Fall 2012

An approach to solve job shop scheduling problem

Shashidhar Reddy Karnati
San Jose State University

An approach to solve job shop scheduling problem

A Writing Project

Presented to

The Faculty of the department of Computer

Science San Jose State University

In Partial Fulfillment

Of the Requirements for the

Degree Master of Computer Science

By

Shashidhar Reddy Karnati

Dec 2012

SAN JOSE STATE UNIVERSITY

The Undersigned Writing Project Committee Approves the Writing Project Titled

An approach to solve job shop scheduling problem

By

ShashidharKarnati

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

_____

Dr. Soon Tee Teoh, Department of Computer Science 12/13 /2012

_____

Dr. Chris Tseng, Department of Computer Science 12/13/2012

_____

Mr. Keerthi Vardhan Gouni, Fuzebox corporaton 12/ 13/2012

ABSTRACT

"A biotechnology device manufacturer needs to devise effective scheduling algorithms for its testing devices. A device is a configuration of machines, each of which performs a specific task, such as washing, reading and cleaning. These devices are used to test human samples to diagnose diseases like cholera, malaria etc. Each test is a job, which is to be processed on these machines for a specific amount of time. Every job has its own pre defined sequence. These samples are to be processed simultaneously on machines owing to constraint that as soon as one machine completes processing a sample, it should be immediately processed by another machine. This constraint is significantly known as no-wait constraint. Given a set of jobs the web application assigns an optimal start time for each job owing to no-wait constraint. This results in reducing the overall time taken to process the jobs, which is formally known as makespan. The main objective of the project is to minimize the makespan.

The application is specific to laboratory platform, which helps them to test the samples in optimal time. The heuristic, which I have implemented, is designed with future advancements in mind. The application can be extended to test different heuristic procedures by keeping the time tabling intact. The development environment to be used in this project will require Microsoft Visual Studio, C#, ASP.NET, and other real time chart tools like Microsoft Silverlight."

Table of Contents

1.0 Introduction and Background

A biotechnology device manufacturer needs to devise an algorithm to test their devices. These devices are used to test blood samples to diagnose diseases like Malaria etc. A device is a configuration of machines, which performs a particular task such as adding reagent, washing and cleaning. These samples are collected in microplate. A microplate is a collection of 96 samples of the same kind of disease to be diagnosed. A test on the sample is called a job. Jobs contain operations such as cleaning and washing to be performed on a particular machine for a given processing time. These jobs are currently being processed sequentially, which is very time consuming, as the machines are being idle. As soon as one machine completes a job the next machine should immediately start processing the job. There should be no wait between two consecutive operations of a job, which is significantly referred to as job scheduling with no wait constraint. I need to implement an algorithm such that given a set of jobs to be processed on a set of machines; I need to output the schedule of jobs with optimal start times assigned to each job, satisfying the no wait constraint. A job shop problem with no wait constraint is an NP-hard problem, which cannot be solved, in polynomial time. There are no exact algorithms to solve these kinds of problems in a given time. It is very difficult to calculate the optimal schedule even for two jobs and two machine problems; hence, we look for heuristics to solve the job shop problem with no-wait constraint. The heuristic gives us a feasible schedule of jobs with start times assigned to each job, guaranteeing the efficient use of resources.

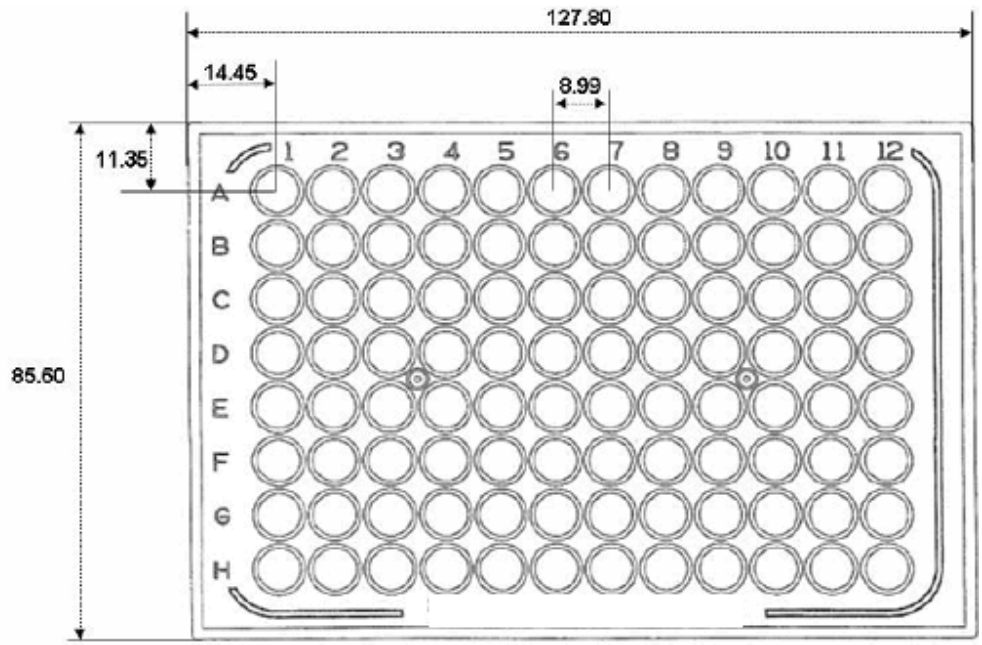


Fig 1: A microplate

I have studied different heuristics and implemented the Tabu search heuristic. I compared and contrasted the Tabu search with other heuristics based on space and time complexity. The Tabu search is easy to understand and can be extendible. I have also implemented the Exhaustive enumeration technique to compare it with the results of the Tabu search. The web application is developed particularly for a laboratory platform. This web application can be used by laboratories to make efficient use of the resources. The application can also be used in other manufacturing and production environments with slight modifications to the problem. The user interface requirements are that user should have the flexibility to input jobs, machines and their processing times. The output should be a schedule of jobs with their respective start times, which the laboratories use to schedule their jobs to minimize the overall processing time to complete all the jobs.

1.1. Objective

The first objective of the project is to implement a job scheduling heuristic algorithm to address a resource allocation and job scheduling in the biotechnology automation platform. Secondly to implement to implement a web application, which accepts set of jobs as input and displays the sequence of jobs with their optimal start times.

1.2. Scope

The scope of the project is limited to automated laboratory platform to make efficient use of resources and to minimize the processing time. The scope of the project is limited to single instance of each machine and can be extended to multiple instances of each machine in the future.

1.3. Software development model:

I have followed the Rapid Application Development model, which allows us to change the requirements and created environment for faster development of software. After each stage, I have tested the prototype with requirements. Rapid application development allows me to have minimal planning. After working with rapid application prototype development, I feel that it is extremely suitable for small teams and faster software development.

2.0 Literature Review

Tabu Search is used widely on machine scheduling and job-shop scheduling problems. In his study Glover (1990)[7], stated that Widmer & Hertz's (1990)[8] applications of tabu search to flow shop sequencing problems succeeded in obtaining solutions superior to the best previously found by applying a range of methods in about 90% of the cases. Tabu search has shown superior results in other recent applications as well. Blazewicz et al. (2008)[9] presented two meta-heuristics Tabu search, and variable neighborhood search (VNS) for the two-machine flow-shop problem with weighted late work criterion and common due date. Initial solutions were generated by Johnson's algorithm (1954)[10] or also called as list scheduling algorithm which is a constructive method, that builds a solution by executing jobs selected according to a given priority dispatching rule.

There are many other application fields and problems in which tabu search is used. For example: Cell planning with capacity expansion in mobile communications (Lee & Kang, 2000[]), application-level synthesis methodology for multidimensional embedded processing systems (Alippi et al. 2003[11]). Cogotti et al. (2000)[12] performed a comparison of optimization techniques for Loney's Solenoids Design and proposed an alternative tabu search algorithm. Emmert et al. (2003)[13] have shown an effective way of bi-partitioning electrical circuits using tabu serach. It was stated that tabu search offered quick convergence to good partitioning solutions for circuits in the range of their application. Their algorithms show dramatic improvement in execution time with good solution quality as compared to a random move SA approach. They also mention that their placement method is suitable for quickly initializing the inputs to other nondeterministic placement algorithms. Rajan et al. (2003)[14]

proposed a neural-based tabu search method for solving unit commitment problem. Blazewicz et al. (2000) [15] proposed a tabu search-based algorithm for DNA sequencing in the presence of false negatives and false positives. Corberan et al. (2000)[16] studied a mixed rural postman problem in which tabu search was used. Ahr &Reinelt (2005)[17] presented a tabu search algorithm for the min-max Chinese postman problem.

