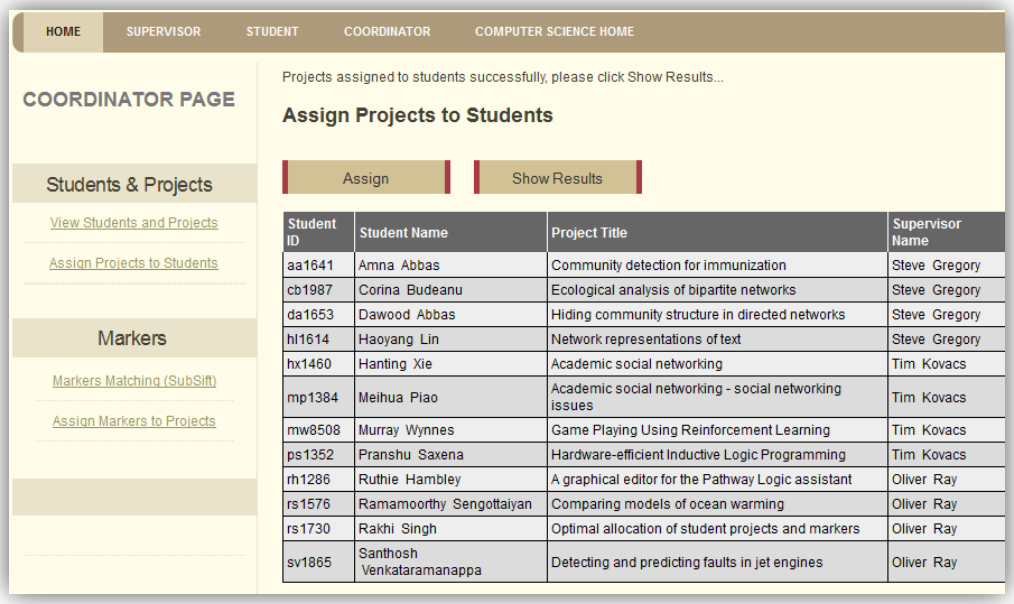


Figure 23 - Coordinator page for assigning projects to students

Allocating markers to projects

This allocation has few differences than the previous one. Although the predicates seem to be the same, they are slightly different in the source of the information. The predicates used in this problem are outlined in table 9 below.

Table 9 - Fact predicates required for allocating markers to projects

	Predicate	Source
1	*rank(<project-id>, <staff-id>, <rank>)*	SubSift xml result
2	*project(<project-id>,<supervisor-id>)*	*ProjectAssignment* database table (as only those projects that are assigned need to be allocated markers)
3	*staff(<staff-id>, <load>)*	*Staff* database table

All ranks are obtained by extracting information from xml nodes. Here we can consider these ranks as projects preference of markers. The same technical processes is applied here where all the ground facts and ASP code is stored in one file for execution. Refer to appendix III for the program file.

Two (distinct) markers are required to be assigned to each project instead of one. In this case, two assignment statements are used for the two markers:

```
1{assign(1,P,M):staff(M,_)}1 :- project(P,_).
1{assign(2,P,M):staff(M,_)}1 :- project(P,_).
```

To ensure these markers are different and marker 1 is more suitable than marker 2 (has lower rank in SubSift result), the following two statements are used:

```
:- assign(1,P,M), assign(2,P,M).
:- assign(1,P,M1), assign(2,P,M2), rank(P,M1,R1), rank(P,M2,R2), R1>R2.
```

Here there is no restriction on assigning the same marker to different projects. Multiple assignments of projects to markers should be performed with load balancing - in the same way as of supervisors - as following:

```
:- assign(_,P,M), staff(M,L), B = #count{assign(_,_,M)}, B > L.
```

This problem involves the constraint of not assigning a marker to a project that he/she supervises and this is encoded as follows:

```
:- assign(_,P,S), project(P,S).
```

Finally, the optimisation is achieved with minimising both marker 1 and marker 2 total and individual ranks as following:

```
bad(N,P,R) :- assign(N,P,M),rank(P,M,R).
worse(N,P1) :- bad(N,P1,R1), bad(N,P2,R2), R1<R2.
worst(N,R) :- bad(N,P,R), not worse(N,P).

totalSum(N,T) :- T = #sum [bad(N,_,R)=R].
#minimize [totalSum(2,T)=T].
```

```
#minimize [totalSum(1,T)=T].

#minimize [worst(2,R)=R].
#minimize [worst(1,R)=R].
```

The program is run through the coordinator user interface and the results of the assignments get extracted and displayed for the coordinator as in figure 25.

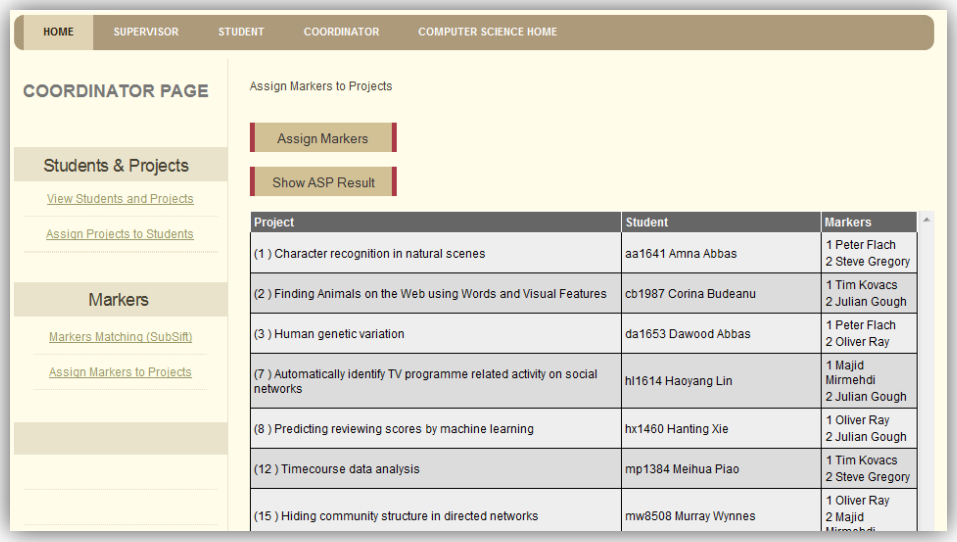


Figure 24 - Coordinator page for assigning markers to projects

4.3.5 Decorating with CSS

The project uses ASP.NET best features to achieve a more appealing look of the user interface. To maintain consistency between pages, and to reduce code duplication, one master page template is developed and applied to all system pages. The master page references a Cascaded Style Sheet (CSS) file where all styles are stored. The CSS file include simple styles like font, colours, margins, etc. it also includes more advanced styles like clicking or mouse hovering behaviours.

4.4 A guide for deployment

To bring the system developed in this project to life, it needs to be deployed to a production web server. The system folder that contains all source code and other resources related to the system needs to be moved to the server using one of the available deployment scenarios. The Web server (IIS server) needs also to be configured by creating a *Virtual Directory* that points to the physical folder location. Deployment phase is out of the scope of this project, but this section provides links to online resources as guidance for deploying the application. Once deployed to server, users can access the application through its URL and

there will be no need for any other configuration from client side. Following is an outline of some resources from Microsoft official documents library.

Overview of Deploying ASP.NET Applications (IIS 6.0)
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/64ac3577-7746-4fbd-a5e6-e50bdecf7a24.mspx
Deploying Your Site Using Visual Studio (C#)
http://www.asp.net/web-forms/tutorials/deployment/deploying-web-site-projects/Deploying-Your-Site-Using-Visual-Studio-cs
Web Application Project Deployment Overview for Visual Studio and ASP.NET
http://msdn.microsoft.com/en-us/library/dd394698.aspx

System database needs also to be moved to production and this can be done by storing the database as scripts (a feature available in SQL server) and these scripts can be easily executed within an empty database in the production SQL server. Theses scripts are all SQL queries for creating all objects in the database as well as inserting data into the created tables.

5 Results

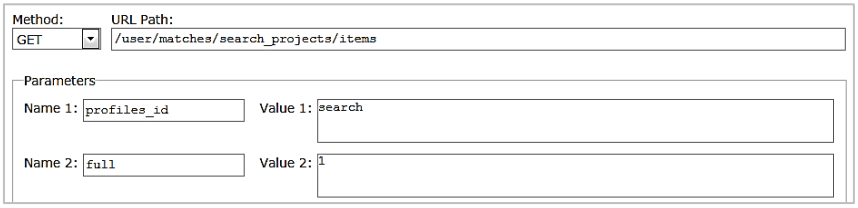
Although the developed system can be used with any size of data, but the time required to obtain the allocation results varies. This is due to the nature of the problem in hand which is NP-hard. This means that the more complex the problem is the more time will be required to be solved. This section goes through some scenarios and analyses the results obtained from each. These scenarios are selected from the whole cycle of the project and marker allocation. All these scenarios are outlined below with discussions available for critical tasks only.

Note:

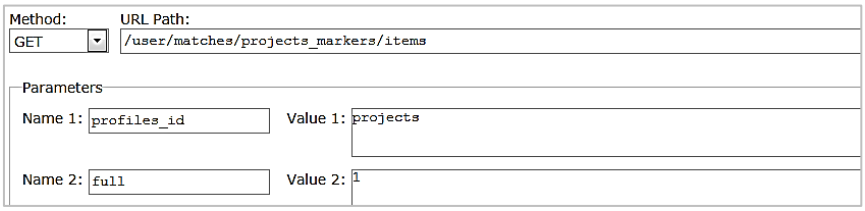
Scenarios are all done for the following setup:

24 projects, 12 students, and 6 staff members.

5.1 Scenarios Group 1: Supervisors Tasks	
Scenario	Propose a project
Procedure	Supervisor menu > select Supervisor id > fill in the fields > Submit
Result	Project added successfully to database and shows in the supervisors own projects list, coordinator list, and student search list
Scenario	Edit or delete a project
Procedure	Supervisor menu > select Supervisor id > select a project under Projects List > make required changes > Update or Delete
Result	Project modified/deleted successfully
5.2 Scenarios Group 2: Student Tasks	
Scenario	Look for a project
Procedure	Student menu > select Student id > select filters and/or enter search words > specify number of projects to show > Submit
Result	Correct number of projects comes up in the accordion list.
Discussion	This list is a result of SubSift matching between projects and

	search terms. Although can be assessed by eye, to make sure the displayed projects are really the best for this search, following procedure is performed: From online SubSift REST API Explorer[1] > select method as GET > enter URL: */<user>/matches/search_projects/items* > add two parameters: *profiles_id* with value *search*, *full* with value 1  This will bring up the entire match matrix grouped by the search term. The scores for all terms are displayed with total score of each match entry. Comparing the scores, it showed really that the best matches are same as those displayed on the student page. Frequency of the terms is really important for this matching, thus supervisors are required to enter sufficient information that are relevant to each project to make the matching more accurate. Some non-relevant results obtained while testing were due to shortage of information (or low-relevancy) in the project titles, abstracts and/or descriptions. Search code takes from 10 to 14 second to bring the final result.
Scenario	Select and rank projects
Procedure	Search result > tick the required projects > do another search > tick projects > enter the rank of the selected project > leave or delete those not required > Submit
Result	Selected projects gets added to student list and submitting the list succeeds with confirmation message displayed.
Scenario	Submit another rank list for a student already submitted a list
Procedure	Student menu > select Student id > select filters and/or enter search words > specify number of projects to show > Submit
Result	Search result shows without "Add to Favourite" button, so he cannot submit again which is correct.
5.3 Scenarios Group 3: Coordinator Tasks	
Scenario	Assign projects to students

[1] http://subsift.ilrt.bris.ac.uk/demo/explorer

Procedure	Coordinator menu > click "Assign projects to students" link > Assign > after confirmation message, click Show Results
Result	A list of all students is displayed with an allocated project and supervisor name displayed
Discussion	This list is a result of ASP code for optimal allocation. It shows no project assigned to more the one student, and each student is assigned one project. Among the 12 students, the number of assigned students to supervisors doesn't exceed the pre-set loads. ASP produces intermediate answers while it looks for the optimum and the program correctly displays the optimum solution found. To make sure the program displays the correct solution, I executed ASP program in DOS command line and checked the output which matches the one displayed in the coordinator page.
Scenario	Match markers and projects
Procedure	Coordinator menu > click "Match markers to projects" link > click prepare markers > click prepare projects > then click match > Show SubSift Result (if received error message, click button again)
Result	Sometimes errors appear, but when clicking again it succeeds and a table of 12 projects with 6 markers next to each is displayed
Discussion	The table displays SubSift result of matching markers and projects. Errors are handled by using Exceptions while calling SubSift methods. The reason of these errors is due internet connectivity to SubSift server. The result of matching seems to show suitable markers, but to make sure, the same procedure as of search matching has been followed: From online SubSift REST API Explorer > select method as GET > enter URL: */<user>/matches/projects_markers/items* > add two parameters: *profiles_id* with value *projects*, *full* with value 1  This xml result (match matrix) shows markers with highest score and hence suitability to each project first. Most of the time is consumed on preparing the markers, where their web pages need to be fetched into SubSift server. It takes around 5 minutes.
Scenario	Assign markers to projects

Procedure	Coordinator menu > click "Assign markers to projects" link > Assign > after confirmation click Show Results
Result	A list of all 12 projects is displayed with two allocated markers and student name displayed
Discussion	This result of ASP code for optimal allocation shows that among the 12 projects, the number of assigned projects to markers doesn't exceed the pre-set loads. I also executed the ASP program in DOS command line and checked the output which matches the one displayed in the coordinator page.

