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 groups of two. However, note that the workload assigned to this project is for groups of two. If you choose to work alone, you may use more hours on your implementation, but the performance requirements are a bit easier. **Each student must attend the demo day.**

Deadline: April 23rd (Thursday), 2020 at 08:00 AM.

Assignment Details

The object of scheduling is to efficiently allocate shared resources (machines, people etc.) over time to competing activities (jobs, tasks etc.) such that the goal(s) can be achieved while satisfying the given constraints. 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 and (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 of that class.

A JSSP involves processing of jobs – each consisting of a sequence of operations, on each of several machines. Further, each operation has to be processed on a particular machine. The goal is thus to ensure that each operation of all jobs is allocated to the appropriate machine at a time in the schedule that ensures the most efficient use of the resources available whilst ensuring that all operations of all jobs are executed in the correct order.

The $n \times m$ JSSP can be described by a set $\{J_j\}_{1 \le j \le n}$ of n jobs which are to be processed on a set $\{M_k\}_{1 \le k \le m}$ of m machines. The challenge is to determine the ***optimal sequence*** $\{C_{jk}\}_{1 \le j \le n, 1 \le k \le m}$ in which the jobs (and operations within) should be processed by the machines in order to optimize the objective(s). Typical objectives for the JSSP include makespan, the mean flow time and the total tardiness of jobs.

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 $j \in J$ on machine $k \in M$ is called the operation O_{jk};
- each job consists of a strict sequence of operations;
- operation O_{jk} requires the exclusive use of machine M_k for an uninterrupted duration t_{jk}, its processing time;
- each operation, once started, runs to completion;

- each machine performs operations one after another.

Table 1 provides an example of a *6*x*6* job-shop scheduling benchmark problem. Each row in the table represents an ordered sequence of operations for a job. **Note that this is only the formulation of the problem, and gives the order of which the operations in each job need to be performed when creating the schedule.** Each operation is described by (*k*,*t*), which means that it is processed by machine *k* for *t* time units and the row of the operation defines which job *n* the operation is a part of. For example, (3,1), which is the first operation in the row of job 1 in the table, means that machine 3 processes job 1 for 1 time unit. The second operation of job 1, is executed on Machine 1 for 3 time units, the third operation of job 1, is executed on Machine 2 for 6 time units, and so forth.

Table 1: A *6*x*6* job-shop scheduling benchmark problem

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 an algorithm for solving the JSSP is a Gantt-Chart presenting the **schedule** (allocation of shared resources over time to competing activities). By definition, a **schedule** is called **active** if one cannot change the order of processing of operations on the machines and have at least one operation finish earlier and no operation finish later. The schedule is the solution achieved through optimizing one or several objectives. Figure 1 presents an optimal schedule of the JSSP for the problem presented in in Table 1. **Note that for the Gantt-Chart in Figure 1, the notation is different from Table 1 as this is the optimal schedule and not the ordering of operations in each job.** In the Gantt-Chart, each row gives the operations performed by one machine and each colored box represents one operation. The length of each colored box on the x-axis, gives the number of time units the operation needs in order to be processed on that machine. The numbers on each operation/colored box, represent the job the operation is a part of and which number the operation has in the ordered sequence of that job. **As for example, (2/1), which is found on the yellow box upper left in the schedule in Figure 1, means that this operation is the second operation of job 1.** Further, the solution given in the Gantt-Chart represents a minimum makespan objective, which is formally defined below.