3.0 Problem Description:

3.1 Terminology

For terminology please refer to appendix A

3.2 Problem Statement:

In my problem there are set of jobs and set of machines. Each job has a sequence of operations that are to be processed on machines for a given duration. These jobs are to be scheduled in such a way that the makespan is minimized. Jobs are subjected to the following constraints:

- No-Wait constraint.
- No two jobs must be processed by a machine at a time.
- No two machines should perform on a job at a given time.

The problem is divided into two sub problems:

Sequencing: A processing sequence of an optimal schedule is found for a given no-wait job shop problem.

Timetabling: A feasible set of start times of the jobs is found in order to minimize the makespan for the processing sequence. These feasible set of start times obtained from the sequencing problem.

3.3 Mathematical Representation:

For mathematical representations of the problem please refer to Appendix B.

4.0 Project Requirements:

The primary goal of the project is to develop a web application that takes number of jobs, and machines as input from user .The output will be a feasible schedule with set of start times assigned to jobs. In order to accomplish this, there are two major tasks: Selecting an appropriate algorithm which best suits the problem and developing a web application.

5.0 Solution approaches:

The primary goal of project is to select an algorithm. There are many instances of a job shop problem, no wait constraint being most important of them. There are only a few algorithms, which talk about no-wait constraint in job shop problems; most of the job shop problems are addressed using heuristic approaches, such as genetic algorithms and the Tabu search. The two algorithms that address the no-wait constraint in most efficient way are The Complete local search with limited memory and job shop scheduling with the Tabu search. I have studied both these algorithms and implemented tabu search because of its simplicity and ease of implementation. After implementation of the Tabu search heuristic I wanted to check the correctness of algorithm by testing it with different test cases. Initially I tested the efficiency of the algorithm with my own test cases; I started with simple test cases and later increased the complexity to check the correctness, later I tested the prototype with standard test cases from the Operations and Research library. These test cases are standard test cases, which are documented and used to as a bench mark to test efficiency of different job shop algorithms. The results are briefly explained in the evaluation section.

5.1 Why the TABU Search:

I compared and contrasted tabu search with complete local search based on cost factors such as time complexity and space complexity. In terms of space, the Tabu search fares well, as the complete local search needs three segments of memory to be allocated. I wanted my application to be scalable and easily maintainable; tabu

search supports the above stated criteria. Tabu search supports a flexible frame, which allows the future users to improve upon the algorithm.

For Complete local search with limited memory refer to Appendix C.

5.2 Algorithm:

Tabu search is a well-known algorithm for solving the combinatorial optimization problems. The algorithm searches the space 'S' of all possible sequences of a given problem. Using the cost function, a sequence within the space is evaluated, and a feasible solution is found. The parameters used in the algorithm are:

Initial solution: The initial sequence is generated using well known heuristic approaches. Some of the heuristic methods are: shortest processing time (SPT), longest processing time (LPT), shortest machine time (SMT), longest machine time (LMT) and Random.

Neighborhoods:

The Neighbor of a given sequence is the sequence which is derived from the original sequence by extracting 'k' consecutive jobs from the sequence and inserting in the same order at different positions in the rest of the sequence. This is referred as the 'k' insertion neighborhood. The concept of tightly intertwined jobs is used as the criteria for selecting the 'k' consecutive jobs. I have used two insertion neighborhoods for finding the next neighbor. I selected two jobs, which are tightly packed with each other, and took the combination of these two jobs. These tightly

packed jobs are inserted between all the positions to calculate the make span. If I have a 6x6 problem then I have five iterations for finding neighbors. The steps of the algorithm are explained below for better understanding.

Steps in Algorithm:

1. Define number of loops (L), a counter (V), a TABU list of length (T), a memory slot B for the best solution so far and a neighborhood (N).
2. Choose an initial solution i ∈ S, set B = i *and* V= 0.
3. Search the neighborhood N (i) for the best solution i^1= N (i)that is not in TABU list and set V= V +1. Enter i^1 into the TABU list and over-write an element by FIFO if necessary.

   If no such solution i^1 exists, then stop.

4. If f (i^1) < f (B), if the makespan of i^1 is less than the value which is in TABU list, then set B= i^1, V=0, i= i^1 and go to step 3.
5. If V=L, then stop.
6. Set i=i^1 and go to step 3.

The function "f" computes the makespan for the sequence from step 3.

6.0 Software Requirements:

Microsoft Visual Studio: Microsoft visual studio is an integrated development environment from Microsoft. It can be used to develop a console and graphical user interface applications, windows form applications etc.

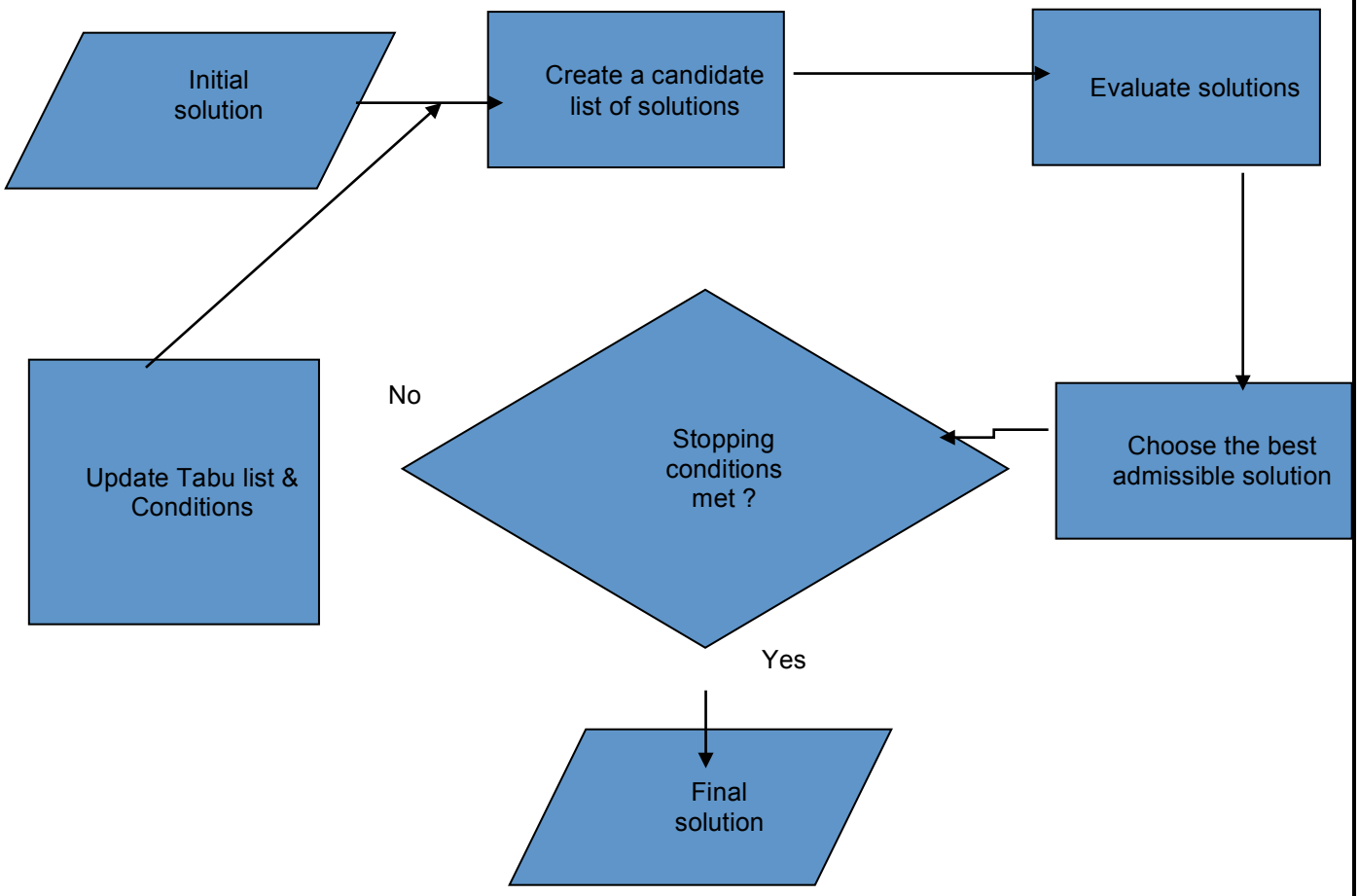
Microsoft ASP.NET: ASP.net allows programmers to build dynamic websites web applications and web sites. ASP.NET is built on the common language runtime. It allows programmers ASP.NET code using any supported .NET language.