6 Evaluation

The purpose of this chapter is to explain how the project was assessed and to which degree the project's objectives are fulfilled. Following are some methods used for evaluating the system.

6.1 Measuring the optimality

Taking the academic year 2011/12 MSc students (121 students) doing the final project, I assumed that each student's current project is the most preferred by him (should have rank 1). Based on this assumption, a scenario is implemented for allocating 182 projects to these 121 students by creating a sample rank list for a selected number of students (12) with the currently allocated project as first choice. The rest of each rank list is selected randomly from available proposed projects (e.g. rank 2, rank 3 …).

Then running the system on this setup, the output was studied and those selected students allocations were checked. The algorithm provides 100% accuracy (12/12) in assigning projects to students.

From this result we can realise that if all students first rank are different from each other, the system will assign all students to their first choice project which is the desired result. Although this situation would not normally happen in reality, i.e. having students with distinct project selections, proving the optimality of the algorithm in such situations would be necessary.

Shifting to the real world situation, where students preferences overlap, we need to prove the optimality of the new allocation process. This means that if there are two (or more) students ranked the same project as their first choice in their project ranked list, then the system will give one of them the first choice and the other(s) second choice (or the next best possible choice) as shown in the toy example.

To check this situation in a real world scenario, I applied the algorithm on 15 students who had not managed to select their projects by the deadline this year. I created a rank list for each of them (with some overlaps) and considering only those projects not assigned to any students; i.e. 106 students were assigned projects and hence only 76 projects left out of the 182. I updated supervisors maximum loads to consider the projects already assigned, and those are to be assigned in this experiment. I then run the allocation algorithm on this setup and the results were satisfying – see figure 25. Note that the run took around 2 seconds on average.

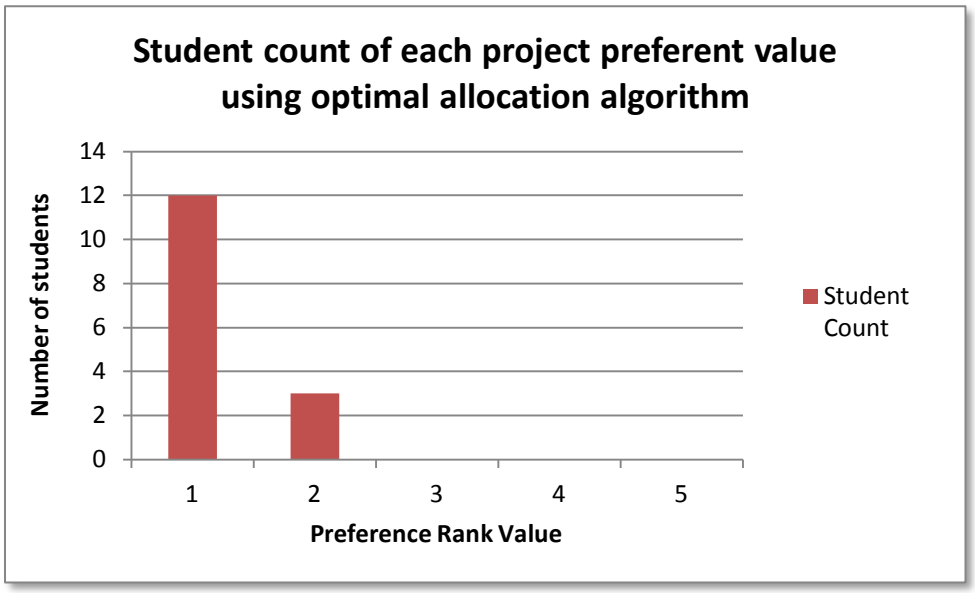


Figure 25 - Optimal allocation result for student-project allocation

The output of this experiment gives an optimum value of rank 2 as a maximum of any allocated project, and a value of 18 as a total of all allocated projects ranks. Altering supervisors loads changes the result of total ranks, but still it gives 2 as maximum individual rank.

Similar process was applied for assigning markers to projects. This time, I applied the process of assignment for 6 markers and 12 projects. We need to select two markers for each project. For the process of markers selection we want to take into consideration all those constraints mentioned in the previous section such as load balancing on each markers and the marker of any project should not be its supervisor. The output result of markers assignment was as required, all selected markers for any project obeyed the criteria, there are no markers for any project which they are also supervisor of these projects, and considering load on markers as required. Table 10 displays each project with assigned two markers.

Please note that markers selection is performed only on 6 supervisors and the system find the best markers for each project after applying constraints on the result of SubSift rank where also SubSift ranked only these 6 markers to each project. Therefore it could be that the result of this assignment is not precise regarding markers expertise related to each project, but as we only have this list to select from (at the moment) for assigning to available projects. Consequently, in reality we have larger list of markers and projects and SubSift result will be the ranks of all markers regarding any project so the markers assignment will be more precise regarding their expertise.

Table 10 - Marker-project allocation result

Project	Student	Markers
(1) Character recognition in natural scenes	aa1641 Amna Abbas	1 Peter Flach 2 Steve Gregory
(2) Finding Animals on the Web using Words and Visual Features	cb1987 Corina Budeanu	1 Tim Kovacs 2 Julian Gough
(3) Human genetic variation	da1653 Dawood Abbas	1 Peter Flach 2 Oliver Ray
(7) Automatically identify TV programme related activity on social networks	hl1614 Haoyang Lin	1 Majid Mirmehdi 2 Julian Gough
(8) Predicting reviewing scores by machine learning	hx1460 Hanting Xie	1 Oliver Ray 2 Julian Gough
(12) Timecourse data analysis	mp1384 Meihua Piao	1 Tim Kovacs 2 Steve Gregory
(15) Hiding community structure in directed networks	mw8508 Murray Wynnes	1 Oliver Ray 2 Majid Mirmehdi
(16) Network representations of text	ps1352 Pranshu Saxena	1 Tim Kovacs 2 Oliver Ray
(19) Game Playing Using Reinforcement Learning	rh1286 Ruthie Hambley	1 Majid Mirmehdi 2 Julian Gough
(20) Hardware-efficient Inductive Logic Programming	rs1576 Ramamoorthy Sengottaiyan	1 Peter Flach 2 Steve Gregory
(23) Optimal allocation of student projects and markers	rs1730 Rakhi Singh	1 Tim Kovacs 2 Majid Mirmehdi
(24) Detecting and predicting faults in jet engines	sv1865 Santhosh Venkataramanappa	1 Peter Flach 2 Steve Gregory

Figure 26 shows the assigned markers' SubSift ranks for the results outlined in table 10. The figure shows the trend of optimising marker 1 and marker 2 ranks and obviously shows that marker 1 has the higher priority in optimisation.

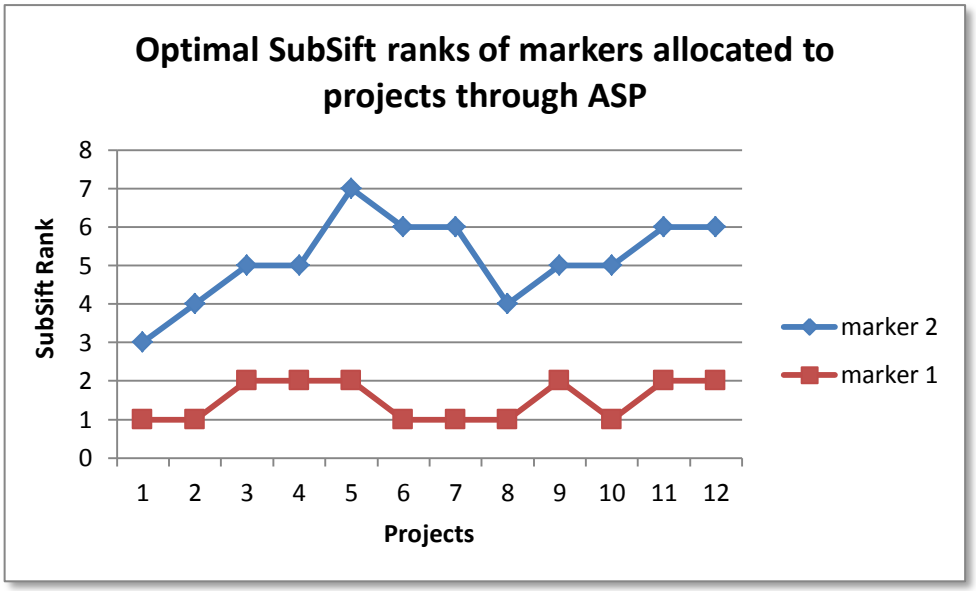


Figure 26 - SubSift ranks of markers allocated using ASP

Figure 27 also shows the count of projects in each marker rank.

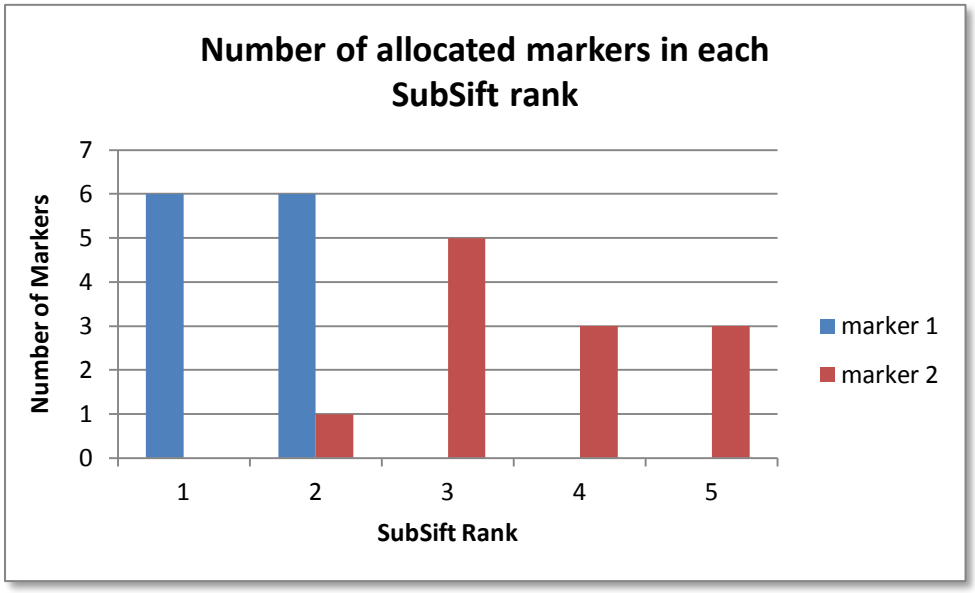


Figure 27 - Number of markers with respect to SubSift Rank

The optimisation problem is hard and these results prove that this project has achieved the objective of optimal allocation.

