Student: Eirik Baug

Project 3 - IT3708

1.

To account for the JSSP being solved as an inherent discrete problem a few modifications have been made to search. As each bee finds a new solution a flower patch is made. This flowerpatch consist of the solution in addition to a size in which the bees can search outside the solution. Every solution is represented by a permutation of integers representing the indexes of the vertices in the graph used to for said solution. A vertex consists of job number, machine number and time required for that part of the job. This could also be viewed as a node.

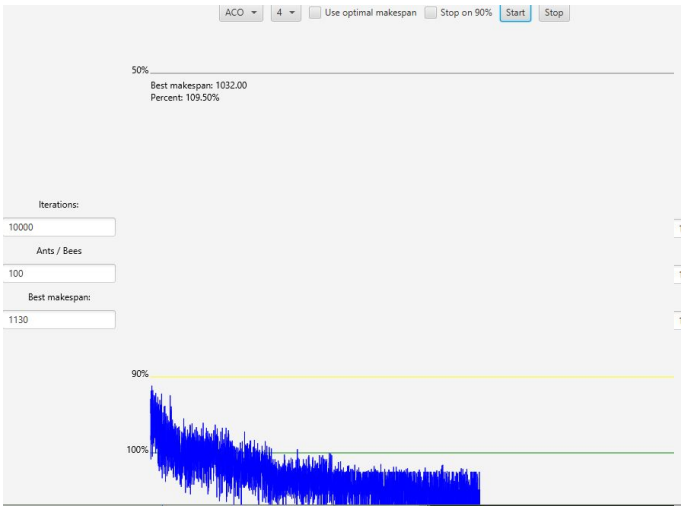
The flower patch size is determined by the fitness of the returned solution, where better solutions get a smaller range to search within. The neighbourhood search algorithm is then modified to build a new solution using exploitation with the vetrix sequence up until a we research the “neighbourhood” determined to be around that solution. From there it’s full on exploration by completing the permutation sequence with new vertices based on a probability distribution for choosing paths based on the heuristic. With this it is a chance the bee will simply find its way back to the original solution, but there is also a good chance it will explore another and better route. Better vertices here have a higher chance of being selected. Aside from how the neighbourhood search is implemented, the rest of BA is fairly standard.

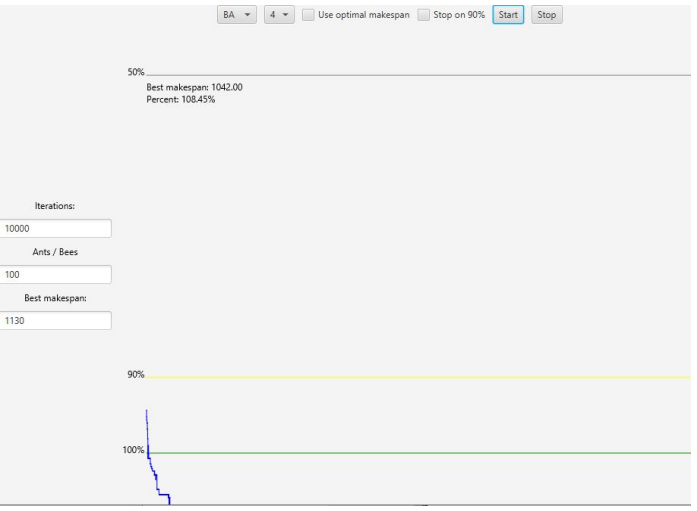
An alternative strategy could be mapping of our integer permutation sequence into real numbers, attempting to optimize this before converting back into a permutation sequence that can be interpreted as a solution. While this has the property of making continuous function optimization possible for the JSSP, it has the drawback of needing a mapping scheme which could become complex to implement and computationally heavy, and therefore we went with what we believed was the fastest and most effective way to reach acceptable values, saving time and computation power.

2.

Vertices Explored						
Problem Set	1	2	3	4	5	6
ACO	337912	983661	988533	985738	1988672	983671
BA	85279	276428	283751	285253	583582	281677
ACO / BA	3.962	3.558	3.484	3.456	3.408	3.492

Student: Eirik Baug





ACO - 10000 iterations *BA - 10000 iterations*

The above images and table shows some differences between ACO and BA, from showing the best found solution each iteration over 10000 iterations against the acceptable values. The overall thing to note is that ACO explores a lot more vertices than BA does (as shown in the table). This is because BA exploits its already found solutions more and performs neighbourhood search around them to find new solutions. This makes it so that a lot of vertices are reused as new solutions are built from old one, while ACO always build new solutions based on heuristic and pheromones. This has the effect on the algorithm that ACO converges slower, but does not as easily get stuck in local optima, but BA converges faster at the cost of more easily getting stuck. This could of course be implementation dependent as we chose an ACO implementation with a high degree of exploration. I.e. BA exploits more previous explored vertices, while ACO explores more.

3.
For ACO we ended up using the MAX - MIN (MMAS) ACO variant. This gave increased performance and saved memory space used by the algorithm. With this we only update the best route every iteration with new pheromone values instead of iterating through every path that was taken by an ant that iteration.
With such algorithms very often having the problem of getting stuck in local optimas we used the tmax and tmin thresholds to avoid this. So we didn't have research ourselves good values for these variables we simply ended up using the values proposed in [1].

The MMAS gave very good results and was a good improvement over the basic ant algorithm. Because of this good performance we felt we had found a good solution to the problem and did not attempt any additional variants like elite and rank.
Tmax and tmin here allowed for more variance and helped us achieve better results. Other strategies were considered, like implementing local pheromone update, which would make the solutions more diverse for each iteration by decreasing the pheromone levels on the vertices after it has been used by an ant. This did, however, not seem necessary as the implementation already performed well and achieved more than satisfactory results in comparison to the acceptable values with good use of exploration and exploitation. MMAS helped improve our use of exploitation on an algorithm that already explored a lot.

[1] https://pdfs.semanticscholar.org/c678/18b2ce1410ba61f29e1f77412fe23c69f346.pdf