Microsoft C#: C# is a type-safe, objected-oriented language. It allows creation of windows applications, web services and controls etc.

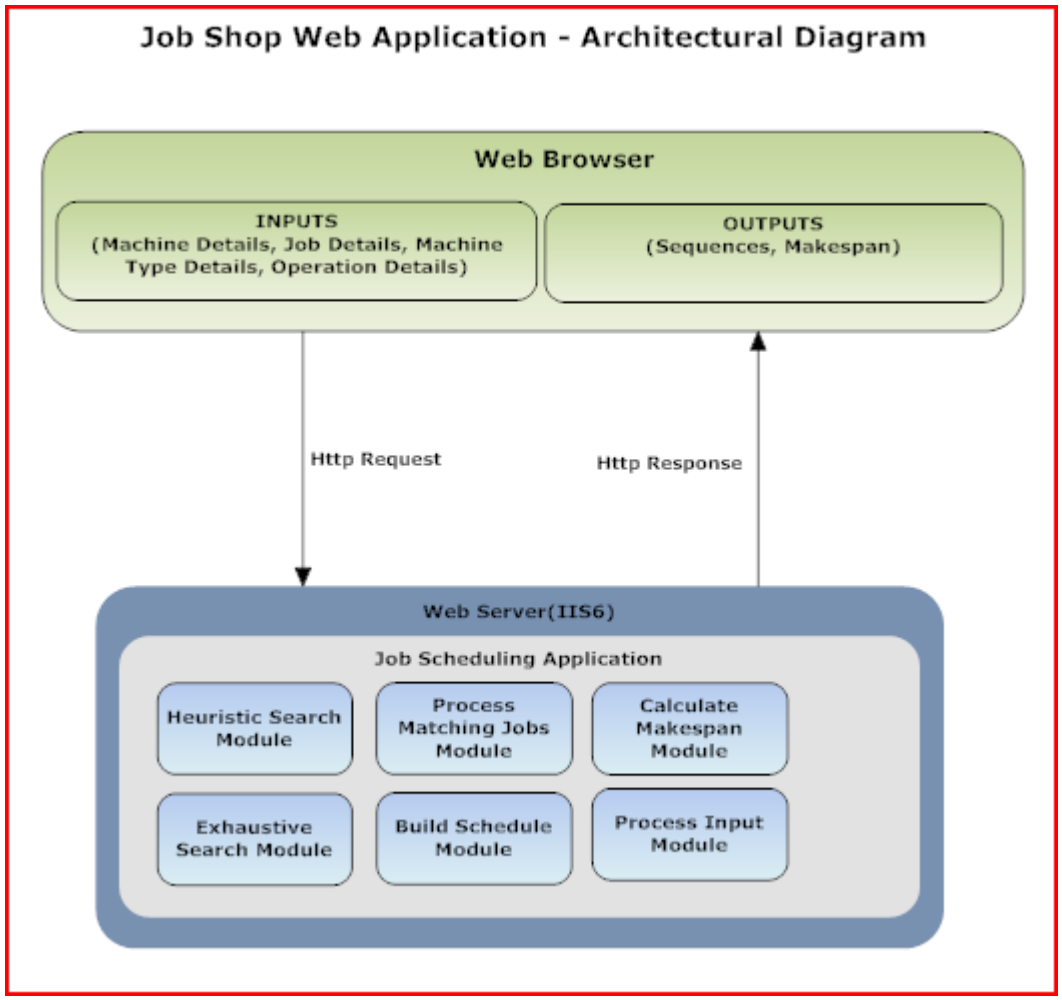
CSS, Java script, Ajax: CSSis used to style web pages written in HTML and XHTML, we used java script to provide enhanced user interface. Ajax uses a combination of HTML and CSS to mark up and style information.

7.0 Design and Implementation:

7.1 Flow Chart

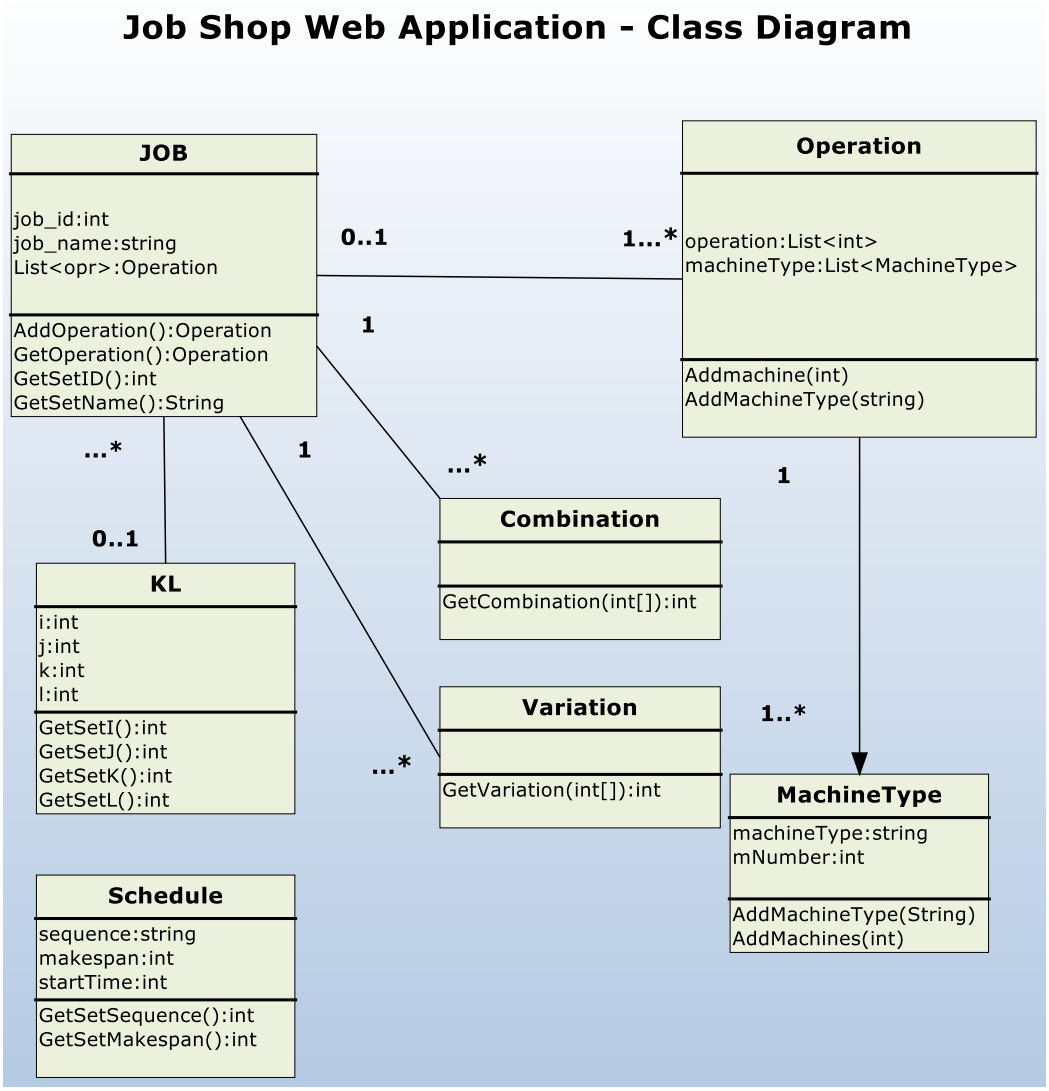


7.2 Architecture Diagram:



The client is a web browser and the server is running the job shop scheduling web application. The job shop scheduling web application contains modules that receive the input, process the input and send the response back to the client.