6.2 Process cost appraisal

Evaluating the cost of the allocation process (i.e. time and effort) is a critical measure for the overall evaluation of project quality. This can be assessed by comparing the time and effort spent on the process before, and after implementing the system. For the manual system, estimations were acquired from students regarding the current project allocation process.

For coordinators, they normally need to encourage students for selecting their projects before the deadline, and then they would require long time to negotiate with students and supervisors to finally allocate projects to those students who have not selected a project by the deadline. They think that such process is complex and would create unnecessary work on the coordinators.

For students, a sample was selected and interviewed for checking and comparing the time required for project selection between the current and the new system, results are shown in figure 28.

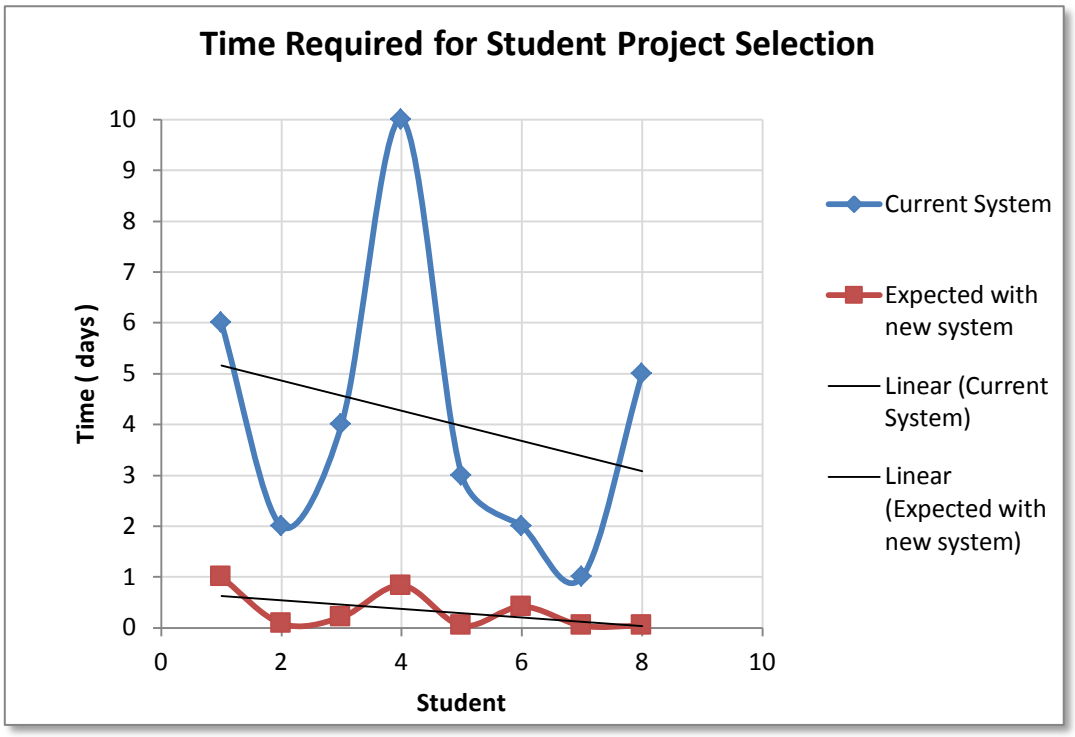


Figure 28 - A Comparison of time required for selecting projects between the current and the new system

For a real system, such measures cane be precisely obtained automatically by embedding time counters in the various steps of the system.

This shows that cost is expected to be reduced dramatically in a real system, proving that the new process will address the limitations of the current system.

6.3 System Users Satisfaction

Students level of satisfaction was measured by implementing satisfaction checks from within the same system. A sample from users was selected to work on and test the system. The experiment was applied on six students to try the new system for project selection. In this experiment, the system was provided to the students and they were asked to imagine that they want to do a new project and need to select a project from available published list. Then their feedback and experience in searching and finding a suitable project was taken by answering the following questionnaire (table 11).

Table 11 - Students questionnaire for assessing users satisfaction

Question	S1	S2	S3	S4	S5	S6
Did the current system consume your time more than necessary for finding the right project?	Yes	Yes	Yes	Yes	Yes	Yes
Did you go through projects which are completely not relevant to your expertise?	Yes	Yes	Yes	Yes	Yes	Yes

Do you think the current system is better to be replaced by new system which provides you only those projects which are suitable regarding your expertise?	Yes	Yes	Yes	Yes	Yes	Yes
Do you think the new system will reduce the time required for finding a suitable project?	Yes	Yes	Yes	Yes	Yes	Yes
Do you think the new system will reduce the effort required to be spent by a student for finding a project?	Yes	Yes	Yes	Yes	Yes	Yes
Has your current project appeared in your search?	Yes	Yes	Yes	Yes	Yes	Yes
If yes please mention its order in the list?	1	5	33	18	46	9

Although the order of each student actual project in the suggested list varied (figure 29), but still most of them found their actual projects among the first 20 projects provided by the system, and all of them acknowledged that finding a suitable project among this list is more easier and require less effort than previous system and also most of them found that all the suggested projects are related to what they are looking for.

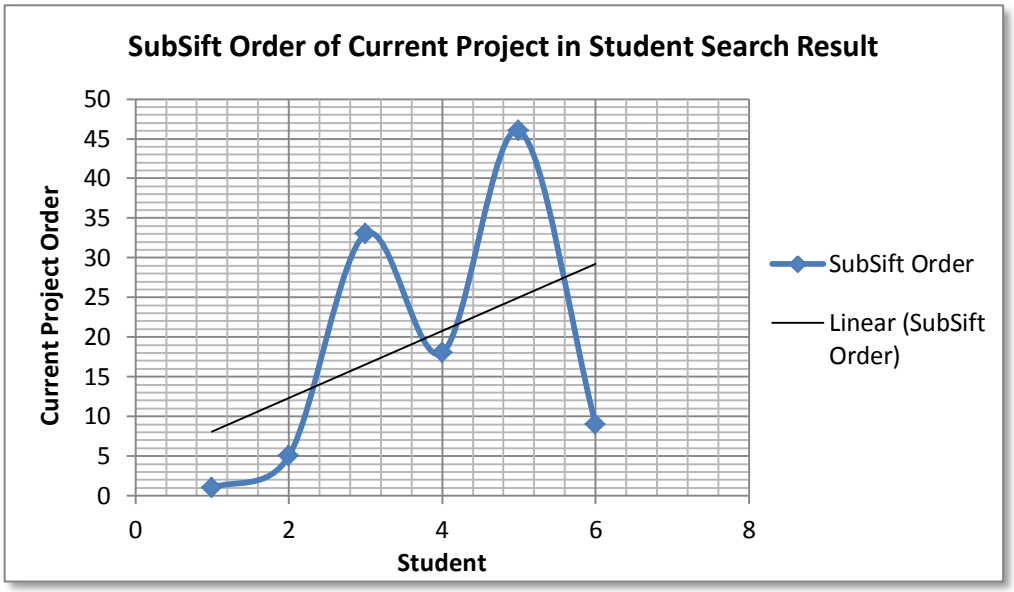


Figure 29 - SubSiftorder of current projects obtained in student search results

We can improve the process of finding a suitable project and make it even more easier and consuming less effort of students to discover a right project in a minimum time by only:

- Advise supervisors to include some key words in their proposed projects;

- Improving students' knowledge about the way they can search and the type of terms that they can use by providing them a sample example and a short description on the student main page.

Summary:

the outcome of this experiment and the result of what I realised from their verbal answers is that all of them agreed that the current system is not effective and it was very hard to select a suitable project. All of them were happy with the result of their search and the suggested list of projects. They all agree that it is more practical and more efficient to select a project from the suggested projects list which is close to their expertise and interest.

Conclusion and future work

This project provides a solution to one NP-hard problem; optimal allocation. It exploits the power of constraint logic programming via using Answer Set Programming for the allocation process. It uses ASP for applying an algorithm adapted from Bottleneck assignment algorithm originated by (Proll, 1972). It applies the developed solution to allocate final projects to MSc students in the University of Bristol. It also attempts to solve another problem using the same algorithm, but this time, with the assistance of another powerful textual matching application; SubSift. SubSift has been used to rank markers with respect to projects. SubSift ranking is passed to ASP to achieve optimal allocation of markers to projects.

Text matching is a useful technique that has many applications. This project realised that and developed a general API linked to SubSift to be used for any text matching problem. All the solutions developed in this project were evaluated and proved satisfaction.

As of the general SubSift API, further work can be continued to achieve a general optimal allocation to solve any allocation problem. These general solutions can be enhanced also by having a general friendly user interface. The user interface can also be empowered by features like master data import and data manipulation in the coordinator workspace. Supervisor workspace can have features like managing the list of students they supervise and being able to assign them manually or contact them by email.

Security of the system can be improved via developing a security system for users' login, or alternatively, the system can be integrated to the university Single Sign On system.

Despite that the problem is NP-hard and would require a relatively long time for achieving final result, opportunities to enhance the performance of the system can also be pursued.

Finally, combining Constraint Programming via ASP, and textual matching via SubSift, and adopt them into wider range of applications would have much impact on the optimality of the outcomes.

References

Gale, D. & Shapley, L. S., 1962. College admissions and the stability of marriage. *The American Mathematical Monthly,* pp. 9 -15.

Arora, S. & Barak, B., 2009. *Computational Complexity: A Modern Approach.* New York: Cambridge University Press.

Berry, M. W. & Castellanos, M., 2008. *Survey of Text Mining: Clustering, Classification, and Retrieval II.* s.l.:Springer.

Çakmak, D., Erdem, E. & Erdoğan, H., 2009. Computing Weighted Solutions in Answer Set Programming. *LPNMR '09 Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning,* pp. 416 - 422.

Eiter, T., 2008. *Answer Set Programming in a Nutshell,* Freiburg: s.n.

Feldman, R. & Dagan, I., 1995. Knowledge Discovery in Textual Databases (KDT).

Feldman, R. & Sanger, J., 2007. *The Text Mining Handbook: Advanced Approaches to Analyzing Unstructured Data.* Cambridge, England: Cambridge University Press.

Feng, X., Shen, J. & Fan, Y., 2009. REST：An Alternative to RPC for Web Services Architecture. *First International Conference on Future Information Networks,* pp. 7 - 10.

Fischer, S., Petrova, M., Mahonen, P. & Vocking, B., 2007. Distributed Load Balancing Algorithm for Adaptive Channel Allocation for Cognitive Radios. *Cognitive Radio Oriented Wireless Networks and Communications,* pp. 508 - 513.

Gardenfors, 1973. Assignment Problem based on Ordinal Preferences. *Mgmnt Sci,* pp. 331-340.

Gavanelli, M. & Rossi, F., 2010. Constraint Logic Programming. *A 25-Year Perspective on Logic Programming,* pp. 64-86.

Han, J. & Kamber, M., 2000. *Data Mining: Concepts and Techniques.* s.l.:Morgan Kaufmann.

KO, K.-I. & LIN, C.-L., 1995. ON THE COMPLEXITY OF MIN-MAX OPTIMIZATION PROBLEMS AND THEIR APPROXIMATION. *Minimax and Applications, Kluwer Academic Publishers, Boston,* p. 219–239.

Leite, J., Alferes, J. & Mito, B., 2009. Resource Allocation with Answer-Set Programming. *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems,* pp. 649 - 656.

Lifschitz, V., 2008. What Is Answer Set Programming. *Proceedings of the AAAI Conference on Artificial Intelligence,* p. 1594 – 1597.

Lu, H. & Carey, M. J., 1986. Load-Balanced Task Allocation in Locally Distributed Computer Systems. *Proceesings of the 1986 International Conference on Parallel Processing,* pp. 1037 - 1039.

Papadimitriou, C. H., 2003. Computational complexity. *Encyclopedia of Computer Science, John Wiley and Sons Ltd.,* pp. 260-265.

Price, S., Flach, P. A. & Spiegler, S., 2010. *SubSift: a novel application of the vector space model to support the academic research process.* Windsor, UK, s.n., pp. 20-27.

