IT 3708: Project 3

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

Lab Goals

- Implement two recent bio-inspired algorithms to solve the 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 group of two. Note that, even though, you work in a group, **you must write the report individually without any collaboration with your group member**. Also, **you must attend the demo individually**.

Deadline: May 09 (Wednesday), 2018 at 08: 00 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 number of goals can be achieved and the given constraints can be satisfied. The solutions that satisfy these constraints are called feasible schedules. 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 scheduling problems. It can be classified into four main categories: (i) single-machine scheduling, (ii) flow-shop scheduling, (iii) job-shop scheduling, (iv) open-shop scheduling. In this project, **you will focus on solving the job shop scheduling problem (JSSP)**. The JSSP is a hard-combinatorial optimization problem, which is not only *NP*-hard but also one of the worst members in that class.

A JSSP involves processing of jobs on several machines without any 'series' routing structure. The *n x m* JSSP can be described by a set $\{J_j\}_{1\le j\le n}$ of *n* jobs which is to be processed on a set $\{M_k\}_{1\le k\le m}$ of *m* machines. The challenge here is to determine the ***optimum sequence*** $\{C_{jk}\}_{1\le j\le n,1\le k\le m}$ in which the jobs should be processed to optimize one or more objective(s), such as the makespan, the mean flow time or the total tardiness of jobs. Each job has a technological sequence of machines to be processed. The JSSP can be characterized as follows:

- each job $j \in J$ must be processed by every machine $k \in M$;
- the processing of job J_j on machine M_k is called the operation O_{jk};
- operation O_{jk} requires the exclusive use of machine M_k for an uninterrupted duration t_{jk}, its processing time;
- each job consists of an operating sequence of operations (technological sequence of each job);
- O_{jk} can be processed by only one machine at a time;
- each operation, which has started, runs to completion;
- each machine performs operations one after another.

Table 1 shows a *6x6* job-shop scheduling benchmark problem. In this example, the Job-1 is processed by Machine-3 for 1-time unit, then by Machine-1 for 3-time units, and so forth.

Table 1: A 6x6 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 a JSSP is a Gantt-Chart presenting the **schedule** (allocation of shared resources over time to competing activities) optimizing one or several objectives. Fig. 1 presents an optimal schedule of the JSSP (Table 1) based on the minimum *makespan* objective.

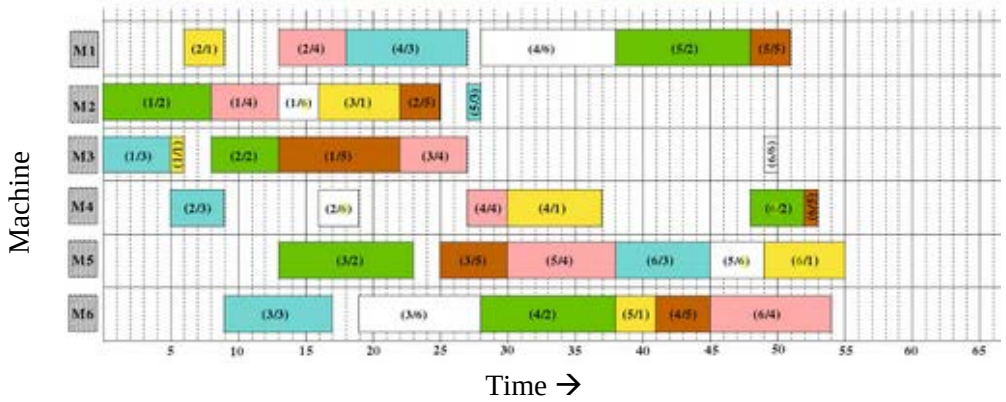

Fig. 1: Gantt-Chart of the optimal schedule with makespan objective.

In this project, you will implement **two bio-inspired algorithms to solve the JSSP by optimizing makespan**. Makespan is the maximum completion time among all jobs. It is defined as:

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

The algorithms are:

1. Ant Colony Optimization (ACO).
2. Bees Algorithm (BA).

For both of two algorithms, in addition to the basic algorithms, several variations have been proposed in the literature. **You can use the basic algorithms or any of these variations**.

Problem Formulation:

Solution (individual/chromosome) representation is a key issue in designing efficient bio-inspired algorithms for JSSPs because choosing appropriate representation will affect all the other steps. In addition, different representation methods create different search spaces and different difficulties for variation/algorithmic operators. For several representation alternatives for the JSSP, you can look at [1].

Solution 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. The schedule builder is a module of the evaluation procedure and should be chosen with respect to the performance-measure of optimization. It is well-established that the minimization of *makespan* plays the major role in converting the chromosomes into feasible schedule. There are several schedule-builder algorithms exist in literature [2]. You can use your preferred choice, if you use indirect representation. Even, you can design our own schedule-builder.