7.3Class Diagram:



The class diagram illustrates the classes used to implement the job shop web application. Each class is described below.

JOB- This class stores the properties of each job like job name, job I.D etc..,

Operation – It stores the operations of a job.

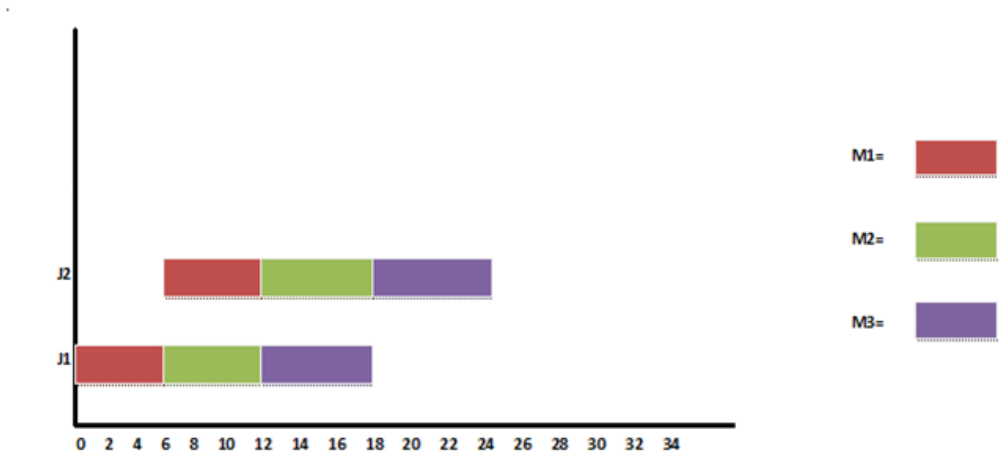
KL – this class stores indices of matching operations from the pair of jobs.

Combination, Variations - this is used for different permutation, combinations of jobs.

Schedule- it holds the makespan and start time of each sequence.

Machinetype- this class is used to get the machine type of the machine.

7.4 Gantt Chart



Scheduling of jobs with no wait constraint.

7.5 Implementation:

7.5.1 Heuristic Function:

The application calls Heuristic function in order to calculate the initial sequence. The processing times of each job is the total time of the every operation in the job, the function then sorts the jobs based on their processing times.

```
protected void Heuristic()
    {
        //Tabu List
        List<string> tabu = (List<string>)Session["tabuList"];
        List<string> tabuII = (List<string>)Session["tabuListII"];

        string tabuSequence = null;

        //order job from lowest to highest processing times
        List<int[]> start = new List<int[]>();

        //new List
        List<Job> sequence = new List<Job>();

        //SortedList heuristicList = new SortedList();
        List<int> heuristicList = new List<int>();
        List<int> heuristicList2 = new List<int>();
        int heuristicTime = -1;

        List<Job> job = (List<Job>)Session["jobPointer"];
        for (int x = 0; x < job.Count(); x++)
        {
            int[] st = new int[2];
            int oprCount = job[x].Operation.Count() - 1;
            heuristicTime = ProcessingTimeH(x, oprCount);

            st[0] = heuristicTime;
            st[1] = x;
            start.Add(st);
        }
        //sort initial sequence in order of processing times for each job
        start.Sort(delegate(int[] a, int[] b)
        {
            int xdiff = a[0] - b[0];
            if (xdiff != 0) return xdiff;
            return a[0] - b[0];
        });

        foreach (int[] seq in start)
        {
            sequence.Add(job[seq[1]]);
        }

        //Create initial sequence in order 0,1,2,3
        for (int s = 0; s < job.Count; s++)
        {
            tabuSequence += s.ToString();
        }
        tabuII.Add(tabuSequence);
        Session["sequence"] = sequence;
        Session["initialSequence"] = sequence;

    }
```

7.5.2 Tabu Function:

The tabu function then computes candidate list of solutions based on the make spans. The sequence generated by the tabu function is then processed by another function to check for conditions computed by the tabu function.

```
//Calculate makespan - for each variation
for (int x = 0; x < tempList.Count; x++)
{
    //initialise ratio for each calculation
    List<string[]> ratio = new List<string[]>();

    tabuSequence = null;
    //sequence = new List<Job>();
    List<Job> tempSequence = new List<Job>();

    foreach (object list in tempList[x])
    {
        list.ToString();
        tabuSequence += list.ToString();
        tempSequence.Add(sequence[(int)list]);  //Maintains order of initial sequence eg 210 or 012
    }
    //Variation sequence eg 01,12,23
    Session["tempSequence"] = tempSequence;
    int variationTime = 0;
    //Session["tabuList"] = tabu;x
    //Session["sequence"] = sequence;
    for (int v = 0; v < tempSequence.Count(); v++)
    {
        int oprCount = tempSequence[v].Operation.Count() - 1;
        variationTime += ProcessingTimeV(v, oprCount);
        //heuristicList.Add(heuristicTime, x);
        if (v == 1)
        {
            string[] vp = new string[2];
            vp[0] = variationTime.ToString();
            vp[1] = tabuSequence;
            ratio.Add(vp); //total processing time of tempsequence

        }
    }
```

7.5.3 Process Tabu Function:

The Process tabu fuction is the main part of the application that checks if the sequences met the conditions and then rearranges the sequence to compute the final sequence.

```
protected void ProcessTabu()
    {
        string newTabu = null;
        string tabuSequenceII = null;
        //Original jobs
        List<Job> job = (List<Job>)Session["initialSequence"];
        //Initial sequence
        List<Job> sequence = (List<Job>)Session["initialSequence"];
        //process original sequence first
        //Initial sequence
        Session["tempSequence"] = sequence;
        ProcessKLII();
        CalculateMakespanII();
        //Tabu List e.g 012, or 210 or 0123456789
        List<string> tabuII = (List<string>)Session["tabuListII"];
        //TabuProcessed e.g 01 or 12 or 23
        //List<string> tabuSequenceII = (List<string>)Session["tabuProcessed"];
        tabuSequenceII = (string)Session["tabuProcessed"];
        //Remove tabu processed from initial sequence
        string temp = tabuSequenceII;
        newTabu = tabuII[0].Replace(temp, "");
        //K interstion of newTabu variable
        for (int x = 0; x < newTabu.Count() + 1; x++)
        {
            string newSequence = newTabu.Insert(x, temp);

            //new sequence
            sequence = new List<Job>();
            foreach (char charSeq in newSequence)
            {
                int intSeq = Convert.ToInt32(charSeq.ToString());
                sequence.Add(job[intSeq]);
            }
            //Find existing sequence in Tabu list
            string result = tabuII.Find(delegate(string tb) { return tb.ToString() == newSequence; });
            if (result == null)
            {
                tabuII.Add(newSequence);
                Session["tabuList"] = tabuII;
                Session["tempSequence"] = sequence;
                ProcessKLII();
                CalculateMakespanII();
            }
        }
    }
```

8.0 Evaluation:

Evaluation is the most important section of the project. I have implemented both exhaustive enumeration and tabu search algorithms as part of my prototype. I started testing these prototypes using simple test cases that can be solved manually. I compared the results of the prototype with manual calculations, and the results were accurate. Later I tested these prototypes with standard test cases, which are used in the tabu search paper. These standard test cases are job shop problems, which have been documented in the Operations Research library and are used for testing different job shop algorithms in the industry. The exhaustive enumeration results for these standard test cases were accurate, and there was a minimal variation in the heuristic results when compared to that of the paper. The minimal variation is due to the initial sequence I randomly generate to start the neighborhood search. I have prepared some test cases particular to the job shop problem with no wait constraint. I also compared and contrasted exhaustive enumeration with tabu search heuristic using different parameters, such as time and size of the problem. I recommend using exhaustive enumeration when there are a less number of jobs, because as the number of jobs increases, the number of permutations that the algorithm has to calculate increases, which results in degraded performance. The Tabu search heuristic works well if there are more jobs. After evaluation of the prototype, I was able to deliver a job shop scheduling web application.