Proll, . L. G., 1972. A Simple Method of Assigning Projects to Students. *Operational Research Quarterly,* pp. 23(2):195-201.

Prud'hommeaux, E. & Seaborne, A., 2008. SPARQL Query Language for RDF. *W3C Recommendation, World Wide Web Consortium.*

Ricca, F. et al., 2011. Team-building with Answer Set Programming in the Gioia-Tauro Seaport. *Theory and Practice of Logic Programming,* p. doi:10.1017/S147106841100007X.

Ryabokon, A. et al., 2012. (Re)Configuration usingWeb Data: A Case Study on the Reviewer Assignment Problem. *Web Reasoning and Rule Systems - 6th International Conference.*

Tovey, C. A., 2002. Tutorial on Computational Complexity. *Interfaces, INFORMS,* p. 30–61.

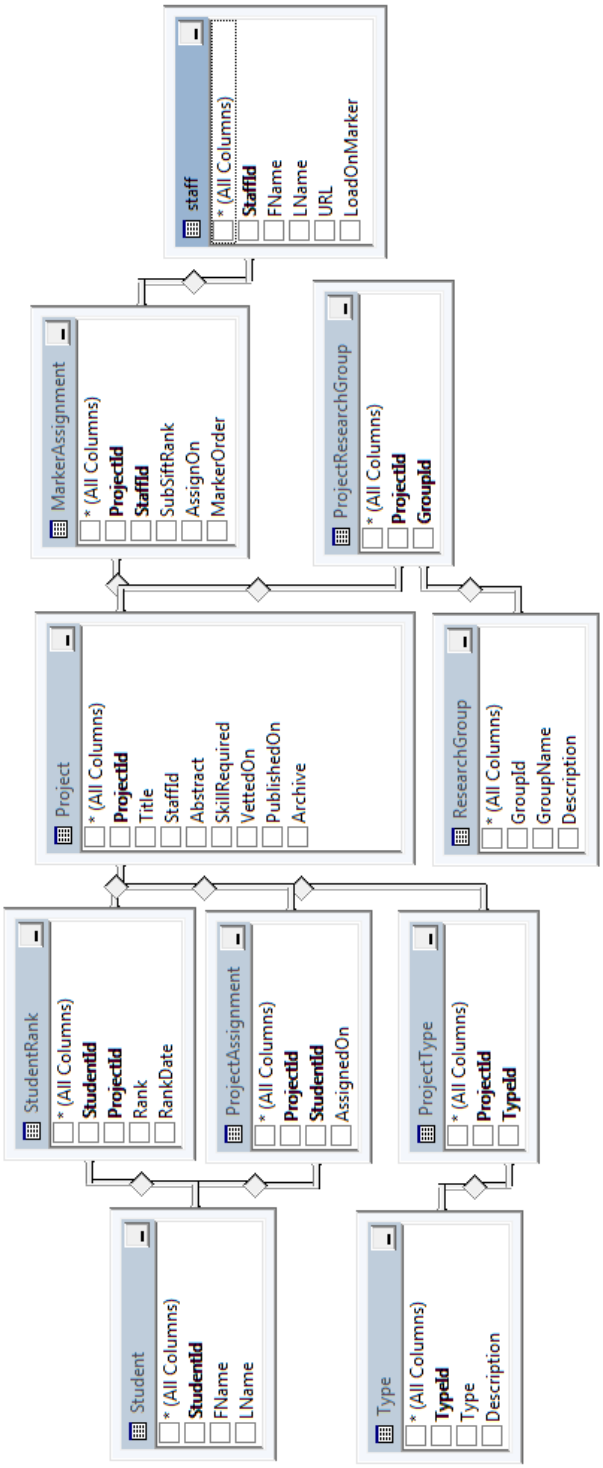
Wilson, L. B., 1977. Assignment Using Choice Lists. *Operational Research Quarterly,* pp. 569-578.

Womersley, R., 2001. The Travelling Salesman Problem and Computational complexity. *Parabola,* pp. Volume 37, Issue 2.

Appendix I – Database structure

Database structure and relations between tables:

(Note that fields in bold are primary keys)



Appendix II – Students-Projects Allocation Program

```
rank(da1653,1,1;; da1653,2,1;; da1653,3,1;; da1653,4,1;; da1653,5,1;; da1653,6,1;; da1653,7,1;;
da1653,8,1;; da1653,9,1;; da1653,10,1;; da1653,11,1;; da1653,12,1;; da1653,13,1;; da1653,14,1;;
da1653,15,1;; da1653,16,1;; da1653,17,1;; da1653,18,1;; da1653,19,1;; da1653,20,1;; da1653,21,1;;
da1653,22,1;; da1653,23,1;; da1653,24,1).

% ranks of the remaining 11 students suppressed

project(1, ccmm).
project(2, ccmm).
project(3, csjjtg).
project(4, csjjtg).
project(5, cspaf).
project(6, cspaf).
project(7, cspaf).
project(8, cspaf).
project(9, ccmm).
project(10, ccmm).
project(11, csjjtg).
project(12, csjjtg).
project(13, cssg).
project(14, cssg).
project(15, cssg).
project(16, cssg).
project(17, cstmdk).
project(18, cstmdk).
project(19, cstmdk).
project(20, cstmdk).
project(21, csxor).
project(22, csxor).
project(23, csxor).
project(24, csxor).

student(aa1641).
student(cb1987).
student(da1653).
student(hl1614).
student(hx1460).
student(mp1384).
student(mw8508).
student(ps1352).
student(rh1286).
student(rs1576).
student(rs1730).
student(sv1865).

staff(ccmm,4).
staff(csjjtg,4).
staff(cspaf,4).
staff(cssg,4).
staff(cstmdk,4).
staff(csxor,4).

1{assign(S,P,V):project(P,V)}1 :- student(S).

:- assign(S1,P,_), assign(S2,P,_), S1 != S2.
:- assign(S,P1,_), assign(S,P2,_), P1 != P2.

% student S has badness R under the assignment
bad(S,R) :- assign(S,P,_), rank(S,P,R).

% some student S2 is worse off than S1
worse(S1) :- bad(S1,R1), bad(S2,R2), R1<R2, S1 != S2.

% don't assign if exceed load
:- assign(_,_,V), staff(V,L), B = #count{assign(_,_,V)}, B > L.

% R is score of worst off student
worst(R) :- bad(S,R), not worse(S).
totalSum(T) :- T = #sum [bad(_,R)=R].

#minimize [totalSum(T)=T].
#minimize [worst(R)=R].

#hide.
#show assign/3.
#show bad/2.
```

Appendix III – Markers-Projects Allocation Program

```
rank(1, cspaf, 1;; 1, cssg, 2;; 1, cstmdk, 3;; 1, ccmm, 4;; 1, csjjtg, 5;; 1, csxor, 6).
rank(12, cstmdk, 1;; 12, cspaf, 2;; 12, ccmm, 3;; 12, csjjtg, 4;; 12, cssg, 5;; 12, csxor, 6).
rank(15, csxor, 1;; 15, cstmdk, 2;; 15, cssg, 3;; 15, cspaf, 4;; 15, ccmm, 5;; 15, csjjtg, 6).
rank(16, cstmdk, 1;; 16, cssg, 2;; 16, csxor, 3;; 16, ccmm, 4;; 16, csjjtg, 5;; 16, cspaf, 6).
rank(19, cstmdk, 1;; 19, ccmm, 2;; 19, csjjtg, 3;; 19, csxor, 4;; 19, cspaf, 5;; 19, cssg, 6).
rank(2, cstmdk, 1;; 2, ccmm, 2;; 2, csjjtg, 3;; 2, csxor, 4;; 2, cspaf, 5;; 2, cssg, 6).
rank(20, cspaf, 1;; 20, cstmdk, 2;; 20, csxor, 3;; 20, cssg, 4;; 20, ccmm, 5;; 20, csjjtg, 6).
rank(23, cspaf, 1;; 23, cstmdk, 2;; 23, csxor, 3;; 23, ccmm, 4;; 23, csjjtg, 5;; 23, cssg, 6).
rank(24, cstmdk, 1;; 24, cspaf, 2;; 24, csxor, 3;; 24, cssg, 4;; 24, ccmm, 5;; 24, csjjtg, 6).
rank(3, cstmdk, 1;; 3, cspaf, 2;; 3, csxor, 3;; 3, cssg, 4;; 3, ccmm, 5;; 3, csjjtg, 6).
rank(7, cstmdk, 1;; 7, ccmm, 2;; 7, csjjtg, 3;; 7, cspaf, 4;; 7, cssg, 5;; 7, csxor, 6).
rank(8, cspaf, 1;; 8, csxor, 2;; 8, cstmdk, 3;; 8, ccmm, 4;; 8, csjjtg, 5;; 8, cssg, 6).

project(1, ccmm).
project(2, ccmm).
project(3, csjjtg).
project(7, cspaf).
project(8, cspaf).
project(12, csjjtg).
project(15, cssg).
project(16, cssg).
project(19, cstmdk).
project(20, cstmdk).
project(23, csxor).
project(24, csxor).

student(aa1641).
student(cb1987).
student(da1653).
student(hl1614).
student(hx1460).
student(mp1384).
student(mw8508).
student(ps1352).
student(rh1286).
student(rs1576).
student(rs1730).
student(sv1865).

staff(ccmm,4).
staff(csjjtg,4).
staff(cspaf,4).
staff(cssg,4).
staff(cstmdk,4).
staff(csxor,4).

1{assign(1,P,M):staff(M, )}1 :- project(P, ).  % assign marker1 to each project
1{assign(2,P,M):staff(M,_)}1 :- project(P,_).  % assign marker2 to each project

:- assign(1,P,M1), assign(2,P,M2), rank(P,M1,R1), rank(P,M2,R2), R1>R2.

:- assign(1,P,M), assign(2,P,M).

:- assign(_,P,S), project(P,S).

:- assign(_,P,M), staff(M,L), B = #count{assign(_,_,M)}, B > L.           % don't assign if exceed
load

bad(N,P,R) :- assign(N,P,M),rank(P,M,R).

worse(N,P1) :- bad(N,P1,R1), bad(N,P2,R2), R1<R2.

worst(N,R) :- bad(N,P,R), not worse(N,P).

totalSum(N,T) :- T = #sum [bad(N,_,R)=R].
#minimize [totalSum(2,T)=T].
#minimize [totalSum(1,T)=T].

#minimize [worst(2,R)=R].
#minimize [worst(1,R)=R].

#hide.
#show assign/3.
#show bad/3.
```

Appendices IV – Source Code

Please note that this code is not the complete one as many parts like styles, formatting, and error handling are suppressed

In this appendix:

