Project 4

Particle Swarm Optimization (PSO)

Solution representation

Our implementation of PSO is mainly based on Lin et al. 2010[1]. For the initial population, individuals are created with a representation which consists of a one-dimensional list with length *n x m* and values between 0 and 100. Here, *n* is the number of jobs and *m* is the number of machines. As per the random-key encoding scheme from Lin et al. (2010), these numbers are further sorted and given an index relating to their position in the sorted list. This is used to create a permutation with job indexes as seen in the third step of figure 1. Finally, this permutation is transformed into a sequence of operations in relation to the input data. With this representation one is guaranteed that jobs are performed in the correct order. Furthermore, this representation can be classified as an indirect representation, and as such, simple algorithmic operations are easy to apply to create variations in the chromosome.

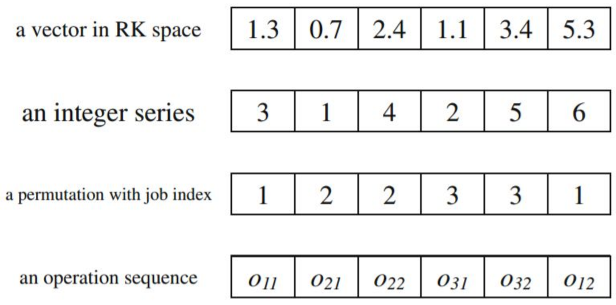

Figure 1: Particle representation.

Schedule builder

With an indirect representation of all of our chromosomes in relation to our algorithms, we are able to use the same schedule builder for all of our algorithms. The schedule builder is simply a greedy algorithm that takes an operational sequence as input, then for each operation it attempts to insert the operation at the first possible time slot for its machine. A list keeps track of when each job has completed its latest task. Another two-dimensional list holds track of when an operation starts as well as the duration of the operation to be performed. Thus, by checking when the operation preceding a given operation in a job is completed, we are able to insert operations in between each other if possible. This gives a feasible solution that also serves as a good initial solution when the input consists of randomized operations.

Ant Colony Optimization(ACO)

Our implementation of the Ant Colony Optimization algorithm is based on previous work published in Zhang et al., 2006[2] and Van der Zwaan & Marques, 1999[3]. The two algorithms presented in these papers have both

[1] Lin, T., Horng, S., Kao, T., Chen, Y., Run, R., & Chen, R. et al. (2010). An efficient job-shop scheduling algorithm based on particle swarm optimization☆. Expert Systems With Applications, 37(3), 2629-2636. http://dx.doi.org/10.1016/j.eswa.2009.08.015

[2] Zhang et Al., 2006, *Implementation of an Ant Colony Optimization technique for job shop scheduling problem, Transactions of the Institute of Measurement and Control 28, 1 (2006) pp. 93/ 108*

[3] Van der Zwaan & Marques, 1999, *Ant Colony Optimisation for Job Shop Scheduling*

implemented an indirect representation of the chromosomes/ants that are used to generate the solution to JSSP problems. We also decided to use indirect representation of the chromosome/ants, so that a schedule builder is required to decode the chromosomes/ants into schedules. We have also chosen a discrete implementation of the ACO, so that the local updates of the pheromone level on edges traversed by ants are conducted after each ant has constructed a feasible solution. The global update of the pheromone levels are conducted after all ants have constructed a feasible solution, so that a global-best-tour can be used to update the pheromone levels on the edges belonging to the global best solution. We have also chosen to represent the JSSP problem as disjunctive graphs G=(V,A,B) in which V = nodes corresponding to operations, A = directed edges between one node/operation and direct following operation on the same job, and B = undirected edges between operations that are not bound by precedence restrictions.

Solution representation

In our implementation, ants are objects with certain attributes. Each ant has two important lists:

- tabuNodes; which keeps track of the order in which operations are selected by the state transition model in each generation
- availableNodes(n); which represents the feasible nodes that can be reached from operation/node n

The representation of the solution is found in the tabuNodes list of each ant as depicted in figure 2.