The user interface of the web application is very user friendly. I have spent a considerable amount of time on user interface, so that the user has the flexibility to input the problem using a text file or manually. Input using a text file allows the user to modify the problem in the text file. Hence, I have an optimized input format. I measured the time taken for executing each test case for both exhaustive enumeration and tabu search. For a given standard 6x6 job shop problem, exhaustive enumeration takes 6.37 Msec where as the tabu search heuristic takes 2 Msec. the output is a sequence of jobs with their staring times currently, I am displaying the best sequence with the minimum make span. Provided the results of the test cases in Appendix E.

8.1 Test Cases:

The input test cases below consist of a matrix looking sequence of numbers. The complete line represents a job. Each operation is separated by a ",", of the pair, first number represents the machine type and the second represents the duration of operation on that particular machine. The input test cases can be manually entered, by entering the machine type, jobs and operations available on the left or by uploading a text file that consists of the test cases like below for easier usage.

The Output consists of both Heuristic and Exhaustive processing results for a better comparison of the processing sequence, the make span and the calculation time. The Tabu search algorithm clearly has resulted in a better processing sequence and the calculation time is drastically reduced, for example in test case 3, the exhaustive computation has resulted in 2497 milliseconds of computation time in comparison to 5 milliseconds of heuristic processing time.

8.1.1Test case 1:

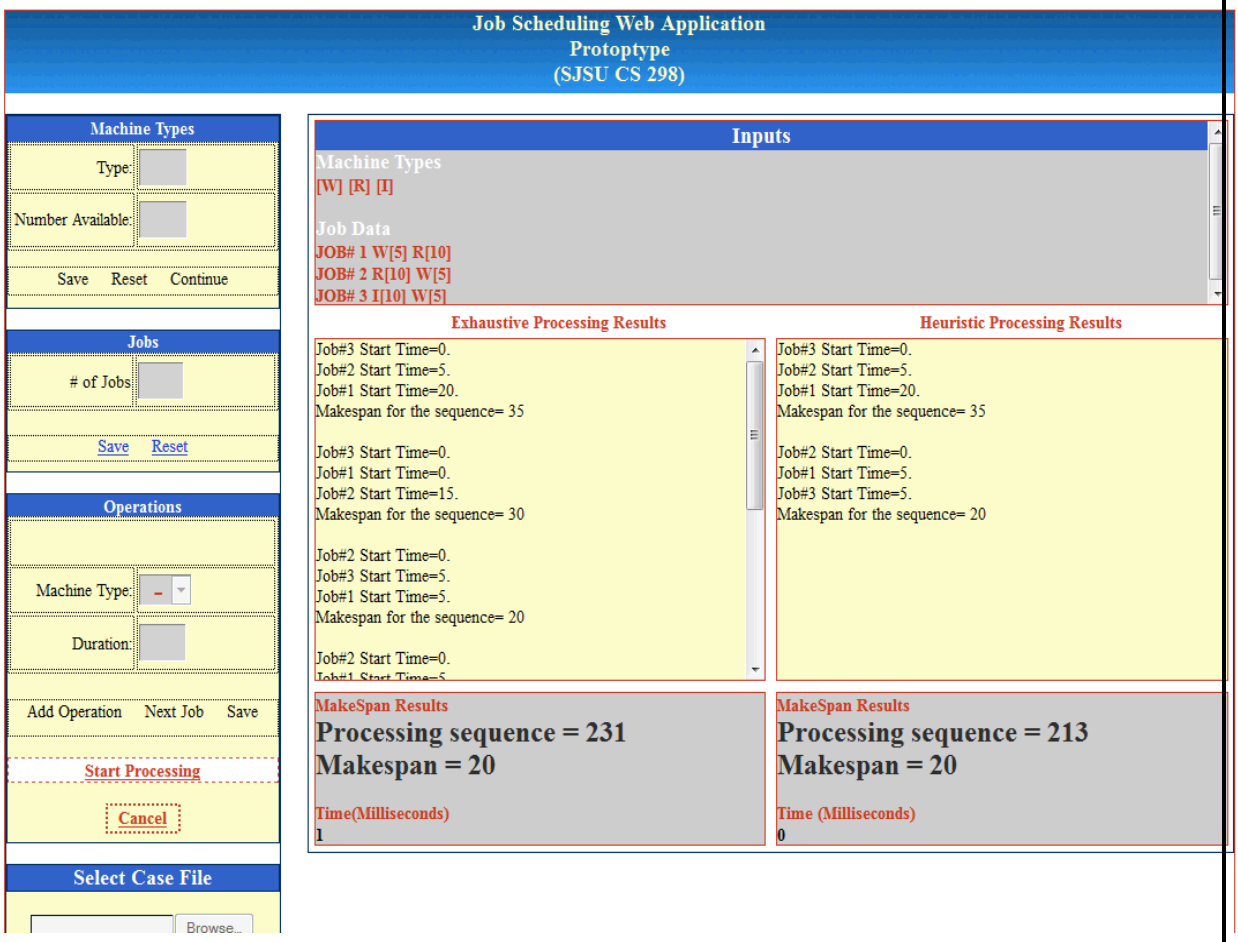
This is the test case for 3x2, which means 3 jobs, each with 2 operations and their duration.

Input sequences:

0 5,1 10

1 10,0 5

2 10,0 5



8.1.2 Test case 2:

This is the test case 4x4

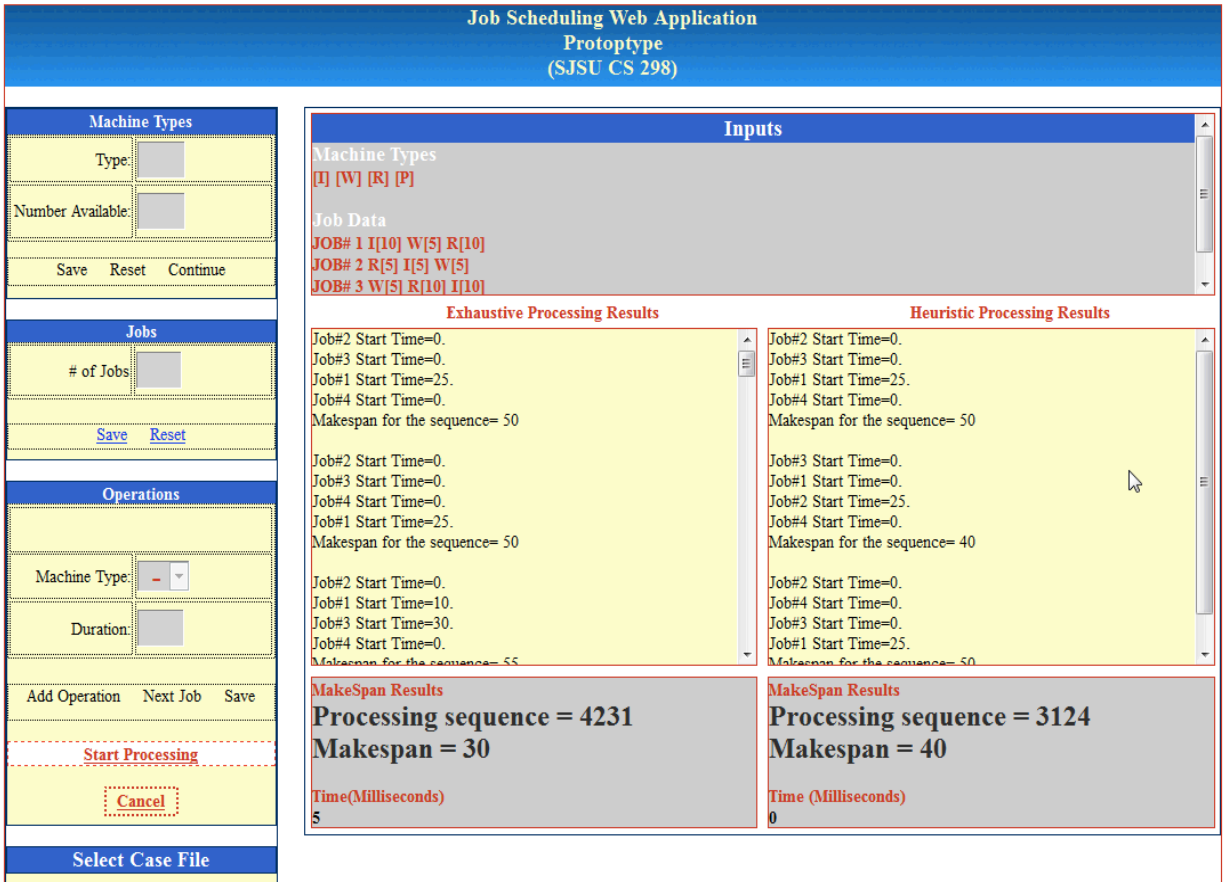
Input Sequence:

