CS 169 Final Project Report

Project Title: Population and Stochastic methods on Job-Shop Scheduling Problem

Members:

Yuting Lu, 23688047, yutil15@uci.edu
Tingyin Ding, 70052496, tingyind@uci.edu
Zezhong Zhang, 62529229, zezhongz@uci.edu

1. Introduction and Problem Statement

Job-Shop Scheduling Problem (JSSP) is one of the NP-hard combinatorial optimization problems that there are no effective algorithms to find their optimal solutions in polynomial time. Using limited resources to meet the various constraints of the processed tasks, as well as determining the processing sequence and time of the workpiece on the relevant equipment to ensure the optimal performance, can potentially improve the economic benefits of the enterprise. JSSP has many practical applications and background, and the development of effective and accurate scheduling algorithms is an important topic in the field of scheduling and optimization.

The JSSP problem is defined as following:

Given:

- A finite set of n jobs, each with m operations and processing time $P_{i,0}, P_{i,1}, \ldots, P_{i,m-1}$,
- A finite set of m machines, $M_0, M_1, \ldots, M_{m-1}$, each can handle at most one operation at a time

Propose to allocate each operation in order for each machine to have the minimum total processing time. With constraints:

- No job can process on different machine at the same time
- No machine can process multiple jobs at the same time
- Once a machine starts processing, it needs to finish the job before processing another job.
- All job must be processed in each machine once and only once

To formulize,

- Job set $J = \{J_1, J_2, \ldots, J_n\}$
- Machine set $M = \{M_1, M_2, \ldots, M_m\}$
- Operations $O = \{O_1, O_2, \ldots, O_n\}$ with $O_i = \{ o_{i,1}, o_{i,2}, \ldots, o_{i,m}\}$ and processing time $\{t_{i,1}, t_{i,2}, \ldots, t_{i,m}\}$
- With an ordered sequence $Seq = \{Seq_1, Seq_2, \ldots, Seq_n\}$ of operations, define the makespan as the maximum total processing time needed.

$$makespan(Seq) = max_i\ Time(Seq_i)$$

- Cost function

$$f(Seq) = min_{Seq}\ makespan(Seq)$$

To address the problem, according to the papers we review, we implement the genetic algorithm, the simulated annealing method, and the particle swarm optimization. We compared their performances on the problem, and found the modified genetic algorithm overperforms other algorithms, while it takes a much longer time to finish running 100 iterations. The particle swarm optimization also achieves a suboptimal solution, and may get to the global minimum with more iterations or particles.

2. Related Work:

The scheduling problem has been researched with varied approaches for years. As an NP-hard problem, JSSP is greatly suitable to be solved by heuristics optimization algorithms, including the genetic algorithm, the simulated annealing method, and the particle swarm optimization. These methods have been further improved by researchers. Monique Simplicio Viana and Orides Morandin Junior conduct a modified genetic algorithm with local search strategies and multi-crossover operator and overperforms the original genetic algorithm [Viana, Monique Simplicio, 20]. With Liu's particle swarm optimization decoding strategy [Liu 07], the PSO algorithm can be used to find the optimal in the form of discrete numbers. Advanced strategies in modifying PSO' inertia weight such as linearly decreasing [Bansal, Singh, et al] is utilized in our algorithm.

3. Description of Technical Approach

- Formulation of Job-Shop Scheduling Problem
  The input will be Processing time, and machine sequence is the object to be optimized.