SubSift Class

SQL Class

Utilities Class

Coordinator Main Page

Supervisor Main Page

Student Main Page

```
public class SubSift
{    /* 'prepareSubSiftBookmarkText' method prepares markers data (from webpages) for SubSift matching
     * the passed text is created in folders with given name textId e.g. abstracts, search, markers, .. */.
    public static string prepareSubSiftBookmarkText(string textId, string bookmarkItems) {
        bool done;
        deleteTextAll(textId);//delete all folders, items previously created by SubSift
        // Create Bookmark folder (step 1)
        done = SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/bookmarks/" + textId, "POST");
        //Create bookmark Items (step 2)
       done=SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/bookmarks/"+textId+"/items.xml","POST",
                          "items_list",bookmarkItems);
        //Create documents folder (step 3)
        done = SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/documents/" + textId, "POST");
        //import document items from web pages (bookmarks) (step 4)
        done = SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/documents/" + textId + "/import/" +
                          textId, "POST");
        Thread.Sleep(300000); //delay 5 minutes as require for SubSift to fetch all data from webpages
        //Create Profiles (step 5)
        done = SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/profiles/" + textId, "POST");
        return "";     //return empty string means preparing bookmark items is finished successfully
    }
 /* 'prepareSubSiftText' method prepares projects abstracts or students search terms 'text' for SubSift matching
  the passed text is created in folders with given name textId e.g. abstracts, search, markers, ... */
    public static string prepareSubSiftText(string textId, string text) {
        bool done;
        deleteTextAll(textId);//delete all folders, items from previously created by SubSift
        //Create Documents Folder (step 1)
        done = SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/documents/" + textId, "POST");
        //Create document Items (step 2)
        done = SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/documents/" + textId + "/items", "POST",
                          "items_list", text);
        //Create Profiles (step 3)
        done = SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/profiles/" + textId + "/from/" +
                          textId, "POST");
        return "";      //return empty string means preparing text is finished successfully
    }
     /* method match is for matching between projects abstracts and markers or students search terms */
    public static string match(string firstTextId, string secondTextId) {
        bool done;
        try {
            deleteMatches(firstTextId, secondTextId);//delete matches results from previously created by SubSift
            //Matching first text and second text Profiles (step 1)
            done = SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/matches/" + firstTextId + "_" +
                          secondTextId + "/profiles/" + firstTextId + "/with/" + secondTextId, "POST");
            if (!done){return "Error Matching " + firstTextId + " and " + secondTextId + " Profiles ";}
            //Get Matched results (step 2). profiles_id parameter is to select which text to group by
            string content = SubSiftGetMatchedItems("http://subsift.ilrt.bris.ac.uk/dabbas/matches/" +
                firstTextId + "_" + secondTextId + "/items?profiles_id=" + secondTextId, "GET");
            if (content == ""){return "Error getting Matched " + firstTextId + " and " + secondTextId;}
            return content;
        }
        catch (Exception ex1) { return ex1.Message;}
    }
      // method 'SubSiftMethod' is takes two parameters url and method (e.g. POST, GET,...)       *
    public static bool SubSiftMethod(string url, string method) {
        try {
            //initialising a new WebRequest instance for the url then adding new values to the Headers
            WebRequest myReq = WebRequest.Create(url);
            myReq.Headers.Add("Token", "2134712a45e47f6eb2482aa73149e3c31b80d14c");
            myReq.Method = method;
            WebResponse wr = myReq.GetResponse();//getting a response from given url
            return true;  //if no error return true otherwise return false
       } catch (Exception){return false;}
    }
     // 'SubSiftMethod' takes 4 parameters URL, method (e.g.POST,...),
     // parameterName e.g. "items_list", and parameterValue (text)
    public static bool SubSiftMethod(string url, string method, string parameterName, string parameterValue) {
        try {
            //initialising a new WebRequest instance for the url, then adding new values to the Headers
            WebRequest myReq = WebRequest.Create(url);
            myReq.Headers.Add("Token", "2134712a45e47f6eb2482aa73149e3c31b80d14c");
            myReq.Method = method;
            string postString = parameterName + "=" + parameterValue;
            StreamWriter requestWriter = new StreamWriter(myReq.GetRequestStream());
            requestWriter.Write(postString);
            requestWriter.Close();
            WebResponse wr = myReq.GetResponse();//provides a response from given url
            return true;
        } catch (Exception e){Console.WriteLine(e.Message); return false;}
    }
```

1

```
        //method 'SubSiftGetMatchedItems' returns match result
        public static string SubSiftGetMatchedItems(string url, string method) {
            try {
                WebRequest myReq = WebRequest.Create(url);
                myReq.Method = method;
                WebResponse wr = myReq.GetResponse();
                Stream receiveStream = wr.GetResponseStream();
                StreamReader reader = new StreamReader(receiveStream, Encoding.UTF8);
                string content = reader.ReadToEnd();
                return content;
            } catch (Exception){return "error";}}
       //methods for deleting all previousely created folders, items, matched results
        public static bool deleteTextAll(string folderName)
        {
            SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/bookmarks/" + folderName, "DELETE");
            SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/bookmarks/" + folderName + "/items", "DELETE");
            SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/documents/"+folderName, "DELETE");
            SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/documents/"+folderName+"/items", "DELETE");
            SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/profiles/"+folderName, "DELETE");
            SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/profiles/"+folderName+"/items", "DELETE");
            return true;
        }
        private static bool deleteMatches(string firstTextId, string secondTextId)
        {
            SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/matches/" + firstTextId + "_" +
                          secondTextId, "DELETE");
            SubSiftMethod("http://subsift.ilrt.bris.ac.uk/dabbas/matches/" + firstTextId + "_" +
                          secondTextId+"/items", "DELETE");
            return true;
        }}}
/*-------------------------------------------------------------------------------------------------------------------*/
    public class SQL
    {
        // 'conn' represents a connection to a SQL Server DB
        static SqlConnection conn =
         new SqlConnection(ConfigurationManager.ConnectionStrings["ProjectAllocConnectionString"].ConnectionString);
        static SqlDataAdapter da;
        //method 'getAbstracts'returns abstracts from DB for SubSift projects items
        public static string getAbstracts(bool assignedOnly) {
            string abstracts = ""; conn.Open();
            if(assignedOnly)     //only those abstracts that assigned to students
                da = new SqlDataAdapter("SELECT * from Project WHERE ProjectId IN (SELECT ProjectId FROM
ProjectAssignment)", conn);
            else                 // select all projects abstracts from DB
                da = new SqlDataAdapter("SELECT * from Project", conn);
            DataSet ds = new DataSet(); da.Fill(ds, "abstracts");
            foreach (DataRow r in ds.Tables["abstracts"].Rows) {
                string description = r["Title"] + " " + r["Abstract"] + " " + r["SkillRequired"];
                description = description.Replace('"', ' ');
                description = description.Replace(';', ' ');
                description = description.Replace('&', ' ');
                description = description.Replace('\n', ' ');
                description = description.Replace('\r', ' ');
                abstracts += r["ProjectId"] + "," + description + "\n";
            }
            conn.Close(); return abstracts;
        }
        // 'getMarkersList' returns staffs details
        public static string getMarkersList() {
            string markers = ""; conn.Open();
            da = new SqlDataAdapter("SELECT * from staff", conn);
            DataSet ds = new DataSet(); da.Fill(ds, "markers");
            foreach (DataRow r in ds.Tables["markers"].Rows)
                markers += r["StaffId"] + "," + r["FName"] + " " + r["LName"] + "," + r["URL"] + "\n";
            conn.Close(); return markers;
        }
        //'GenerateNewProjectId' return new project id for new project to be published
        public static long GenerateNewProjectId() {
            conn.Open();
            da = new SqlDataAdapter("SELECT MAX(ProjectId) AS MaxId FROM Project", conn);
            DataSet ds = new DataSet(); da.Fill(ds, "ProjectId");
            DataRow dr = ds.Tables["ProjectId"].Rows[0]; conn.Close();
            if (dr["MaxId"] == DBNull.Value) return 1; //first time will be 1 otherwise the max number + 1
            else return Convert.ToInt64(dr["MaxId"]) + 1;
        }
        //'createPredicates' method read from DB required data and convert them to predicate shape
        public static string createPredicates(bool assignedOnly) {
            conn.Open();
            string predicates = "\n";
```