2 10,0 5,1 10

1 5,2 5,0 5

0 5,1 10,2 10

3 10,3 5,3 15



8.1.3 Test case 3:

This is the test case for 6x6

Input Sequence:

2 1,0 3,1 6,3 7,5 3,4 6

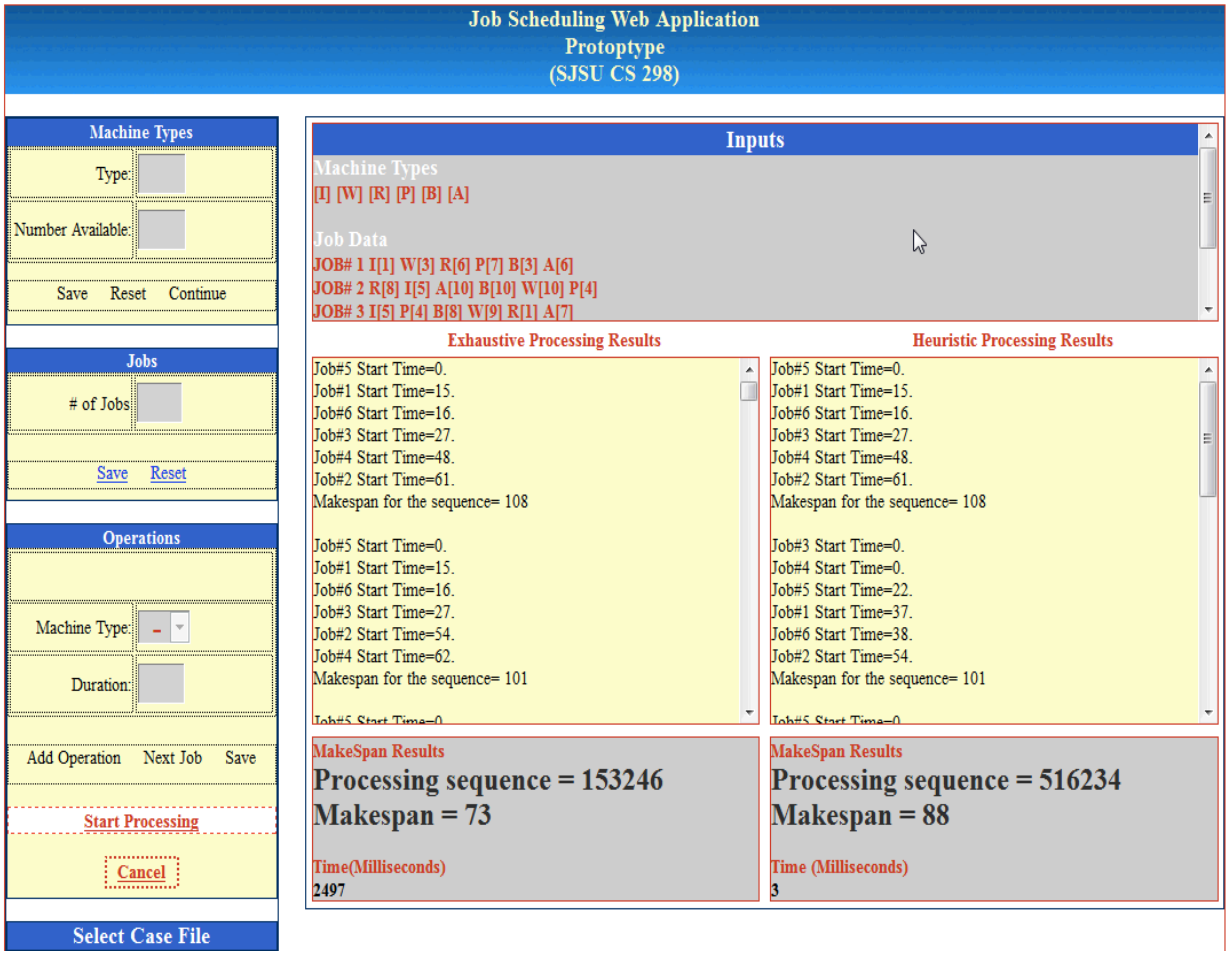
1 8,2 5,4 10,5 10,0 10,3 4

2 5,3 4,5 8,0 9,1 1,4 7

1 5,0 5,2 5,3 3,4 8,5 9

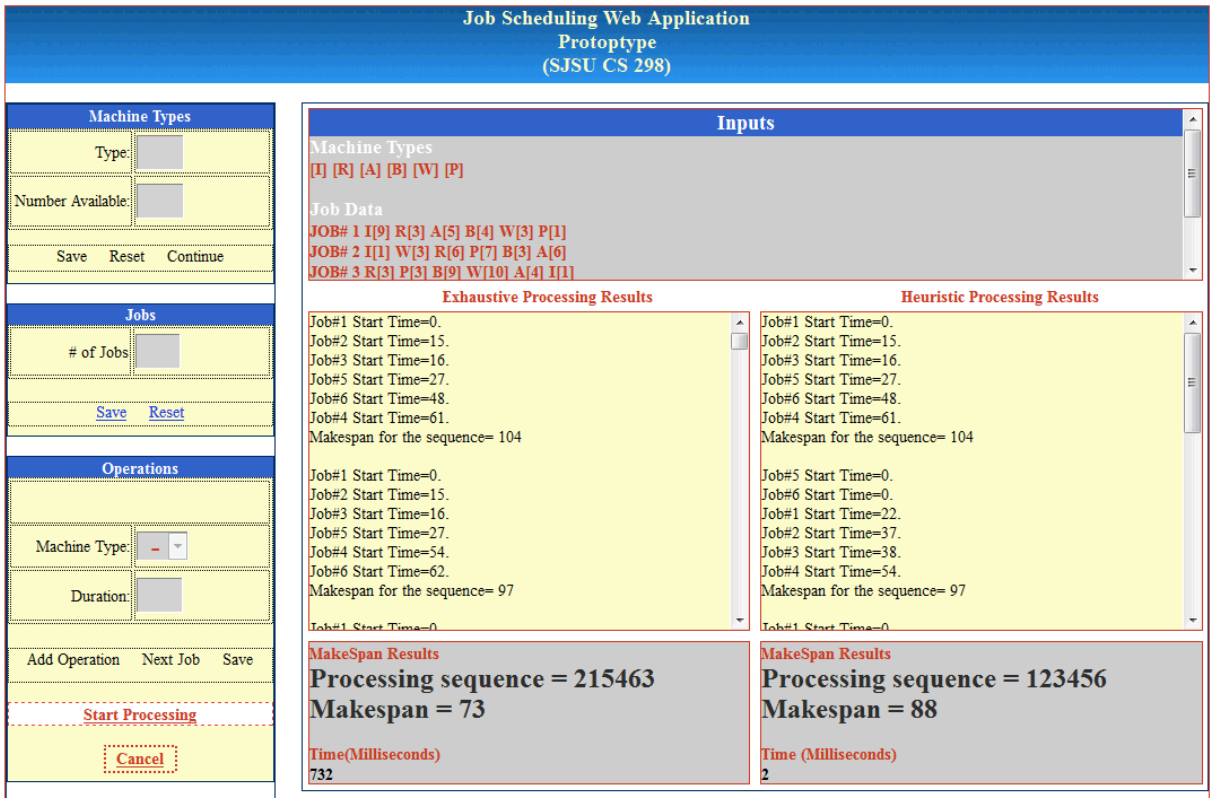
2 9,1 3,4 5,5 4,0 3,3 1

1 3,3 3,5 9,0 10,4 4,2 1



8.1.4 Analysis

Based on the below analysis rearranging the data to be the best case scenario, the time taken to compute the makespan is lesser than the worst case scenario, but did not affect the makespan.



