IT 3708: Project 4

Solving Multi-Objective Job Shop Scheduling Problem (JSSP) Using Bio-Inspired Algorithms

Lab Goals

- Implement three recent bio-inspired algorithms to solve multi-objective job shop scheduling problem (JSSP).
- Compare the performance of your implemented algorithms on several benchmark problems.

Groups Allowed? Yes. For this project you may work alone or in groups of two.

Deadline: May 04, 2017 (Thursday) at 07: 59 AM.

Assignment Details

The objective of scheduling is to efficiently allocate shared resources (machines, people etc) over time to competing activities (jobs, tasks, etc.) such that a certain number of goals can be economically achieved and the given constraints can be satisfied. The solutions that satisfy these constraints are called feasible schedule. In general, the construction of a schedule is an optimization problem of arranging time, space, and (often limited) resources simultaneously.

Among various types of scheduling problems, the shop scheduling is one of the most challenging one. It can be classified into four main categories: (i) single-machine scheduling, (ii) flow-shop scheduling, (iii) job-shop scheduling, and (iv) open-shop scheduling. Among these, the job-shop scheduling problem (JSSP) perhaps receives the most attention by a plethora of different approaches. The JSSP is a hard-combinatorial optimization problem, which is not only *NP*-hard but also one of the worst members in that class. In this project, **you need to solve the JSSP using three bio-inspired algorithms by optimizing two objectives simultaneously**.

A JSSP involves processing of the jobs on several machines without any 'series' routing structure. The n x m JSSP can be described by a set of n jobs $\{J_j\}_{1\le j\le n}$ which is to be processed on a set of m machines $\{M_k\}_{1\le k\le m}$. Each job $j \in J$ must be processed by every machine $k \in M$. Each job has a *technological sequence* of machines to be processed. The processing of job J_j on machine M_k is called the *operation* O_{jk}. Operation O_{jk} can be processed by only one machine k at a time; and O_{jk} requires an exclusive use of machine M_k for an uninterrupted duration t_{jk}, its *processing time*. Each operation, which has started, runs to completion; and each machine performs operations one after another. A schedule is a set of completion times for each operation $\{C_{jk}\}_{1\le j\le k, 1\le k\le m}$ that satisfies given constraints. **The challenge here is to determine the optimum sequence** in which the jobs should be processed to optimize one or more performance measure, such as the makespan (the maximum completion time of all jobs), the mean flow time (the average response of the schedule to the individual demands of jobs for service), or the total tardiness of jobs (the lateness of any job measures the conformity of the schedule to that job's committed date), etc.

Table 1 shows an example of *6*x*6* job-shop scheduling benchmark problem. In this example, the Job-1 is processed by Machine-3 for 1 time unit, and then it is processed by Machine-1 for 3 time units, and so forth.

Table 5.1: A *6*x*6* job-shop scheduling benchmark problem

Job-*n*	(*k*,*t*)	(*k*,*t*)	(*k*,*t*)	(*k*,*t*)	(*k*,*t*)	(*k*,*t*)
Job-1	3,1	1,3	2,6	4,7	6,3	5,6
Job-2	2,8	3,5	5,10	6,10	1,10	4,4
Job-3	3,5	4,4	6,8	1,9	2,1	5,7
Job-4	2,5	1,5	3,5	4,3	5,8	6,9
Job-5	3,9	2,3	5,5	6,4	1,3	4,1
Job-6	2,3	4,3	6,9	1,10	5,4	3,1

The typical output of any JSSP is a **Gantt-Chart presenting the schedule** (allocation of shared resources over time to competing activities) optimizing one objective. Fig. 1 presents an optimal schedule of the JSSP (presented in Table 1) based on the minimum *makespan* objective (optimizing makespan).