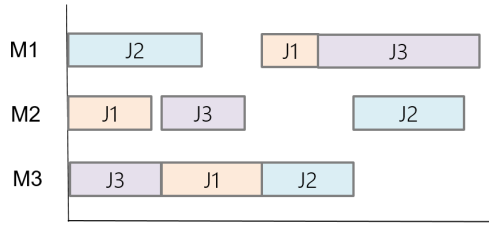
• Processing time

Operation	O1	O2	O3
Job 1	8	10	5
Job 2	15	9	12
Job 3	9	7	19

• Machine sequence

Operation	O1	O2	O3
Job 1	M2	M3	M1
Job 2	M1	M3	M2
Job 3	M3	M2	M1

• Gantt chart



[Wu 18]

- Genetic Algorithm [Gen, Tsujimura, Kubota, 94]

1) Chromosomes Representation
   The n by m JSP matrix is represented by a nm vector that each machine will show n times. For example, if there are n=2 jobs and m=3 machines, a chromosome c = [2, 1, 1, 0, 0, 2] and represent the matrix = [2, 1, 0; 1, 0, 2].
2) Population Initialization
   The initial population is made up of random chromosomes created by reordering $[0, \ ..., 0, 1, ..., 1, ..., m-1, ..., m-1]$.
3) Selection
   Roulette wheel selection is used and the likelihood is calculated as max_c(f(c)) - f(c) to select two chromosomes for each parent.
4) Crossover
   A crossover function set including one point crossover, two point crossover, and uniform crossover is created for choosing each time.
   Since this process will make the chromosome invalid, the child is fixed after crossovering.
5) Mutation
   A mutation function set including swap function that swaps two points and insert function that inserts point j at point i.
6) The best result will be kept and the algorithm will repeat step 3 to step 5 for n iterations.

- Modified Genetic Algorithm with Local Search Strategies and Multi-Crossover Operator [Viana, Monique Simplicio, 20]
  - Selection
    Roulette wheel selection is used and the likelihood is calculated as max_c(f(c)) - f(c) to select three chromosomes for a specific N times.
  - Crossover
    The crossover will be performed to three selected chromosomes with all possible combinations among the three, and will return the best three children. In this way, the crossover part ensures that the return children are not worse than their parents.
  - Mutation
    With a probability of ϵ_{LS}, a local search will be performed with a max iteration limit, and it will try different mutations. Otherwise, only one mutation will be performed.
  - Massive search
    The k best fit children will try all possible mutation parameters, and return the k best fit children.
- Simulated Annealing
  1) The algorithm starts with a given temperature T, decreasing speed y, and iterations control.

2) In each iteration, the algorithm considers some neighbor states s'. In our case, the neighbor will be a swap of an operation pair of one job.
3) The algorithm will move to the neighbor state if a random number p is less than the probability $p = e^{-cost/T}$.
4) The best state with lowest cost will be recorded. As the temperature cools down, it is less possible to move to a neighbor state.

- Particle Swarm Optimization [Liu 07]
  1) PSO starts with P random "particles" initialized into random n by m matrices, where each row in the matrices is a permutation of m machines. Each of these "particles" also obtain a random velocity that is bounded to [-m, +m], since machine 0 can move to machine m with velocity +m, and vice versa. For example, if job 0 (represented by row 0 in every matrix) will be operated by 4 machines, then row 0 of a position matrix may be initialized into [3, 1, 0, 2] with velocity [-1, -2, 1, 3].
  2) In each iteration, new velocity and position is assigned to each particle, and each particle's optimal function value as well as the global optimal function value are stored.

     $$x^i = x^i + v^i$$
     $$v^i = wv^i + c_1 r_1 (x^i{}_{best} - x^i) + c_2 r_2 (x_{best} - x^i)$$

     In the update functions above, c1 and c2 are momentum coefficients; r1 and r2 are random values drawn from U(0,1); and w is the inertia weight [Kochenderfer & Wheeler].
  3) According to PSO's decoding rule on JSP, the initialization of the particles should follow the exact permutation of operation, whereas the operation sequences may become continuous numbers during PSO iterations. During PSO iterations, the velocity is bounded to [-m, m] to ensure the particles shift at reasonable rates [Liu 07].
  4) Since the jobs can be arranged greedily to find the maximal/minimal makespan given increasing or decreasing operation sequence, we can decode the continuous position numbers of PSO into operation sequences by taking the sorted positional arguments of each matrix [Liu 07]. With this decoding strategy, the continuous transition of each particle becomes feasible to represent discrete job arrangement.