8.1.5 Comparisons

For the below heavy test data, Tabu search results in a relatively better makespan with very less computational time

2 44,3 5,5 58,4 97,0 9,7 84,8 77,9 96,1 58,6 89

4 15,7 31,1 87,8 57,0 77,3 85,2 81,5 39,9 73,6 21

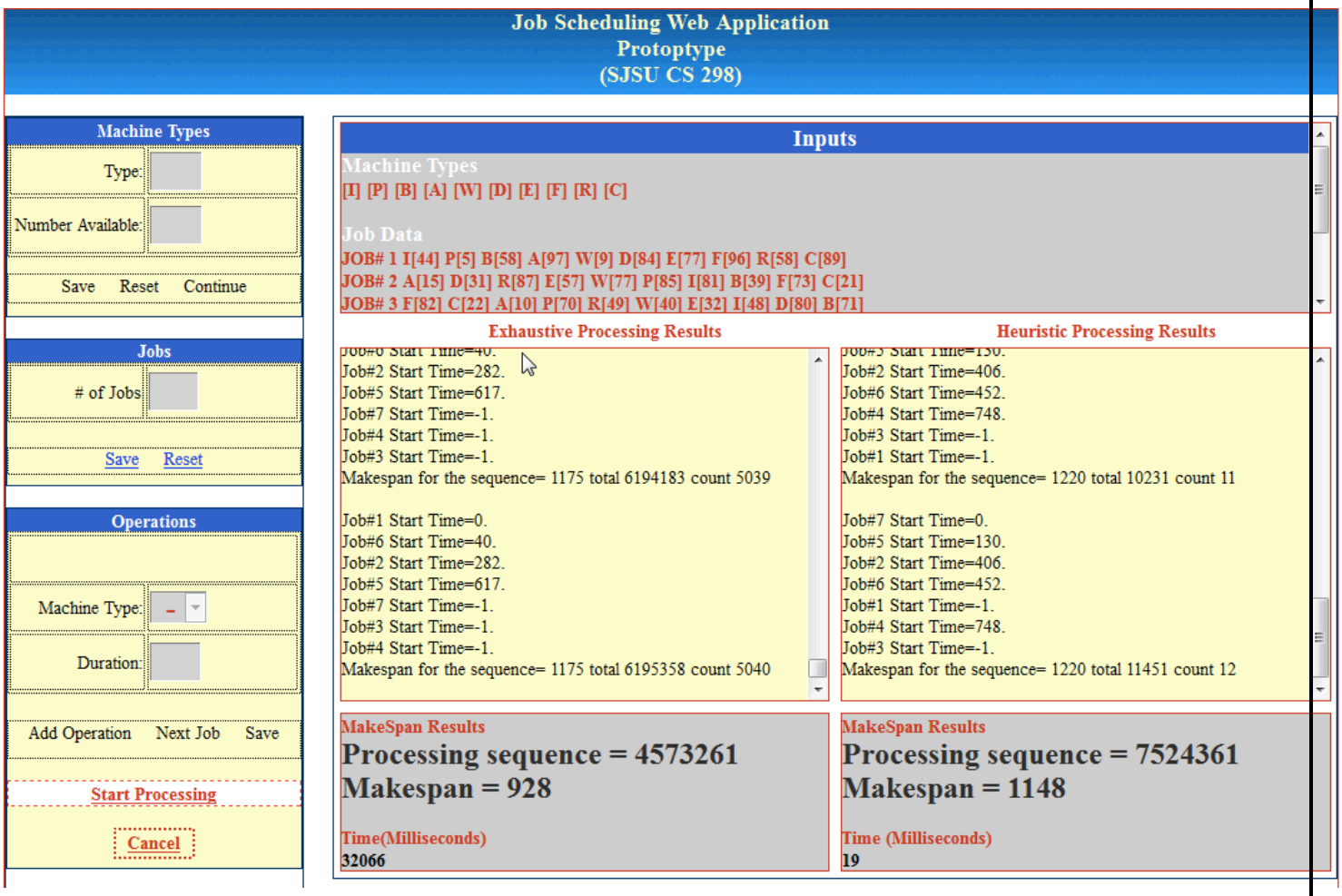
9 82,6 22,4 10,3 70,1 49,0 40,8 32,2 48,7 80,5 71

1 91,2 17,7 62,5 75,8 47,4 11,3 7,6 72,9 35,0 55

6 71,1 90,3 75,0 64,2 94,8 15,4 12,7 67,9 20,5 50

7 70,5 93,8 77,2 29,4 58,6 93,3 68,1 57,9 7,0 52

6 87,1 63,4 26,5 6,2 82,3 27,7 56,8 48,9 36,0 95



Comparisons	**Tabu Search**	**Median of the tabu list**	**Exhaustive**	**Median of all Permutations**
Make Span	1148	11451 / 12 = 954	928	6195358 / 5040 = 1230
Time to compute (in milli seconds)	19	Based on the tabu list.	32066	Based on the exhaustive results.

Based on the above comparisons table for heavy test data, median of tabu list yields comparatively better makespan in very less computational time when compared to exhaustive.

9.0 Implementation Issues:

I had few issues when implementing the tabu search heuristic. I was not able to understand the concept of k-insertion and implemented my own understanding of the algorithm. I faced few logical errors in the development of the prototype. The major problem I faced was in implementing non-delay timetabling, which satisfies the no wait constraint specified in the requirements. I am glad to say that the non-delay timetabling is working well.

10.0 Lessons Learned:

From the academic perspective, I have learnt new concepts about job shop scheduling and I was exposed to different job shop problems and their solution approaches. I also understood how these problems could be modeled using a disjunctive graph approach. From the project management perspective, I have understood the role of a software development model in completing a project successfully.

11.0 Conclusion and Future Work:

I have done all the hard work and have successfully implemented the initial project requirements. I have designed the prototype in such a way that it is flexible and one can implement a new heuristic algorithm by keeping the non delay timetabling intact. The code and technical details are well documented, such that readers can understand the functionality of the system.

The scope of the project can be extended to multiple instances of each machine. I recommend modeling the job shop problem using the Disjunctive graph model as it is widely used at the industry level.

12.0 References:

[1] C. Rajendran, "A no-wait flowshop scheduling heuristic to minimize makespan," Journal of the Operational Research Society, vol. 45, pp. 472-478, 1994.

[2] C. Schuster, "No-wait job shop scheduling: Tabu search and complexity of subproblems," Mathematical Methods of Operations Research, vol. 63, pp. 473-491, 2006.

[3] J. Zhu, et al., "Complete local search with limited memory algorithm for no-wait job shops to minimize makespan," European Journal of Operational Research, vol. 198, pp. 378-386, 2009.

[4] C. Sriskandarajah and P. Ladet, "Some no-wait shops scheduling problems: complexity aspect," European Journal of Operational Research, vol. 24, pp. 424-438, 1986.

[5]. P. Brucker, *Scheduling algorithms*: Springer Verlag, 2007.

[6] http://las.perkinelmer.com/Catalog/ProductInfoPage.htm?ProductID=6005176

[7] Glover, F. (1989). Tabu Search – Part II, ORSA Journal on Computing, Vol.2, No.1, Winter 1990, 0899-1499/90/0201-0004

[8] M. WIDMER & A. HERTZ, A New Approach for Solving the Flowshop Sequencing Problem, to appear in Euro-pean Journal of Operational Research

[9] BLAZEWICZ, J., et al. (2008). Metaheuristic approaches for the two-machine flow-shop problem with weighted late work criterion and common due date. Computers & Operations Research, 35 (2), pp. 574-599.