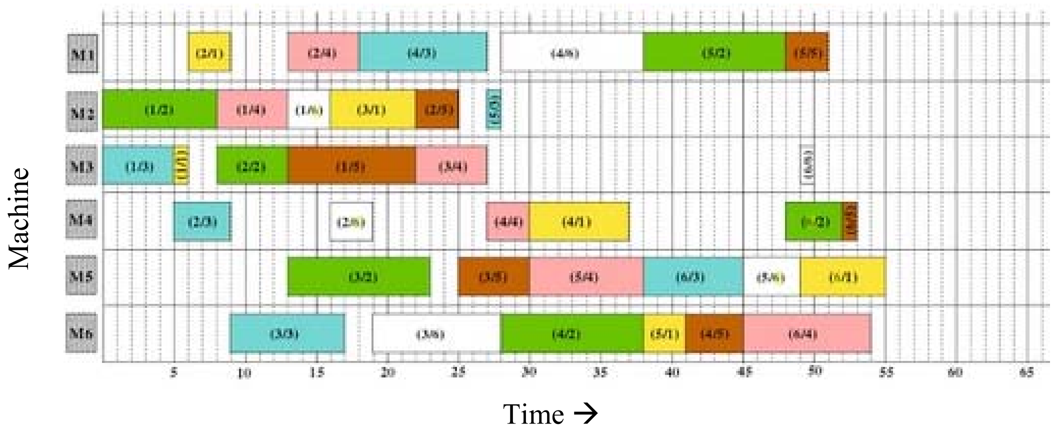

Fig. 1: Gantt-Chart of the optimal schedule with makespan objective.

In this project, you will implement **three bio-inspired algorithms to solve the JSSP by optimizing two objectives simultaneously**. The algorithms are:

1. Particle Swarm Optimization (PSO),
2. Ant Colony Optimization (ACO), and
3. Bees Algorithm (BA).

Problem Formulation:

Solution (individual/chromosome) representation is a key issue in designing efficient bio-inspired algorithms for JSSPs, because different methods for representing parameters in scheduling solutions create different search spaces and different difficulties for variation/algorithmic operators. The individual representation can be classified into two major approaches: (i) direct representation, and (ii) indirect representation. In indirect representation, the individual encodes a sequence of preferences. These decision preferences can be heuristic rules or simple ordering of jobs in a machine. After that a ***schedule-builder*** is required to decode the chromosome into a schedule. In a direct representation, the schedule itself is directly encoded onto the individual/solution/chromosome and thereby eliminating the need for a complex schedule-builder. At the same time, applying simple variation/algorithmic operators on direct representation string often results in infeasible schedule solutions. For this reason, domain-specific variation/algorithmic operators are required.

As mentioned earlier, in indirect representation, the individual/solution/chromosome contains an encoded schedule and a scheduler-builder is used to transform the individual/solution/chromosome into a feasible schedule. The schedule-builder is a module of the evaluation procedure and should be chosen with respect to the optimization objective. It is well-established that the minimization of *makespan* plays the major role in converting the solution-representation into feasible and optimal schedule. There are several schedule-builder algorithms exist in literature. For example, one of the efficient approaches is the Giffler & Thompson algorithm [1, 2]. You are free to choose your own schedule-builder, *off-course if you use indirect representation*. Even, you can design our own schedule-builder. Be noted that if you use schedule-builder, **always prefer makespan to construct the schedule as the primary (may be the only) objective**.

In this project, you need to simultaneously optimize the following two objectives for the JSSP.

(i) **Makespan**: The maximum completion time among all jobs. It is subject to minimization and can be defined as:

$$f_1 = \max_{j=1}^{n} C_j$$, where C_j is the completion time of job j.

(ii) **Total Tardiness Time**: The sum of tardiness (lateness) time for all jobs. The tardiness of any job is the lateness of that job in completing all of its operations from the predicted due time for this job. It is also subject to minimization and can be defined as:

$$f_2 = \sum_{j=1}^{n} T_j \ ;$$

$$T_j = \max(C_j - D_j, 0)$$

where T_j is the tardiness of job j, C_j is the actual completion time for job j, and D_j is the predicted due time for job j.

