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

Lab rapport IT3708 - Project 3

Sigve Skaugvoll

May 2018

1 JSSP

1.1 BA strategy

The biggest changes in the strategy from a standard BA, is the way local search is implemented. The strategy chosen for the bees is that for each bee's solution, a region or parameter around the solution is also created, we call this the "solutionRegion". The size of the region is determined by how good the solution is. We think it's efficient to try small changes, when we have found a good solution, to try and find a better solution, and if the solution is bad, we want a bigger region, so that we can cover more ground in finding a better solution. Thus the region size is defined by the solutions fitness.
The solution is represented as a string of integers, representing indexes to nodes in the graph. The local search used is used to find a better solution, with nodes up to a predefined neighborhood size. Then the new path is chosen based on a probability using heuristics.
We think that this strategy introduces a chance that if the bee doesn't find a better solution it finds the original solution.

Another solution would be to represent the solution as a string, representing the order job-tasks was selected, i.e "1,2,3,4,1,1,2,3,4" read this as (job,operation) which converts to (1,1), (2,1) (3,1) (4,1) (1,2) (1,3) (2,2) (3,2) (4,2). With this implementation, a local search could be to try different permutations of the string, to try and minimize the makespan. The drawback of this strategy is the the complexity and computationally heavy and use of a custom schedule builder.

1.2 ACO vs BA

When running the two algorithms on task 1, we identified some remarkable differences. The number of nodes in the graph generated, BA is approx. 4.2 times smaller in search space, which corresponds with the time which is approx. 4.6 times faster! The number of nodes grows fast, because we not managed to implement a global graph of only the "best" solutions, both algorithms build a huge global graph, which has all paths explored. This is a huge drawback of our implementation.

Algorithm	#Nodes	Time used	#Iterations	maskespan	acceptable
ACO	1 257 450	2487ms,	100	55	55
BA	293 290	533ms,	100	55	55

Table 1: Comparison from runs

I think what makes the the difference in nodes explored, are that ACO, use stigmergy, which influences the path and solution. If there is a good solution, and a new ant finds an ever better, the colony will most likely not take the better in use, because the pheromones are stronger in the mediocre solution. The bees uses the solution region to improve the solution to get even better solutions, and thus not use stigmergy to communicate. Bees also have a more sophisticated way to communicate, through, waggle dance, which gives more information such as quality, distance and direction of the solutions, which allows more concentrated search.

1.3 ACO variant

The ACO is Max-Min variant.

1.3.1 Max-Min Ant System

MMAS differs from AS in that (i) only the best ant adds pheromone trails, and (ii) the minimum and maximum values of the pheromone are explicitly limited (in AS and ACS these values are limited implicitly, that is, the value of the limits is a result of the algorithm working rather than a value set explicitly by the algorithm designer).

The Max-Min differentiates it self from standard AS; by exploit the best solutions found during an run or iteration, and only update the pheromones, using the best solution. MM also has limits on pheromones, based on highest and lowest pheromones. And initially instantiate the pheromone trails whit the highest pheromone, to get higher exploration rate in the start of the algorithm. I think our (MMAS) implementation is better, because we have the drawback that the number of nodes increases fast, and thus limits the number of iterations for our algorithm. Thus finding good solutions fast, is better for our implemented variant.