[10] S.M. Johnson, Optimal two- and three-stage production schedules with setup times included, Naval Res. Log. Quart. I(1954)61-68.

[11] Alippi, C., Galbusera A., & Stellini M. (2003). An Application-Level Synthesis Methodology for Multidimensional Embedded Processing Systems, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 22, No. 11, 0278-0070

[12] E. Cogotti, A. Fanni, and F. Pilo, "A comparison of optimization techniques for Loney's solenoids design: An alternative Tabu Search approach," IEEE Trans. Magn., vol. 36, no. 4, pp. 1153–1157, Jul. 2000.

[13] Emmert, J., M.; Lodha, S. & Bhatia, D., K. (2003). On Using Tabu Search for Design Automation of VLSI Systems, Journal of Heuristics, 9, 2003, 75–90

[14] Rajan, C.C.A., Mohan M.R. & Manivannan, K. (2003). Neural-based tabu search method for solving unit commitment problem, IEE Proceedings Online, Vol. 150, No. 4, July 2003,469-475

[15] Blazewicz, J.; Formanowicz, P.; Kasprzak, M.; Markiewicz, W., T. & Weßglarz, J. Tabu Search for DNA Sequencing with False Negatives and False Positives, European Journal of Operational Research, 125, 2000, 257-265, 0377-2217

[16] Corberan, A.; Marti, R. & Romero, A. (1998). Heuristics for the Mixed Rural Postman Problem, Computers & Operations Research, 27, 2000, 183-203, 0305-0548

[17] Ahr, D. & Reinelt G. (2005). A tabu search algorithm for the min–max k-Chinese postman problem, Computers and Operations Research, 33 (7 December 2005), 3403-3422, 0305-0548.

13.0 Appendices:

13.1 Appendix A: Terminology

Sample: A biological fluid (typically liquid) such as serum, blood, saliva, etc. on which the immunoassay test is performed.

Test: A procedure involving multiple Jobs

Operation: Tasks that are performed on samples in a microplate.

Job: Contains number of operations to be performed on a particular machine

Machine: The devices that are used to perform the operations on microplate for specific amount of time. Ex. Washer, Shaker, Incubator, etc…

Schedule: Sequence of jobs and their respective start times.

Start Time: The time at which the first operation of a job starts.

End Time: The time at which the last operation of a job ends.

Makespan: Difference between the last jobs end time and the first jobs start time in a schedule.

13.2 Appendix B: Mathematical representation

In order to understand the job shop problem with no-wait constraint effectively, the problem represented using mathematical notations. Mathematical notations give the better understanding of the problem.

$n \in Z^+$	Number of jobs.
$u \in Z^+$	Number of machines.
$v \in Z^+$	Number of machine types.
$J= \{J_i \mid i \in [1, n]\}$	Set of jobs.
$b_i \in Z^+$	Number of operations in job J_i.
$M= \{M_r \mid r \in [1, v]\}$	Set of machine types.
$m= \{m_x \mid x \in [1,u]$ and type$(m_x) \in M\}$	m is set of machines
$p_{i,k} \in Z^+$ where $k \in [1, b_i]$ and $i \in [1, n]$	$p_{i,k}$ is the processing time of k^{th} operation of job J_i.
$\Psi(i, k) \in [1,v]$ where $k \leq b_i$ $\Psi:(Z^+, Z^+) \longrightarrow Z^+$	$\Psi(i, k)$ is the machine type index of the k^{th} operation of job J_i. Ψ is a mapping function which takes two integer values i, k and returns an integer, which is an index for the machine type.

$o_{i,k} = (M_{\psi(i,k)}, p_{i,k})$	$o_{i,k}$ is the k^{th} operation of job J_i.
$J_i = seq\{ o_{i,1}, o_{i,2} \ldots\ldots o_{i,b_i} \}$	Sequence of operations of job J_i.
$A_{i,k} = \sum_{j=1}^{k} p_{i,j}$ where $k \leq b_i$	Cumulative processing time of job J_i upto the k^{th} operation
$d_i = A_{i,b_i} = \sum_{j=1}^{b_i} p_{i,j}$	Total processing time of job J_i
$S_i \in Z^+$	Start time of job J_i in a schedule.
$S = \{ s_i \mid i \in [1,n]\}$	Schedule S for problem is a set of start times of jobs satisfying the constraints.
$e_i = s_i + d_i$	End time of the job J_i.
$s_{min} = \min \{s_i \mid i \in [1,n] \}$	Start time of the first scheduled job.
$e_{max} = \max \{e_i \mid i \in [1,n] \}$	completion time of the last scheduled job

Schedule:

A schedule S for Job shop scheduling problem is a set of start times of jobs satisfying the constraints defined below.

$$S = \{s_i \mid i \in [1,n]\}$$

Operation $o_{i,k}$ is assigned a machine m_x of type $M_{\psi(i,k)}$ at time $t = s_i + P_{i,k-1}$

Makespan:

Makespan is the difference between the end time of the last job and start time of first scheduled job.

Makespan = $e_{max} - s_{min}$.

Assumptions:

- The time taken to move a job from one machine to other machine is considered to be negligible.

Constraints

$st_{i,j} \in Z^+$	Start time of j^{th} operation of job J_i.
$et_{i,j} \in Z^+$	End time of j^{th} operation of job J_i.
$et_{i,j} = st_{i,j+1}$ where $i \in [1,n]$ and $j \in [1, b_i]$	**No-wait constraint**. End time of the operation in a job should be the start time of its consecutive operation.
$E_{\beta,\Upsilon} = \{(g,h)\}$ where $\beta,\Upsilon \in [1, n]$ $g \in [1,b_\beta]$, $h \in [1,b_\Upsilon]$	Set of pairs of indices of operations $o_{\beta,g}$ and $o_{\Upsilon,h}$ which need same type of machine

such that the below conditions are satisfied $s_\Upsilon - s_\beta \geq A_{\beta,g} - A_{\Upsilon,h-1}$ or $s_\beta - s_\Upsilon \geq A_{\Upsilon,h} - A_{\beta,g-1}$ $\forall \{g,l\} \in E_{\beta,\Upsilon}$, $\beta<\Upsilon$	. No machine processes two operations atthe same time.
$A_{i,j} \leq A_{i,k-1} \forall k > j$ and k, $j \in [1,b_i]$	No job should be processed by two machines at the same time.

13.3 Appendix C: Complete Local Search with Limited Memory (CLLM):

CLLM starts with a given sequence as an initial solution and puts it into NEWGEN, NEGIHBOR and DEAD are empty.

(1) Generate neighbors for all solutions in the NEWGEN and put them in the NEIGHBOR if NEGIHBOR is not full. By using pair-wise exchange and 1-insertion on every solution of NEWGEN neighbors can be generated. The set of neighbors for a solution π generated by the above procedure is denoted as GN(π).

(2) Empty NEWGEN

(3) Check solutions of NEIGHBOR belonging to DEAD or not. The solutions belonging to DEAD are called good neighbors and others are called not good neighbors.

(4) Among all good neighbors those having makespan less than the certain threshold value are transported to NEWGEN for the next iteration and the others are kept in DEAD.

C_{best} * (ĩ + 1) is set as the threshold value

C_{best} = is the best-so- for makespan

I_{temp} = is the number of iterations without improvement

N_{max} = is the maximum number of consecutive iterations without improvement

Boolean flag is set to false at first iteration and it is set to true once improvement has been improved.

(5) Empty the neighbor, after performing successive iterations R_{max} if there is no improvement then the current instance is reversed.

(6) Maximum number of iterations the algorithm can perform is I_{max}. I_{add} constant is added to it once there is improvement $I_{max} \leftarrow I_{max} + I_{add}$ and are flag is set to true.

This iteration process is repeated until I_{max} iterations have been performed no improvement has been made for N_{max} successive iterations or NEWGEN is empty.