Things To Do

The 30 points total for this project is 30 of the 100 points available for this course. The 30 points will be distributed on two parts: (*i*) demo and (*ii*) report. **The demo can give you a maximum of 27 points and the report can give you a maximum of 03 points**.

To test your code, we uploaded 06 benchmark JSSP instances with acceptable values for both objectives. The description of input files and the predicted due times (or how to calculate the predicted due times) are also included in the test data.

Along with presenting the optimal values and Pareto-front, you need to produce two types of Gantt-Charts: (i) targeting makespan (as in Fig. 1), and (ii) targeting tardiness of jobs. Be noted that **both may produce the same Gantt-Chart**.

(a) Demo (27p):

There will be a demo session where you will show us the running code and we will verify that it works. If you work in a group, **both group members need to attend the demo session together**.

In the demo session, you need to describe how you designed and implemented your algorithms. Also, you need to test your code by solving 02 (two) JSSP instances which will be supplied during the demo. For the demo session, we will supply the due times for all jobs, you need not to calculate like the test data. **Your implementation should have the option to accept our supplied due times for all jobs and calculate the objective value based on those**.

Note that, **you must run your code and show us all the requirements within 30 (thirty) minutes**. The point distribution for the demo is as follows:

(1) For each of the three algorithms, you need to run 02 (two) JSSP instances. Each algorithm can give you a maximum of 09 points. (27p = 9 x 3)

- For each algorithm, each JSSP instance can give you a maximum of 4.5 points. (9p = 4.5 x 2)
    - For Makespan (the best makespan from the Pareto-front) (2p)
        - If your value is within 10% of acceptable value, you will get full points.
        - If your value is within 20% of acceptable value, you will get 1.5 points.
        - If your value is within 30% of acceptable value, you will get 1 points.
        - Otherwise, you will get 0.
    - For Total Tardiness (the best total tardiness from the Pareto-front) (2p)
        - If your value is within 10% of acceptable value, you will get full points.
        - If your value is within 20% of acceptable value, you will get 1.5 points.
        - If your value is within 30% of acceptable value, you will get 1 points.
        - Otherwise, you will get 0.
    - Show the final Pareto-front and two Gantt-Charts: (i) targeting the best makespan, & (ii) targeting the best tardiness of jobs. (0.5p)

(b) Report (03p):

You should write a report answering the points below. Your report must not exceed 04 (Four) pages in total. **Over length reports will result in points being deducted from your final score**. Print on both sides of the sheet, preferably. **Bring a hard copy of your report to the demo session**. If you work in a group, you only need to submit one single report on behalf of the group.

1. Representation of solutions (individual/chromosome) for each of the three algorithms. **Using figure(s) for each representation is a must**. (0.5p)
2. For each of the three algorithms, how does you build the schedule from respective solution? (1p)
3. For all three algorithms, draw the final Pareto-front and two Gantt-Charts: (i) targeting the best makespan, & (ii) targeting the best tardiness of jobs; **using the test problem-3**. (1.5p)

Delivery

You should deliver your report + a zip file of your code on *itslearning*. The submission system will be closed at 07:59 AM on May 04, 2017.

If you work in a group, you only need to deliver once on *itslearning* (but both group members must the registered as part of the submission on *itslearning*!). **Both group members need to attend the demo session**.

You must attend the demo on the scheduled demo date which has been declared on *itslearning***.** Since the demo dates were declared at the beginning of the semester**, no early or late demo will be entertained** except for extreme emergency like sickness with medical certificate, job interview, attending funeral or the like**.** Traveling or holidays will not be considered as emergency situation.

References:

[1] B. Giffler, G. Thompson, “Algorithms for solving production scheduling problems,” *Operations Research*, vol. 8, no 4, 1960, pp. 487-503.

[2] R. Varela, D. Serrano, and M. Sierra, “New codification schemas for scheduling with genetic algorithms,” in: J. Mira, J. R. Álvarez, Ed., *Lecture Notes in Computer Sciences*, vol. 3562, Springer, 2005, pp. 11–20.