2

```
            if(assignedOnly)     // select only those projects which assign to students
                da = new SqlDataAdapter("SELECT ProjectId,StaffId FROM Project WHERE ProjectId IN " +
                                        "(SELECT ProjectId FROM ProjectAssignment)", conn);
            Else da = new SqlDataAdapter("SELECT ProjectId,StaffId FROM Project", conn);   // select all projects
            DataSet ds = new DataSet(); da.Fill(ds, "projects");
            da = new SqlDataAdapter("SELECT StudentId FROM Student", conn);  da.Fill(ds, "students");
            foreach (DataRow r in ds.Tables["projects"].Rows)// loop on selected projects
                predicates += "project(" + r["ProjectId"] + ", " + r["StaffId"] + ").\n";
            predicates += "\n";  //new line
            foreach (DataRow r in ds.Tables["students"].Rows)// loop on selected students
                predicates += "student(" + r["StudentId"] + ").\n";
            predicates += "\n"; da = new SqlDataAdapter("SELECT StaffId FROM staff", conn);
            da.Fill(ds, "staff");
            foreach (DataRow r in ds.Tables["staff"].Rows)//loop on selected staffs
                predicates += "staff(" + r["StaffId"] + ",4).\n"; //staff load as '4' but it can be changed
            predicates += "\n"; conn.Close(); return predicates;
        }
        // method for deleting records in DB (from the passed table name)
        public static bool deleteRecords(string table) {
            conn.Open();
            SqlCommand cmd = new SqlCommand(); cmd.CommandType = CommandType.Text;
            cmd.CommandText = "DELETE FROM " + table; cmd.Connection = conn;
            try { cmd.ExecuteNonQuery();  conn.Close(); return true;
            } catch (Exception) { conn.Close(); return false; }
        }
        //method for updating students rank after submission of their ranked project list
        public static void UpdateStudentRank(string StudentID, string ProjectID, int Rank) {
            conn.Open();
            SqlCommand cmd = new SqlCommand(); cmd.CommandType = CommandType.Text;  cmd.Connection = conn;
            cmd.Parameters.AddWithValue("@StudentId", StudentID);
            cmd.Parameters.AddWithValue("@ProjectId", ProjectID);
            cmd.Parameters.AddWithValue("@Rank", Rank);
            cmd.Parameters.AddWithValue("@RankDate", DateTime.Now);
            cmd.CommandText = "UPDATE StudentRank SET " +
                "Rank = @Rank,RankDate = @RankDate  " +
                "WHERE ProjectId=@ProjectId AND StudentId = @StudentId";
            cmd.ExecuteNonQuery(); conn.Close();
        }}}
/*-------------------------------------------------------------------------------------------------------------------------*/
    public class Utilities
    {
        //method for reading files taking file path as the parameter and returns file content
        public static string ReadFile(string filepath) {
            try{ StreamReader sr = new StreamReader(HttpContext.Current.Server.MapPath(filepath));
                string fileContent = "";
                while (!sr.EndOfStream) fileContent += sr.ReadLine() + "\n";
                return fileContent;
            } catch (Exception){return "";}
        }
        //method for writing to files and returns path of file that written in
        public static string writeToFile(string fileContent, string fileName) {
            string path = HttpContext.Current.Server.MapPath(fileName);
            using (StreamWriter sw = File.CreateText(path))// Create a file to be written in
            { sw.Write(fileContent); } return path;
        }
    //method for appending data to an existing file
      public static void appendFile(string pathTo, string pathFrom, string predicateName) {
            // Open the file to read from
            using (StreamReader sr = File.OpenText(pathFrom))
            {   string s = "";
                // Create a file to write to if not exist
                using (StreamWriter sw = File.AppendText(pathTo))
                {   if (predicateName == "code") { //if ASP code read and write to the file with no change
                        sw.WriteLine("\n");
                        while ((s = sr.ReadLine()) != null) sw.WriteLine(s);
                    } if (predicateName == "staff") {
                        sw.WriteLine("\n");
                        sw.WriteLine("%(staff(username,first name,last name, accepted load)");
                        while ((s = sr.ReadLine()) != null) sw.WriteLine(predicateName + "(" + s + "," + 8 + ").");
                    } sw.WriteLine("\n");
                    while ((s = sr.ReadLine()) != null) sw.WriteLine(predicateName + "(" + s + ").");
      }   }   }
       //appending coming string to the target file
      public static void appendFile(string pathTo, string content) {
          using (StreamWriter sw = File.AppendText(pathTo))
          { sw.Write(content);}
      }
      //reference: http://forums.asp.net/t/1705264.aspx/1
      public static DataTable ReadToEnd(string filePath) {
          DataTable dtDataSource = new DataTable();
```



```
          //Read all lines from selected file and assign to string array variable.
          string[] fileContent = File.ReadAllLines(filePath);
          //Checks fileContent count > 0 then we have some lines in the file. If = 0 then file is empty
          if (fileContent.Count() > 0) {
              //In CSV file, 1st line contains column names. When you read CSV file, each delimited by ','.
              //fileContent[0] contains 1st line and splitted by ','. columns string array contains list of columns.
              string[] columns = fileContent[0].Split(',');
              for (int i = 0; i < columns.Count(); i++) {dtDataSource.Columns.Add(columns[i]);}
              //Same logic for row data.
              for (int i = 1; i < fileContent.Count(); i++) { string[] rowData = fileContent[i].Split(',');
                  dtDataSource.Rows.Add(rowData);
              }   } return dtDataSource;
      }
      //method for running ASP command and result of ASP will be saved in file 'outputFile'
      public static bool runCommand(string inputFile, string outputFile, string workingDir) {
          //run ASP file from commandline
          try {   //string predicatePath = Server.MapPath("~");
              System.Diagnostics.Process si = new System.Diagnostics.Process();
              si.StartInfo.WorkingDirectory = workingDir;
              si.StartInfo.UseShellExecute = false;
              si.StartInfo.FileName = "cmd.exe";
              si.StartInfo.Arguments = "/c gringo " + inputFile + " | clasp > " + outputFile;
              si.StartInfo.CreateNoWindow = true;
              si.StartInfo.RedirectStandardInput = true;
              si.StartInfo.RedirectStandardOutput = true;
              si.StartInfo.RedirectStandardError = true;
              si.Start();
              string output = si.StandardOutput.ReadToEnd();
              si.Close(); return true;
          } catch (Exception){ return false;}
      } }
/*-------------------------------------------------------------------------------------------------------------------------*/
    public partial class Projects : System.Web.UI.Page
    {
        SqlConnection conn; SqlDataAdapter da; DataTable dt;
        protected void Page_Load(object sender, EventArgs e) {
 conn = new SqlConnection(ConfigurationManager.ConnectionStrings["ProjectAllocConnectionString"].ConnectionString);
       }/*-----------------------------------------------------------------------------------------*
         *  'btnPrepareMarkers' is the button for Preparing marekers
         * ----------------------------------------------------------------------------------------*/
        protected void btnPrepareMarkers_Click(object sender, EventArgs e)  {
            string markersList = SQL.getMarkersList();  //read Markers data from DB (calling SQL class)
            // "markers" is the name that SubSift will use for naming text folders
            // markersList is information about markers 'Staff id, Staff name, URL '
            string message = SubSift.prepareSubSiftBookmarkText("markers", markersList);
            lblMessage.Text = message; //Display return message on label 'lblMessage'
        }/*-----------------------------------------------------------------------------------------*
         *  when button 'Prepare Projects' is clicked the following will be performed
         *----------------------------------------------------------------------------------------*/
        protected void btnPrepareProjects_Click(object sender, EventArgs e) {
            string projectsText = SQL.getAbstracts(true);//read projects from DB (true means only assigned projects)
            // "abstracts" is the name that SubSift will use for naming text folders
            // projectsText is the text content
            string message = SubSift.prepareSubSiftText("abstracts", projectsText);
        }/*-----------------------------------------------------------------------------------------*
         * when button Match is Clicked
         *----------------------------------------------------------------------------------------*/
        protected void btnMatch_Click(object sender, EventArgs e) {
            //calling method 'match' in SubSift class, to match between 'markers' and 'abstracts'
            string SubSiftResult = SubSift.match("markers", "abstracts");
            SubSiftMatchResult(SubSiftResult);  }             //calling method 'SubSiftMatchResult'
        /*-----------------------------------------------------------------------------------------*
         *  method SubSiftMatchResult reads from SubSifit result in xml format
         *----------------------------------------------------------------------------------------*/
        private void SubSiftMatchResult(string SubSiftResult) {
            //string 'result' will be used only for displaying ranked markers for each project
            string result = "Project Id,Project Title,Marker 1, Marker 2, Marker 3, Marker 4, Marker 5, Marker 6\n";
            string rankingMarker = "";  // variable to hold ranked markers for each project
            XmlDocument matchedResult = new XmlDocument(); matchedResult.Load(new StringReader(SubSiftResult));
            XmlNodeList projectNodes = matchedResult.SelectNodes("result/match");//find key words result and match
            foreach (XmlNode project in projectNodes) { //loop on projectNodes
                rankingMarker += "\nrank(";     // the word 'rank' used in predicate file for running ASP
                XmlNodeList markerNodes = project.SelectNodes("item");  //find 'item' in the current node
                string projecId=project.SelectSingleNode("id").InnerText;//read project id & store it in 'projecId'
                conn.Open();//open the connection to DB and read title from 'Project' table
                da = new SqlDataAdapter("SELECT Title FROM Project WHERE ProjectId ="+projecId,conn);
                DataSet ds = new DataSet(); da.Fill(ds,"title");
                result += projecId + ", " + ds.Tables["title"].Rows[0]["Title"] + ",";
                conn.Close();       // close the connection to DB
                int i = 0;      //variable 'i' is for rank number
```

```
            foreach (XmlNode marker in markerNodes) { i++;
                if (i <= markerNodes.Count)//if still not reach end of the node
                //append to the string result the'projec Id' ,"description" name of markers and their userid
                   result += marker.SelectSingleNode("description").InnerText + " (" +
                             marker.SelectSingleNode("id").InnerText + ")";
      //append to the string rankingMarker 'projec Id' ,',' and marker id. 'i' is marker rankes for this project
                    rankingMarker += projecId + ", " + marker.SelectSingleNode("id").InnerText + ", " + i;
                    if (i < markerNodes.Count){rankingMarker += ";; "; result += ",";} }
            //append to the strings rankingMarker close bracket and comments, and result new line
            rankingMarker += ")." + "  % ranked markers for project " + projecId;  result += "\n"; }
        Utilities.writeToFile(result, "~/Result.csv");// writing the result in file
        //for future a page can be devoloped for coordinater to enter name and path of files
        string path = Utilities.writeToFile(rankingMarker, "~/Predicate.pl"); // predicate file for running ASP
        //create project, student, and staff predicates from database tables
        string predicates = SQL.createPredicates(true);
        Utilities.appendFile(path, predicates);//appending all above predicates to the file 'Predicate.pl'
//reading and appending all following (panel,cohort, ...) files to the 'Predicate.pl' file
        string path3 = Server.MapPath("~/App_Data/panel.csv");
        Utilities.appendFile(path, path3, "panel");
        string path4 = Server.MapPath("~/App_Data/cohort.csv");
        Utilities.appendFile(path, path4, "cohort");
        string path5 = Server.MapPath("~/App_Data/tutor.csv");
        Utilities.appendFile(path, path5, "tutor");
        string path9 = Server.MapPath("~/App_Data/markerAssignCode.txt");
        Utilities.appendFile(path, path9, "code");
        lblMessage.Text = "";//clear the message label
    }
    /*-------------------------------------------------------------------------------------------*
     * when button 'Assign Markers' is clicked
     *-------------------------------------------------------------------------------------------*/
    protected void btnASPAssignment_Click(object sender, EventArgs e) {
        string workingDirectory = Server.MapPath("~");
        // Calling method' runCommand' in Utilities class to run ASP command
        bool done = Utilities.runCommand("Predicate.pl", "ASPMarkersRank.txt", workingDirectory);
        if (!done)lblMessage.Text = "Error Assigning Markers to Projects...";
        else{StoreMarkerResultsToDB(); gvStudentASPResult.DataBind();}
    }
    protected void btnSubSiftResult_Click(object sender, EventArgs e) {Response.Redirect("SubSiftResult.aspx");}
    protected void btnASPResult_Click(object sender, EventArgs e) { gvASPResult.Visible = true; }
    protected void btnAssignProjectToStudents_Click(object sender, EventArgs e) {
        conn.Open(); string StudentsRankes = "";
        da = new SqlDataAdapter("SELECT StudentId FROM Student ", conn); //select all students
        DataSet ds = new DataSet(); da.Fill(ds, "Student");
     da=new SqlDataAdapter("SELECT ProjectId FROM Project ", conn);da.Fill(ds, "Project"); //select all projects
        foreach (DataRow rs in ds.Tables["Student"].Rows) {
            string StudentId = rs["StudentId"] + "";