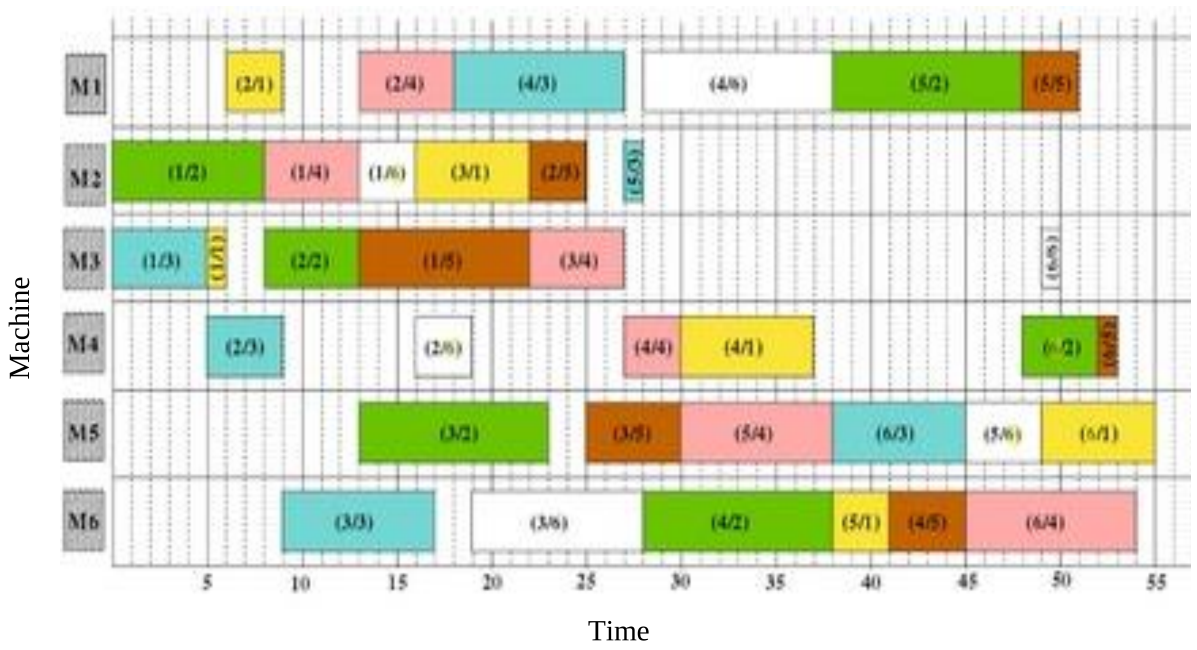

Fig. 1: Gantt-Chart of the optimal schedule with makespan objective.

In this project, you will implement **two** bio-inspired algorithms to solve the JSSP: **Particle Swarm Optimization (PSO)** and **Ant Colony Optimization (ACO)** by optimizing – that is, **minimizing – makespan**. Makespan is the maximum completion time among all jobs and is defined as:

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

In Figure 1, the last job that is finished is job 1, which can be seen in the Gantt-Chart from the yellow box numbered (6/1), as this box ends at the latest time on the x-axis, at time 55. **Hence, the makespan for the schedule in Figure 1 is 55 time units.**

For both PSO and ACO, 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**. Note that for PSO, many hybrid algorithms have been developed that seek to achieve more optimal/effective solutions for more challenging instances of JSSP. However, there is no requirement for such hybrid algorithms for the instances of JSSP that you will meet in this project.

Problem Formulation:

Representation is a key issue in designing efficient bio-inspired algorithms for JSSPs. The chosen representation will not only affect all other steps but create different search spaces and can provide additional challenges for variation/algorithmic operators. Representation can be classified into two major approaches: (i) direct representation and (ii) indirect representation.

In an direct representation, the schedule itself is directly encoded into the individual (genotype), thereby eliminating the need for a simple/complex schedule builder. However, applying simple variation/algorithmic operators on direct representations often results in infeasible schedules. For this reason, domain-specific variation/algorithmic operators are required. However, an alternative is to select an indirect representation.

In indirect representation, a **schedule builder** is required to decode the chromosome into a schedule. The schedule builder is a module of the evaluation procedure whereby the genotype is converted into the phenotype before evaluation of the solution. The design/selection of an appropriate schedule builder should reflect the objective sought i.e. minimization of makespan. There are several schedule-builder algorithms mentioned in the literature. If you use an indirect representation, you can either apply your preferred schedule-builder from the literature or make your own.