$O_{i,j}$	$O_{i+a,j+b}$	$O_{i+a,j+c}$	$O_{i,j+b}$	$O_{i+a,j}$	$O_{i+a,j+a}$

Figure 2 - Ant Solution representation

The representation of the solutions in the ACO is an implicit integer sequence of all the operations in the JSSP problem, bound by the precedence rules. The integer values are as attributes in node-objects that represent operations in the graph G. Each operation in the JSSP problem is initially assigned an integer-value based on its' corresponding job- and machine number. The integer values range from 1 to n*m, where n is the number of jobs, and m is the number of machines. Operation 1 can be denoted O_{11} where the first 1 stands for the job number and the latter 1 for the machine number. O_{12} is assigned the integer value 2, O_{13} = 3, O_{14} = 4, etc. The tabuNodes list holds node-objects. The node objects has attributes that holds the job- and the machine number that it belongs to. The node objects also keeps track of the edges (A and B) that goes from the current node to feasible nodes in the graph G. In relation to figure 1. the representation of the ACO chromosomes are immediately represented as an operation sequence, and does not have to undergo transformation before it is fed into the schedule builder. In sum, the chromosome representation is a sequence of operations that have to be decoded in order to create a schedule.

Schedule builder

The schedule builder that is used to decode the ACO chromosome solutions into schedules, is identical to the ones used in PSO and Bee's. The reason for why we are able to use the same schedule builder, is because we have the same chromosome representation, i.e. an operation sequence representation of the chromosomes, that can be fed into the schedule builder. As mentioned, the tabuList of the ant's are fed into the schedule builder to decode the sequence of operations into a schedule. Given the operation number of a node in the sequence, the schedule builder is implemented to calculate which job and which machine the operation number belongs to. In that sense, the schedule builder sort operations based on the machine they are using.

After this sorting is completed, the schedule builder will greedily try to perform the next operation on each machine according to the operation sequence until all operations are completed.

Bee's Algorithm

Our implementation of Bee's Algorithm is mainly based on Pham et al. (2005)[4]. Our Bee's, or our chromosomes use the same representation used in our PSO, and following this it is an indirect representation of a chromosome. Consequently we are also able to use the same schedule builder described above.

Gantt-chart's for problem-3:

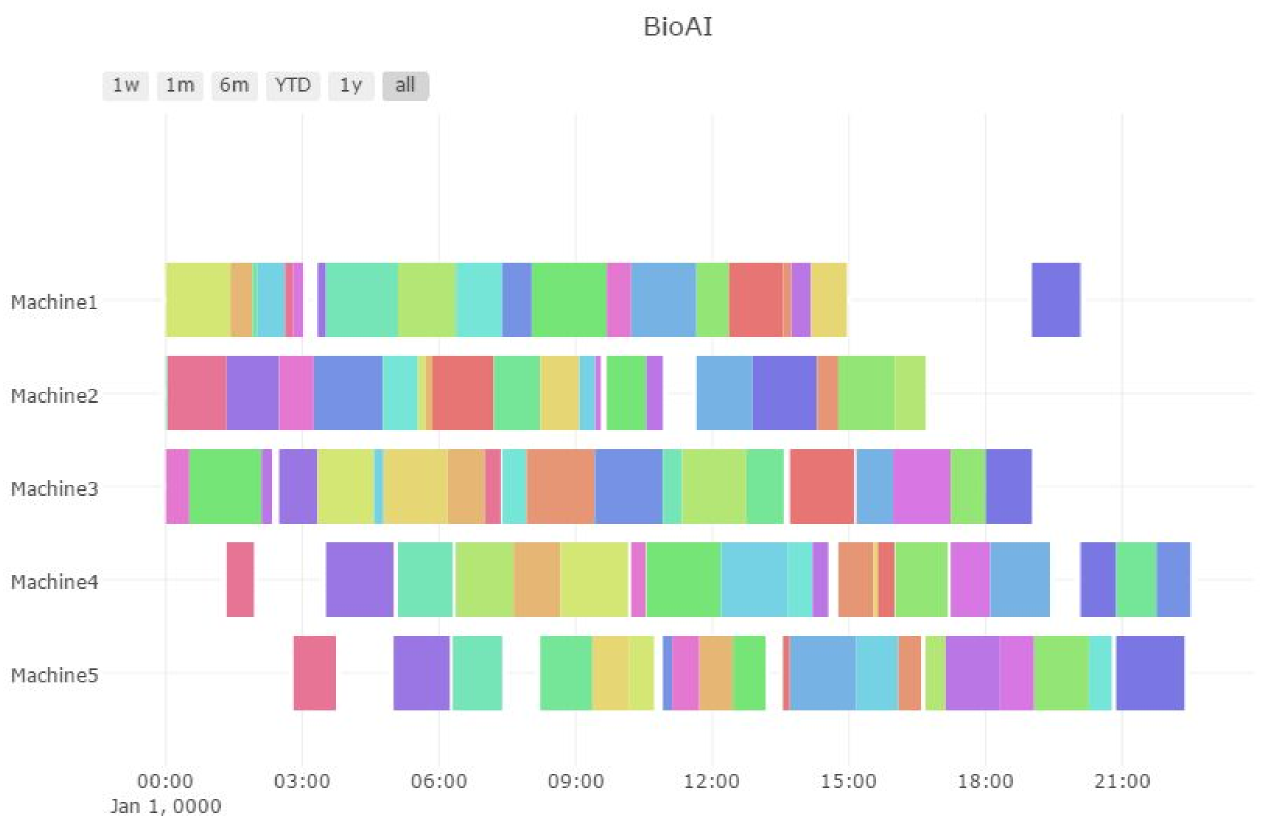

Figure 3: PSO on problem-3

[4] Pham, D., Ghanbarzadeh, A., Koç, E., Otri, S., Rahim, S., & Zaidi, M. (2006). The Bees Algorithm — A Novel Tool for Complex Optimisation Problems. Intelligent Production Machines and Systems, 454-459. doi:10.1016/b978-008045157-2/50081-x