            StudentsRankes += "rank(" + StudentId+","; int i = 0;
            foreach (DataRow rp in ds.Tables["Project"].Rows) { i++;
                string ProjectId = rp["ProjectId"] + "";
                StudentsRankes += ProjectId+",";
                da = new SqlDataAdapter("SELECT StudentId, ProjectId, Rank FROM StudentRank "+
                    "WHERE StudentId ='" + StudentId + "' AND ProjectId=" + ProjectId, conn);
                DataSet dsRank = new DataSet(); da.Fill(dsRank, "StudentRank");
                if (dsRank.Tables["StudentRank"].Rows.Count != 0) {
                    DataRow rr = dsRank.Tables["StudentRank"].Rows[0];
                    if (i != ds.Tables["Project"].Rows.Count) StudentsRankes += rr["Rank"] + ";; ";
                } else {
                    //if this project is not ranked by the student, the rank is defaulted to max rank + 1
                   if (i != ds.Tables["Project"].Rows.Count) StudentsRankes += getStudentMaxRank(StudentId) + ";; ";
                    } if (i != ds.Tables["Project"].Rows.Count) StudentsRankes += StudentId + ",";
            } StudentsRankes += getStudentMaxRank(StudentId);  StudentsRankes += ").\n";
        } conn.Close();
        string path = Utilities.writeToFile(StudentsRankes, "~/StudentRankPredicate.pl");
        //create project, student, and staff predicates from database tables
        string predicates = SQL.createPredicates(false); Utilities.appendFile(path, predicates);
        string path4 = Server.MapPath("~/App_Data/studentAssignCode.txt");
        Utilities.appendFile(path, path4, "code"); string workingDirectory = Server.MapPath("~");
        bool done = Utilities.runCommand("StudentRankPredicate.pl", "ASPStudentsAssign.txt", workingDirectory);
        if (!done) lblMessage.Text = "Error Assigning Projects to Students...";
        else { lblMessage.Text = "Projects assigned to students successfully, please click Show Results...";
            StoreResultsToDB();           //inserting the result of assigning markers to projects into DB
            gvStudentASPResult.DataBind();        //updata the grid view (table which displayed for user)
    }   }
    /*-------------------------------------------------------------------------------------------*
     * method for inserting the result of projects assignment to students into the DB
     * ------------------------------------------------------------------------------------------*/
    private void StoreResultsToDB() {
        // call method deleteRecords in SQL class paramete "ProjectAssignment" is the table name
        SQL.deleteRecords("ProjectAssignment");
        //reading the result of ASP assignments and extracting the requred information to be inserted into DB
```



```
            string aspOutput = Utilities.ReadFile("ASPStudentsAssign.txt");
            int start = aspOutput.LastIndexOf("Answer")+10;
            int end0 = aspOutput.LastIndexOf("assign("); int end = aspOutput.IndexOf(" ",end0);
            string assign = aspOutput.Substring(start, end - start);
            assign = assign.Replace("assign(", ""); assign = assign.Replace(")", "");
            string[] a = assign.Split(' '); //the required data is stored in array string 'a'
            conn.Open();//open connection and loop on 'a'
            for (int i = 0; i < a.Length; i++) {
                int Coma1 = a[i].IndexOf(",");  int Coma2 = a[i].LastIndexOf(",");
                //store student id in 'StudentId' and project id in 'ProjectId' then insert into DB
                string StudentId=a[i].Substring(0, Coma1); string ProjectId=a[i].Substring(Coma1+1, Coma2-Coma1-1);
                SqlCommand cmd = new SqlCommand(); cmd.CommandType = CommandType.Text;
                cmd.Parameters.AddWithValue("@ProjectId", ProjectId);
                cmd.Parameters.AddWithValue("@StudentId", StudentId);
                cmd.Parameters.AddWithValue("@AssignedOn", DateTime.Now);
                cmd.CommandText = "INSERT INTO ProjectAssignment (ProjectId, StudentId, AssignedOn) " +
                                  "VALUES (@ProjectId, @StudentId, @AssignedOn)";
                cmd.Connection = conn;
                try {cmd.ExecuteNonQuery();} catch (Exception) {}
            } conn.Close();
        }
        // method to store the result of projects assignment to markers in the DB
        private void StoreMarkerResultsToDB() {
            //reading the result of ASP assignments and extracting the requred information to be inserted into DB
            string aspOutput = Utilities.ReadFile("ASPMarkersRank.txt");
            int start = aspOutput.LastIndexOf("Answer") + 10;
            int end0 = aspOutput.LastIndexOf("assign("); int end = aspOutput.IndexOf(" ", end0);
            string assign = aspOutput.Substring(start, end - start);
            assign = assign.Replace("assign(", ""); assign = assign.Replace(")", "");
            string[] a = assign.Split(' ');          //array string 'a' for storing the extracted data
            SQL.deleteRecords("MarkerAssignment");  //delete old data from DB
        //open connection and loop on 'a' for inserting the result of ASP assignment of markers to projects into DB
            conn.Open();
            for (int i = 0; i < a.Length; i++) {
                int Coma1 = a[i].IndexOf(",");  int Coma2 = a[i].LastIndexOf(",");
                string order = a[i].Substring(0, Coma1);  //order indicate that this marker is marker 1 or 2
                string ProjectId = a[i].Substring(Coma1 + 1, Coma2 - Coma1 - 1);
                string StaffId = a[i].Substring(Coma2 + 1, a[i].Length - Coma2 - 1);
                SqlCommand cmd = new SqlCommand(); cmd.CommandType = CommandType.Text;
                cmd.Parameters.AddWithValue("@ProjectId", ProjectId);
                cmd.Parameters.AddWithValue("@StaffId", StaffId);
                cmd.Parameters.AddWithValue("@MarkerOrder", Convert.ToInt16(order));
                cmd.Parameters.AddWithValue("@AssignOn", DateTime.Now);
                cmd.CommandText = "INSERT INTO MarkerAssignment (ProjectId, StaffId, AssignOn, MarkerOrder) " +
                                  "VALUES (@ProjectId, @StaffId, @AssignOn, @MarkerOrder)";
                cmd.Connection = conn;
                try { cmd.ExecuteNonQuery(); } catch (Exception) {}
            } conn.Close();
        }
        // method to get student max rank and return 1 if student not ranked any project otherwise max+1
        private int getStudentMaxRank(string StudentId) {
            da = new SqlDataAdapter("SELECT MAX(Rank) AS MaxRank FROM StudentRank WHERE StudentId ='" +
                                    StudentId+"'", conn);
            DataSet ds = new DataSet(); da.Fill(ds, "MaxRank");
            DataRow dr = ds.Tables["MaxRank"].Rows[0];
            //first time will be 1 otherwise the max number + 1
            if (dr["MaxRank"] == DBNull.Value) return 1; else return Convert.ToInt32(dr["MaxRank"]) + 1;
        }
        // button to display the result of ASP result for assigning students to projects
        protected void btnStudentASPResult_Click(object sender, EventArgs e) {gvStudentASPResult.Visible = true;}
        protected void gvASPResult_RowDataBound(object sender, GridViewRowEventArgs e) {
            if (e.Row.RowType == DataControlRowType.DataRow) { //perfomance for rows without header and footer
           ((SqlDataSource)e.Row.FindControl("SqlDataSourceMarkersAssign")).SelectParameters[0].DefaultValue =
                ((Label)e.Row.FindControl("lblProjectId")).Text;
                ((GridView)e.Row.FindControl("GridView1")).DataBind();
        }   }
        protected void gvStudentASPResult_RowDataBound(object sender, GridViewRowEventArgs e) {
            if (e.Row.RowType == DataControlRowType.DataRow) {
                string StudentId = ((Label)e.Row.FindControl("lblStudentId")).Text;
                string ProjectId = ((Label)e.Row.FindControl("lblProjectId")).Text;
                da = new SqlDataAdapter("SELECT Rank FROM StudentRank WHERE StudentId='"+ StudentId
                                        + "' AND ProjectId=" + ProjectId, conn);
                DataSet ds = new DataSet(); da.Fill(ds, "rank");
                if (ds.Tables["rank"].Rows.Count == 0) ((Label)e.Row.FindControl("lblRank")).Text = "1";
                else { DataRow r=ds.Tables["rank"].Rows[0]; ((Label)e.Row.FindControl("lblRank")).Text=r["Rank"]+"";
}         }     }
/*-------------------------------------------------------------------------------------------------------------------*/
    public partial class SupervisorMainPage : System.Web.UI.Page
    {
        SqlConnection conn; SqlDataAdapter da, daType;
```

```
        //  On page load populate all dropdown lists for availabel project types from DB
        protected void Page_Load(object sender, EventArgs e) {
            conn = new
SqlConnection(ConfigurationManager.ConnectionStrings["ProjectAllocConnectionString"].ConnectionString);
            if (!IsPostBack) {
                conn.Open();
                daType = new SqlDataAdapter("SELECT * FROM Type", conn);//read from DB all project types
                DataSet ds = new DataSet(); daType.Fill(ds, "type");
                //add empty row in all dropdown lists, so by default not selected any type
                Utilities.addEmptyItem(ddlProjectType1);
                Utilities.addEmptyItem(ddlProjectType2);
                Utilities.addEmptyItem(ddlProjectType3);
                int i = 1;
                foreach(DataRow r in ds.Tables["type"].Rows) {
                    //add available types which obtained from DB to dropdown lists
                   ddlProjectType1.Items.Insert(i,new ListItem(r["Type"]+": "+r["description"],r["TypeId"]+""));
                   ddlProjectType2.Items.Insert(i, new ListItem(r["Type"]+": "+r["description"], r["TypeId"]+ ""));
                   ddlProjectType3.Items.Insert(i, new ListItem(r["Type"]+": "+r["description"], r["TypeId"]+ ""));
                   i++;
                } conn.Close();
        }   }
         //  when button 'Submit' is clicked insert the proposed project into DB by reading data from page
        protected void btnSubmitProject_Click(object sender, EventArgs e) {
            conn.Open();
            long ProjectId = SQL.GenerateNewProjectId();  //method automatically generate an id for new project
            SqlCommand cmd = new SqlCommand();
            cmd.CommandType = CommandType.Text;
            cmd.Parameters.AddWithValue("@ProjectId",ProjectId);
            cmd.Parameters.AddWithValue("@Title",txtProjectTitle.Text);
            cmd.Parameters.AddWithValue("@Abstract",txtProjectAbstract.Text);
            cmd.Parameters.AddWithValue("@SkillRequired", txtSkillReqDescription.Text);
            cmd.Parameters.AddWithValue("@StaffId",ddlSupervisor.SelectedValue);
            cmd.Parameters.AddWithValue("@VettedOn",DateTime.Now);
            cmd.CommandText = "INSERT INTO Project (ProjectId, Title, Abstract, " +
                              "SkillRequired, StaffId, VettedOn) VALUES (@ProjectId, @Title,"+
                              "@Abstract, @SkillRequired, @StaffId, @VettedOn)";
            cmd.Connection = conn; cmd.ExecuteNonQuery();
            //call method for inserting project type for any project proposed by supervisor
            addProjectType(ProjectId, ddlProjectType1.SelectedValue);
            addProjectType(ProjectId, ddlProjectType2.SelectedValue);
            addProjectType(ProjectId, ddlProjectType3.SelectedValue);
            //call method for inserting project group for any project proposed by supervisor
            addProjectGroup(ProjectId, ddlResearchGroup1.SelectedValue);
            addProjectGroup(ProjectId, ddlResearchGroup2.SelectedValue);
            addProjectGroup(ProjectId, ddlResearchGroup3.SelectedValue);
            conn.Close();  GridView1.DataBind(); reset();
        }
        //method reset for cleaning the page and make the page ready for supervisor to propose a new project
        private void reset() { // clears all fields content - suppressed }
        //method for adding new proposed project type to DB
        private void addProjectType(long ProjectId, string TypeId) {
            if (!TypeId.Equals("")) {
                SqlCommand cmd = new SqlCommand();
                cmd.CommandType = CommandType.Text;
                cmd.Parameters.AddWithValue("@ProjectId", ProjectId);
                cmd.Parameters.AddWithValue("@TypeId", Convert.ToInt32(TypeId));
                cmd.CommandText = "INSERT INTO ProjectType (ProjectId, TypeId) " +
                                  "VALUES (@ProjectId, @TypeId)";
                cmd.Connection = conn;
                try {cmd.ExecuteNonQuery();}
                catch(Exception){}
        }   }
        //method for adding research group of new proposed project to DB
        private void addProjectGroup(long ProjectId, string GroupId) {
            if (!GroupId.Equals("")) {
                SqlCommand cmd = new SqlCommand();
                cmd.CommandType = CommandType.Text;
                cmd.Parameters.AddWithValue("@ProjectId", ProjectId);
                cmd.Parameters.AddWithValue("@GroupId", Convert.ToInt32(GroupId));
                cmd.CommandText = "INSERT INTO ProjectResearchGroup (ProjectId, GroupId) " +
                                  "VALUES (@ProjectId, @GroupId)";
                cmd.Connection = conn; cmd.ExecuteNonQuery();
        }   }
        /*-------------------------------------------------------------------------------------------*
         *  when supervisors select their user name from dropdown list the system displays supervisor name by
         * reading from DB
         * ------------------------------------------------------------------------------------------*/
        protected void ddlSupervisor_SelectedIndexChanged(object sender, EventArgs e) {
            conn.Open();
            da = new SqlDataAdapter("SELECT FName, LName FROM Staff WHERE StaffId = '"
```



```
                                + ddlSupervisor.SelectedValue + "'", conn);
            DataSet ds = new DataSet();
            da.Fill(ds,"Supervisor");
            //if this supervisor is found in DB display his/her name on the label 'lblSupervisorName'
            if(ds.Tables["Supervisor"].Rows.Count !=0) {
                DataRow r = ds.Tables["Supervisor"].Rows[0];
                lblSupervisorName.Text = r["FName"] + " " + r["LName"];
            } conn.Close(); reset();
        }
        /*-------------------------------------------------------------------------------------------*
         * GridView1 is the place on right side of the page where displayed all projects that are published for
         * the current staff. The publish projects are obtained from DB. By selecting any project from this list