The selection of an appropriate representation is critical to both PSO and ACO. Scheduling problems are inherently discrete in nature. PSO is traditionally applied to continuous optimization problems but may also be applied to such discrete optimization problems. The articles selected for this project [1,2,3,4] provide different suitable representations for job scheduling as well as the authors' reasoning for their choice of representation. You will also find discussions about initialization and feasible/infeasible solutions in some of the articles. Note that some of the articles may not consider JSSP itself or minimization of makespan but provide relevant useful information for JSSP. Other articles of relevance may be found in the literature. It should be noted that some more recent articles about PSO for shop scheduling focus on hybrid algorithms, i.e., PSO together with another algorithm such as simulated annealing. However, such hybrid algorithms are not necessary to achieve strong results for the instances of JSSP you will encounter in this project.

Note that the terminology applied herein can be confusing with regards to indirect developmental mappings. Such mappings are complex mappings with no one-to-one mapping from the genotype to the phenotype. In the context herein, in the indirect mappings described, there is a one-to-one mapping from the genotype to the phenotype. Indirect refers to the need for a mapping (schedule builder), although a relatively simple mapping compared to that of a developmental mapping.

Things To Do

The 15 points total for this project are 15 of the 100 points available for this course. The 15 points will be distributed in two parts: (i) code demo (12 points) and (ii) questions during demo (3 points).

In this project, you need to **minimize makespan** for the JSSP using PSO 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 7 benchmark JSSP instances with corresponding acceptable values for makespan. The description of the input files is included in the test data.

Demo (15p):

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 are required to test your code by running 3 (three) JSSP instances that you will be supplied during the demo. Note that **you must run your code and fulfill all the requirements (including the explanations) within 30 (thirty) minutes**. The point distribution for the demo is as follows:

(1) Code Demo, in groups (12p = 2 x 2 x 3)

For **each** of the two algorithms, you need to run 3 (three) JSSP instances. For each of the JSSP instances, 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, see Fig. 1.

- For **each** algorithm, each JSSP instance can give you a maximum of 2 Points.
  - If your value is within 10% (or 12 % if you work alone) of the acceptable value, you will get full points.
  - If your value is within 20% (or 22 % if you work alone) of the acceptable value, you will get 1.5 points.
  - If your value is within 30% (or 32 % if you work alone) of the acceptable value, you will get 1 point.
  - Otherwise, you will get 0 points.
- **You can only get a maximum of 1 point per JSSP instance for Demo part (1) if you do not present the appropriate Gantt-Chart.** Finding any makespan value smaller than 30% (or 32 % if you work alone) of acceptable value will not give any points for that instance, even though you present a Gantt-Chart.

(2) Questions during demo, individually or in groups (3p)

- Describe your implementation and show different parts of the code for both PSO and ACO.
- What differences have you observed between PSO and ACO in handling exploration and exploitation while solving the JSSP? You need to explain your answer based on your implementation and achieved results.

Any questions regarding this assignment will only be handled at the lab hours or on the Blackboard forum.

Delivery

You should deliver a zip file of your code on BlackBoard. The submission system will be closed at 08:00 AM on Thursday April 23th. **Every student must sign up and attend the demo day**. **No early or late demos will be entertained**.

If you submit the project after Easter, please note that there will be **NO Lab Hours or Forum support during the Easter vacation**. Please ensure you use the support available before Easter. There will be a final lab meeting on Thursday April 16th after Easter.

References:

1. Zhixiong Liu, "Investigation of Particle Swarm Optimization for Job Shop Scheduling Problem*" *Third International Conference on Natural Computation* (ICNC 2007)

2. M.F. Tasgetiren, M. Sevkli, Y. Liang and G. Gencyilmaz “Particle Swarm Optimization Algorithm for Single Machine Total Weighted Tardiness Problem”, Proceedings of the 2004 IEEE Congress on Evolutionary Computation, pp 1412-1419, 2004.
3. W. Xia and Z. Wu , “A hybrid particle swarm optimization approach for the job-shop scheduling problem”, International Journal of Advanced Manufacturing Technologies, 29: 360, 2006
4. Zhou, R., Lee, H.P., & Nee, A.Y. (2008). Applying Ant Colony Optimisation (ACO) algorithm to dynamic job shop scheduling problems. *IJMR*, *3*, 301-320.