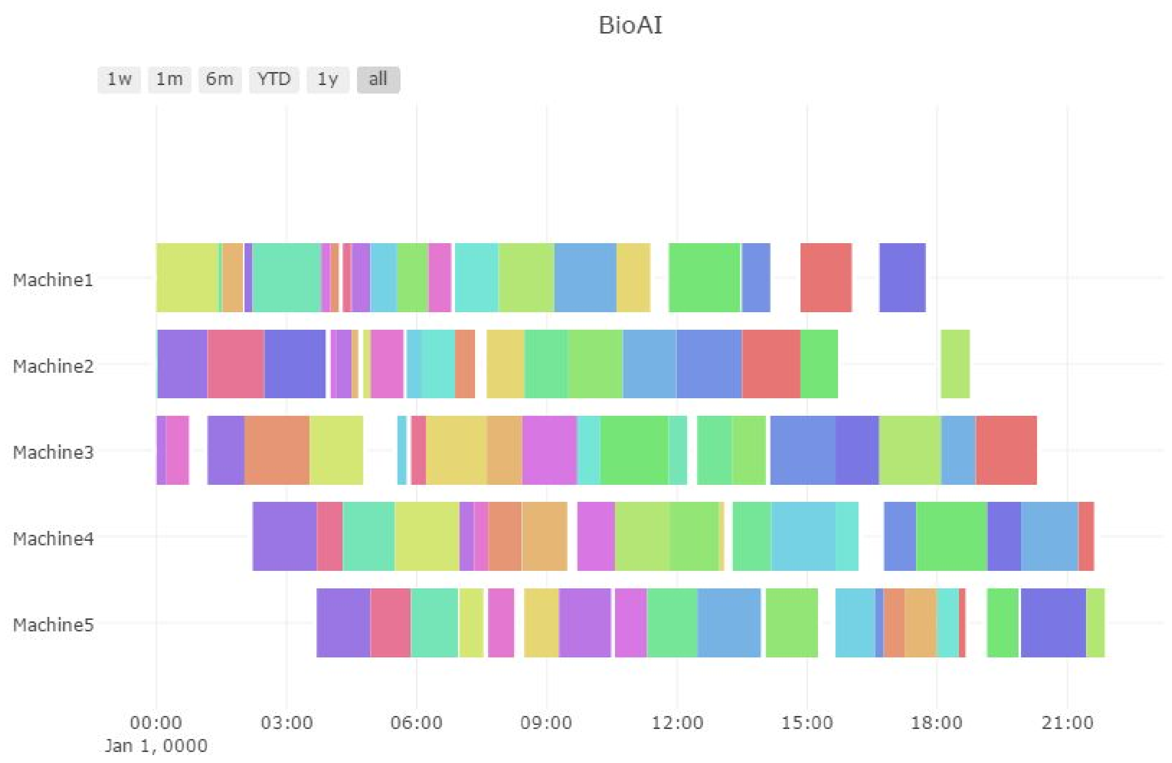

Figure 4: ACO on problem-3

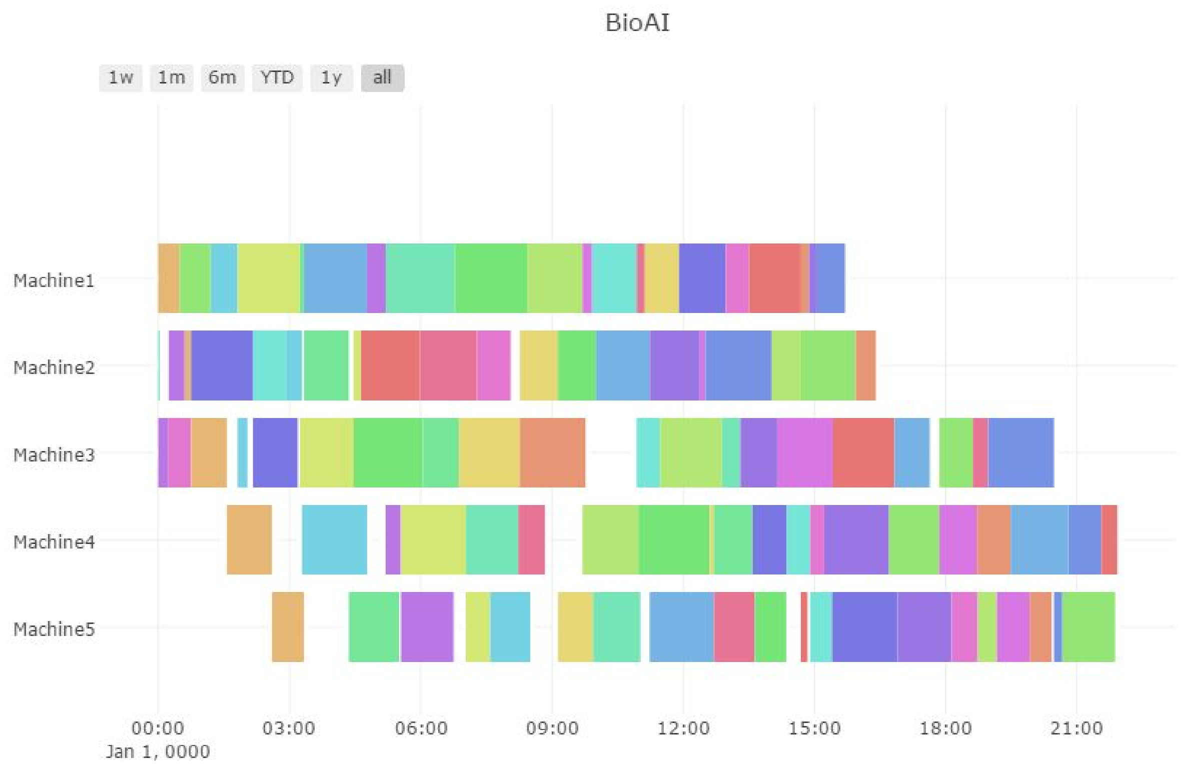

Figure 5: Bee's Algorithm on problem-3