         * the project will be displayed on the page so the staff can update or delete them
         * ------------------------------------------------------------------------------------------*/
        protected void GridView1_SelectedIndexChanged(object sender, EventArgs e)
        {
            conn.Open();
            da = new SqlDataAdapter("SELECT * FROM Project WHERE ProjectId='"+GridView1.SelectedValue+"'", conn);
            DataSet ds = new DataSet();
            da.Fill(ds, "Project");
            DataRow dr = ds.Tables["Project"].Rows[0];
            txtProjectTitle.Text = dr["Title"]+"";
            txtProjectAbstract.Text = dr["Abstract"]+"";
            txtSkillReqDescription.Text = dr["SkillRrequired"]+"";
      da = new SqlDataAdapter("SELECT * FROM ProjectType WHERE ProjectId='" + GridView1.SelectedValue + "'", conn);
            da.Fill(ds, "ProjectType");
            int j = 1;
            foreach(DataRow dr2 in ds.Tables["ProjectType"].Rows) {
                switch(j) {
                    case 1: ddlProjectType1.SelectedValue = dr2["TypeId"]+""; break;
                    case 2: ddlProjectType2.SelectedValue = dr2["TypeId"]+""; break;
                    case 3: ddlProjectType3.SelectedValue = dr2["TypeId"]+""; break;
                } j++;
            }
da = new SqlDataAdapter("SELECT * FROM ProjectResearchGroup WHERE ProjectId='"+GridView1.SelectedValue+"'", conn);
            da.Fill(ds, "ProjectGroup");
            int k = 1;
            foreach (DataRow dr3 in ds.Tables["ProjectGroup"].Rows) {
                switch (k) {
                    case 1: ddlResearchGroup1.SelectedValue = dr3["GroupId"] + ""; break;
                    case 2: ddlResearchGroup2.SelectedValue = dr3["GroupId"] + ""; break;
                    case 3: ddlResearchGroup3.SelectedValue = dr3["GroupId"] + ""; break;
                } k++;
            } btnDeleteProject.Visible = true; btnUpdateProject.Visible = true; btnSubmitProject.Visible = false;
        }
        // when button 'DELETE PROJECT' pressed that project will be flaged as 'Archive'
        protected void btnDeleteProject_Click(object sender, EventArgs e) {
            conn.Open();
            SqlCommand cmd = new SqlCommand(); cmd.CommandType = CommandType.Text;
            cmd.CommandText = "UPDATE Project SET Archive = 1 WHERE ProjectId='" + GridView1.SelectedValue + "'";
            cmd.Connection = conn; cmd.ExecuteNonQuery();
            GridView1.DataBind(); reset();
        }
        // when button 'UPDATE PROJECT' pressed that project record will be modified in DB and
        protected void btnUpdateProject1_Click(object sender, EventArgs e) {
            conn.Open();
            SqlCommand cmd = new SqlCommand(); cmd.CommandType = CommandType.Text; cmd.Connection = conn;
            long ProjectId = Convert.ToInt64(GridView1.SelectedValue);
            cmd.Parameters.AddWithValue("@ProjectId", GridView1.SelectedValue);
            cmd.Parameters.AddWithValue("@Title", txtProjectTitle.Text);
            cmd.Parameters.AddWithValue("@Abstract", txtProjectAbstract.Text);
            cmd.Parameters.AddWithValue("@SkillRequired", txtSkillReqDescription.Text);
            cmd.CommandText = "UPDATE Project SET Title = @Title, Abstract = @Abstract,"+
                "SkillRequired = @SkillRequired WHERE ProjectId=@ProjectId";
            cmd.ExecuteNonQuery();
            cmd.CommandText = "DELETE FROM ProjectType WHERE ProjectId=@ProjectId"; cmd.ExecuteNonQuery();
            addProjectType(ProjectId, ddlProjectType1.SelectedValue);
            addProjectType(ProjectId, ddlProjectType2.SelectedValue);
            addProjectType(ProjectId, ddlProjectType3.SelectedValue);
            cmd.CommandText = "DELETE FROM ProjectResearchGroup WHERE ProjectId=@ProjectId"; cmd.ExecuteNonQuery();
            addProjectGroup(ProjectId, ddlResearchGroup1.SelectedValue);
            addProjectGroup(ProjectId, ddlResearchGroup2.SelectedValue);
            addProjectGroup(ProjectId, ddlResearchGroup3.SelectedValue);
            conn.Close(); GridView1.DataBind(); reset();
        }
        private void updateProjectType(long ProjectId, string TypeId) {
            if (!TypeId.Equals("")) {
                SqlCommand cmd = new SqlCommand(); cmd.CommandType = CommandType.Text;
                cmd.Parameters.AddWithValue("@TypeId", Convert.ToInt32(TypeId));
                cmd.CommandText = "UPDATE ProjectType SET TypeId = @TypeId  WHERE ProjectId='" + ProjectId + "'";
```

```
            cmd.Connection = conn; cmd.ExecuteNonQuery();
    }   }   }
/*-------------------------------------------------------------------------------------------------------------------*/
    public partial class StudentMainPage : System.Web.UI.Page
    {
        SqlConnection conn; SqlDataAdapter daType, da; DataTable dt;
        protected void Page_Load(object sender, EventArgs e) {
            conn =
      new SqlConnection(ConfigurationManager.ConnectionStrings["ProjectAllocConnectionString"].ConnectionString);
            if (!IsPostBack) {
           /* RangeValidator is used for validating the number of projects for display entered by user
                RangeValidator1.MaximumValue = (SQL.GenerateNewProjectId() - 1)+"";
                RangeValidator1.ErrorMessage += RangeValidator1.MaximumValue;
                txtBestFit.Text = RangeValidator1.MaximumValue;
         // populate all dropdown lists for availabel project types from DB
                conn.Open();
                daType = new SqlDataAdapter("SELECT * FROM Type", conn);
                DataSet ds = new DataSet(); daType.Fill(ds, "type");
                int i = 1;
                foreach (DataRow r in ds.Tables["type"].Rows) {
                    ddlProjectType1.Items.Insert(i, new ListItem(r["Type"]+": "+r["description"], r["TypeId"]+""));
                    ddlProjectType2.Items.Insert(i, new ListItem(r["Type"]+": "+r["description"], r["TypeId"]+ ""));
                    ddlProjectType3.Items.Insert(i, new ListItem(r["Type"]+": "+r["description"], r["TypeId"]+ ""));
                    i++; }  conn.Close();
        }   }
        //button for displaying matched students search projects
        protected void btnProjectsFilter_Click(object sender, EventArgs e) {
            string SubSiftResult="";    //storing student entry in a variable
            string studentSearch = "studentSearch, " + ddlProjectType1.SelectedItem.Text + " " +
                ddlProjectType2.SelectedItem.Text + " " + ddlProjectType3.SelectedItem.Text + " " +
                ddlResearchGroup1.SelectedItem.Text + " " + ddlResearchGroup2.SelectedItem.Text + " " +
                ddlResearchGroup3.SelectedItem.Text + " " + txtSearch.Text;
        // prepare the search query (first and second parties) for SubSift matching by calling the method
        * 'prepareSubSiftText'
        * in SubSift class. "search" and "projects" is the name that SubSift will use for naming text folders
            string search = "search";
            string message = SubSift.prepareSubSiftText(search , studentSearch); lblMessage.Text = message;
            if (message.Equals("")) { //if there is no error message in preparing student terms
                string projectsText = SQL.getAbstracts(false);  string projects = "projects";
                message = SubSift.prepareSubSiftText(projects, projectsText); lblMessage.Text = message;
       /* if there is no error message in preparing abstracts then match between two parties projects abstracts and
        * student search terms then store the result in variable 'SubSiftResult'*/
                if (message.Equals(""))  {
                    SubSiftResult = SubSift.match(projects, search);
                    showResults(SubSiftResult); // method to displays the result of SubSift
                    ProjectAccordion.Visible = true;
     //if student not already submited his/her favourite ranked list the submit button set visible to this student
                    if (!studentSubmitted(ddlStudent.SelectedValue.ToString())) btnAddToFavorite.Visible = true;
        }   }   }
        //method check if student already submit his/her projects ranked list
        private bool studentSubmitted(string StudentId) {
            da = new SqlDataAdapter("SELECT * FROM StudentRank WHERE StudentId = '" + StudentId +
                "' AND Rank <> 0", conn);
            DataSet ds = new DataSet(); da.Fill(ds,"StudentRank");
            if (ds.Tables[0].Rows.Count != 0) return true;
            return false;
        }//method to displays list of projects which sorted by SubSift with regard to any student search
        private void showResults(string SubSiftResult) {
            //create datatable for Accordion
            dt = new DataTable("projects");
            dt.Columns.Add(new DataColumn("Order")); dt.Columns.Add(new DataColumn("ProjectId"));
            dt.Columns.Add(new DataColumn("Title")); dt.Columns.Add(new DataColumn("Abstract"));
            XmlDocument matchedResult = new XmlDocument(); matchedResult.Load(new StringReader(SubSiftResult));
            //select all project items below <match> tag
            XmlNodeList projectNodes = searchNode.SelectNodes("item");
            int i = 0;
            foreach (XmlNode project in projectNodes) {
                DataRow row = dt.NewRow(); i++; // 'txtBestFit' is number of projects displayed for current student
                if (i > Convert.ToInt32(txtBestFit.Text)) break;
                    //extract 'project Id'  from SubSift result then find matched project from DB
```



```
                    string id = project.SelectSingleNode("id").InnerText;
            XmlNode searchNode = matchedResult.SelectNodes("result/match")[0];  //select the 'search' node
                    da = new SqlDataAdapter("SELECT * FROM Project WHERE ProjectId=" + id, conn);
                    DataSet ds = new DataSet(); da.Fill(ds, "Project");
                    if (ds.Tables[0].Rows.Count != 0) {
                        DataRow projectRow = ds.Tables["Project"].Rows[0];
                        row["Order"] = i;     // add order number to row
                        row["ProjectId"] = projectRow["ProjectId"];    //add project id
                        row["Title"] = projectRow["Title"];    //add project Title
                        row["Abstract"] = projectRow["Abstract"];    //add Abstract
                    } dt.Rows.Add(row);  } }
            ProjectAccordion.DataSource = dt.DefaultView; ProjectAccordion.DataBind();
        }// button when student want to see all projects
        protected void btnShowAllProjects_Click(object sender, EventArgs e)
        {
        Response.Redirect("https://www.cs.bris.ac.uk/project/proposals/index.jsp?ticket=ST-10677980-
dWW1Fkad5AoqtjFFCcit-nAC10040C2095");
        }
        //button for adding the student selected projects to his/her favourite list for later ranking them
        protected void btnAddToFavorite_Click(object sender, EventArgs e) {
            conn.Open();
     // loop on all projects on page,checking if any project is selected by student to be added to favourite list
            for (int i = 0; i < ProjectAccordion.Panes.Count; i++ ) {
                AccordionPane ap = ProjectAccordion.Panes[i];
                CheckBox c = (CheckBox)ap.FindControl("cbSelect");
                if (c.Checked) { string id = ((Label)ap.FindControl("lblProjectId")).Text;
                    SqlCommand cmd = new SqlCommand(); cmd.CommandType = CommandType.Text;
                    cmd.Parameters.AddWithValue("@ProjectId", id);
                    cmd.Parameters.AddWithValue("@StudentId", ddlStudent.SelectedValue.ToString());
                    cmd.CommandText = "INSERT INTO StudentRank (ProjectId, StudentId, Rank) " +
                                      "VALUES (@ProjectId, @StudentId, 0)";
                    cmd.Connection = conn;  try { cmd.ExecuteNonQuery(); } catch (Exception) { }
            }   } conn.Close(); GridView1.DataBind();
            if(GridView1.Rows.Count!=0) btnStudentRank.Visible = true;
        }//button for submiting the student projects ranked list
        protected void btnStudentRank_Click1(object sender, EventArgs e) {
            string StudentID = ddlStudent.SelectedValue.ToString();//reading student id from dropdown list
            int rank=0;
            foreach (GridViewRow r in GridView1.Rows) { //loop on ranked projects
                if (((TextBox)r.FindControl("txtRank")).Text != "")
                    rank = Convert.ToInt32(((TextBox)r.FindControl("txtRank")).Text);//read rank for this project
                string ProjectID = ((Label)r.FindControl("lblProjectId")).Text;//read project id for this project
               SQL.UpdateStudentRank(StudentID, ProjectID, rank);//insert into DB
            } DeleteStudentRank(StudentID); //remove temporarily favourite list from DB
            GridView1.DataBind(); txtConfirmation.Text = "Your project list has been successfully submitted";
            btnStudentRank.Visible = false;  btnAddToFavorite.Visible = false;
       }//method removing the temporarily saved fovourite list of projects before submission ranked projects
        private void DeleteStudentRank(string StudentID) {
            conn.Open(); SqlCommand cmd = new SqlCommand();
            cmd.CommandType = CommandType.Text; cmd.Connection = conn;
            cmd.Parameters.AddWithValue("@StudentId", StudentID);
            cmd.CommandText = "DELETE FROM StudentRank WHERE "+"StudentId = @StudentId AND Rank = 0";
            cmd.ExecuteNonQuery(); conn.Close(); }
    //when student select his/her user name from dropdown list the system will fetch student full name from DB
        protected void ddlStudent_SelectedIndexChanged1(object sender, EventArgs e) {
            conn.Open();
            da = new SqlDataAdapter("SELECT FName, LName FROM Student WHERE StudentId = '" +
ddlStudent.SelectedValue + "'", conn);
            DataSet ds = new DataSet(); da.Fill(ds, "Student");
            if (ds.Tables["Student"].Rows.Count != 0) {
                DataRow r = ds.Tables["Student"].Rows[0]; lblStudentName.Text = r["FName"] + " " + r["LName"];
            } conn.Close(); }
        protected void GridView1_RowDataBound(object sender, GridViewRowEventArgs e) {
            if (e.Row.RowType == DataControlRowType.DataRow) {
                string projectId = ((Label)e.Row.FindControl("lblProjectId")).Text;
                da = new SqlDataAdapter("SELECT Title FROM Project WHERE ProjectId=" + projectId, conn);
                DataSet ds = new DataSet(); da.Fill(ds, "ProjectTitle");
                DataRow r = ds.Tables["ProjectTitle"].Rows[0];
                ((HyperLink)e.Row.FindControl("lblTitle")).Text = r["Title"] + "";
}         }}
```