IT3708 Assignment 3

Marius Aarsnes

April 2019

1

The JSSP is solved as an inherent discrete problem. To account for this, there have been made a dew modifications to the search procedure.

A new individual is made for every time a bee finds a new solution. This individual consists of the solution together with a value identifying how much exploration away from the solution a bee may explore. The problem is represented as a graph, and each solution contains a combination of integers representing the indices of the vertices in the graph. Each node in the graph has a job ID, machine ID and a the required time for that specific part of the job.

The size of the flower patch is determined by the fitness of the returned solution. The "worse" solutions get a greater neighbourhood to search, while the search space for good solutions are kept small. The neighbourhood search algorithm first uses exploitation until we reach the neighbourhood size. At this point, the algorithm explores possible combination sequences based on probability and a heuristic. Better vertices has a higher change of being selected.

A possible strategy could have been to represent the sequence as real numbers, and try to optimize this before converting back into a combination which can be interpreted as a schedule. This approach would make continuous function optimization possible. However, for this to work, a possibly complex mapping scheme would be necessary. This has the possibility to become too complex and computationally challenging. Therefore, to save time and computational power, the approach described above was chosen.

2

PSO uses the Equation 1 and Equation 2 to move withing the search space, and update a velocity vector and a position vector which identifies different individuals.

$$V_{id} = \omega * V_{id} + C_1 \times Rand() \times (P_{id} - X_{id}) + C_2 \times Rand() \times (P_{gd} - X_{id}) \quad (1)$$

$$X_{id} = X_{id} + V_{id} \quad (2)$$

In my solution a $jobCount \times machineCount$ long array is used to represent a particle. Each element in the array belongs to a specific job and has a position-value and velocity-value defined by Equation 2 and Equation 1. By sorting the array in ascending order using the position-value, a schedule may be obtained by using the ordered array and the problem description saying which particular machine a job needs for which sub-process and how much time this sub-process takes. This is similar to the approach taken in [1].

Another possible approach would be to have a separate array for each machine. Each array would be $jobCount$ long and each element would contain a position-value and an operation-value. Instead of using real values for the positions each position-value is rounded to the closest integer. The position-values then represent the ordering of jobs on each of the machines. This is similar to the approach in [2].

Figure 1: Best makespan per generation for problem 6

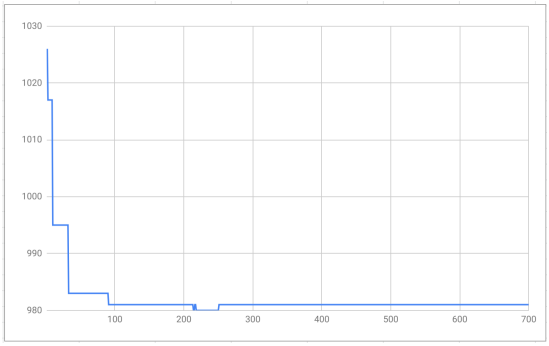

Figure 2: Bee's Algorithm

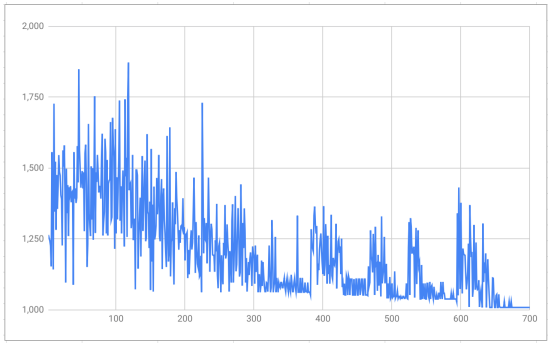

Figure 3: Particle Swarm Optimization

The upside with my approach is that all results are valid solutions. It is not possible to get an invalid or unfeasible solution. The other option has the benefit of being easy to understand, and a closer in representation to the actual phenotype. However, using this representation it is possible to get invalid of unfeasible solutions due to the rounding of values. This may result in duplicate and missing values for certain jobs. This may results in much overhead if a cleanup process is used every time a particle is updated.

3

Figure 1 shows the best makespan for each generation running the two algorithms. Note, it is not the globally best solution for the whole run, but for each individual generation. The most prominent thing to notice is that the PSO algorithm explores much more than the Bee's algorithm. This is due to the Bee's algorithm use of already found (good) solutions and neighbourhood search around them to find new solutions. This results in many vertices being used over and over again and new solutions are made out of old ones. The particles in PSO move around all the time, remembering previously visited solutions (global optimal and local optimal). This way, the particles move around and explore much more in the search space while moving towards good solutions.

The benefits of the Bee's algorithm is that is converges very quickly. This has the added risk of being more prone to getting stuck at local optima. The PSO on the other hand takes longer to converge, but is less prone to getting stuck at local optima. In short, the Bee's algorithm exploits more, while the PSO explores more.

References

[1] Zhixiong Liu. Investigation of particle swarm optimization for job shop scheduling problem. In *Third International Conference on Natural Computation (ICNC 2007)*, volume 3, pages 799–803. IEEE, 2007.

[2] Wei-jun Xia and Zhi-ming Wu. A hybrid particle swarm optimization approach for the job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 29(3-4):360–366, 2006.