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.

Since the traditional Bees algorithm is intended for continuous function optimization, it cannot be directly applied to scheduling problems with inherent discrete nature. As for the JSSP, each solution is naturally a permutation of integers. To address this issue, two kinds of approaches can be identified in the literature. **Use as per your requirement or propose a new one**.

(i) A transformation scheme is established to convert permutations into real numbers and vice versa [3].

(ii) The search operators in BA are modified to suit the permutation representation [4].

Things To Do

The 25 points total for this project is 25 of the 100 points available for this course. The 25 points will be distributed on two parts: (*i*) demo and (*ii*) report. **The demo can give you a maximum of 18 points and the report can give you a maximum of 07 points**.

In this project, you need to ***minimize makespan*** for the JSSP using BA and ACO. Along with presenting the optimal values for makespan, **you need to plot the Gantt-Chart targeting makespan** (as in Fig. 1). To test your code, we uploaded 06 benchmark JSSP instances with acceptable values. The description of input files is also included in the test data.

(a) Demo (18p):

There will be a demo session where you will show us the running code and we will verify that it works. Even though you work in a group, **you must attend the demo individually on the scheduled demo date**. In the demo session, you need to describe how you designed and implemented your algorithms. Also, you have to test your code by running 03 (three) JSSP instances that you will be supplied during the demo. Note that, **you must run your code and show us all the requirements (including the explanation) within 30 (thirty) minutes**. The point distribution for the demo is as follows:

(1) For each of the two algorithms, you need to run 03 (three) JSSP instances. For each of the JSSP instance, you must present the obtained schedule using a Gantt-Chart that will clearly present the assignment of each operation to each machine as well as the processing time, for example Fig. 1. (15p = 2 x 2.5 x 3)

* For each algorithm, each JSSP instance can give you a maximum of 2.5 Points.
    - 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.

* **You can only get a maximum of 1 point per JSSP instance for Demo Section 1** (provided that you can achieve the makespan within the desired limit)**, if you do not present the appropriate Gantt-Chart.** Finding any makespan value lower than 30% of acceptable value will not give any point for that instance, even though you present a Gantt-Chart.

(2) Explanation (3p)

* Based on any of the test instances for both algorithms, you need to be ready to explain the reason behind the differences in achieved makespan. Also, you need to describe the effect of transforming the continuous nature of BA in a discrete nature problem like JSSP, in comparison to the working procedure of ACO. You need to explain based on your implementation and achieved results.

(b) Report (07p):

You should write a report answering the points below. Your report must not exceed 02 (Two) 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 must write the report individually without any collaboration with your group member as well as you need to submit individually.

1. As mentioned in this project description, additional strategy is required for BA for solving the JSSP which is discrete in nature. There are different strategies for this purpose. Your implemented BA have a certain strategy. In comparison to any other existing strategy, why do you think your implemented strategy is better? If not, why not? (2.5p)
2. What differences have you observed between ACO and BA in handling exploration and exploitation while solving the JSSP? You need to explain your answer based on your implementation and achieved results. (2.5p)
3. As mentioned earlier, there are several variants of ACO. Which variants of ACO have you used in your implementation? In comparison to any one of the other existing variants of ACO, why do you think your implemented variant is better? If not, why not? (2p)

Delivery

You should deliver your report + a zip file of your code on *BlackBoard*. The submission system will be closed at 08:00 AM on May 09, 2018. **If you worked in a group, both of you should submit individually,** mentioning your group member's name in the report (precisely as your group member). **You must attend the demo individually on the scheduled demo date**, which have been already declared on *BlackBoard*. **No early or late demo will be entertained except for extreme emergency**.

References:

[1] R. Cheng, M. Gen, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms – I: Representation," *Computers and Industrial Engineering,* vol. 30, no. 4, pp. 983-997, 1996.

[2] R. Varela, D. Serrano, and M. Sierra, "New codification schemas for scheduling with genetic algorithms," *Lecture Notes in Computer Sciences*, vol. 3562, pp. 11–20, 2005.

[3] Y.J. Shi, F.Z. Qu, W. Chen, and B. Li, "An artificial bee colony with random key for resource-constrained project scheduling," *Lecture Notes in Computer Science*, vol. 6329, pp. 148–157, 2010.

[4] Q.-K. Pan, M.F. Tasgetiren, P.N. Suganthan, and T.J. Chua, 2011. "A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem," *Information Sciences*, vol. 181, no. 12, pp. 2455–2468, 2011.

NB:

1. Every question regarding this project will **only be handled** through the Slack channel.