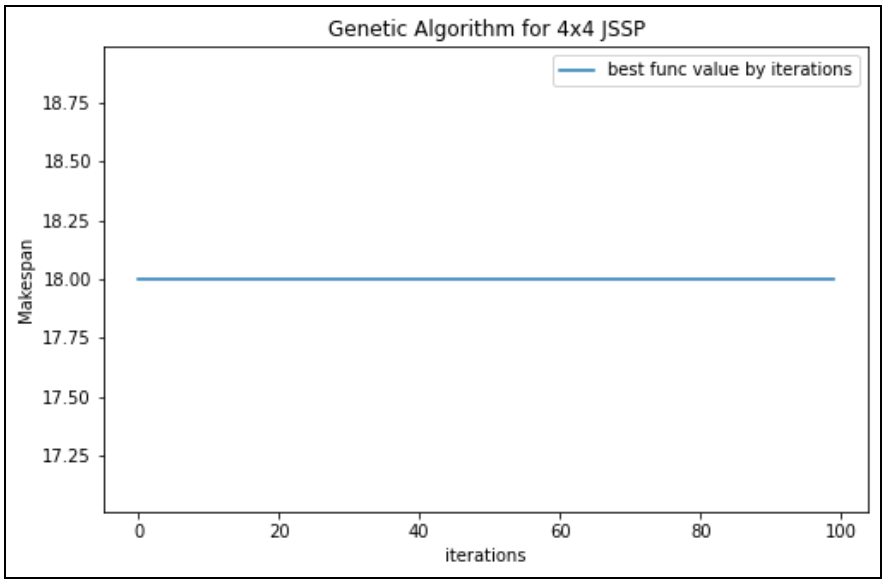
4. Experiments and Evaluation

Basic parameters:
Test 1 #jobs = 4, #machines = 4, time cost = [[1,4,2,1],[2,3,6,2],[3,7,2,3],[4,1,5,8]]
Test 2 #jobs = 10, #machines = 10, time cost = [[7, 9, 2, 1, 10, 14, 17, 11, 6, 11],
[19, 12, 16, 5, 16, 3, 10, 9, 8, 19],
[15, 8, 12, 9, 5, 1, 1, 18, 6, 8],

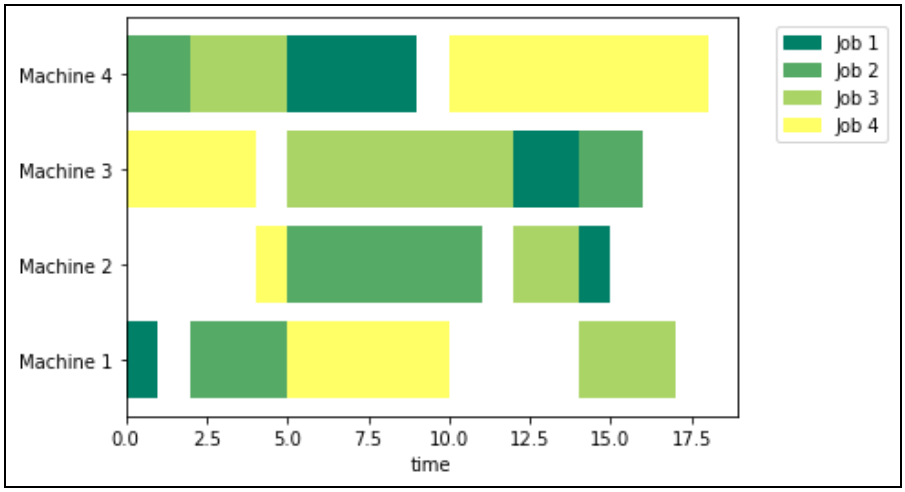
[6, 6, 13, 10, 16, 7, 18, 15, 16, 17],
[4, 13, 9, 6, 17, 12, 18, 13, 11, 1],
[7, 14, 13, 9, 10, 13, 1, 11, 12, 4],
[17, 2, 18, 16, 14, 3, 7, 15, 18, 3],
[7, 9, 8, 12, 2, 9, 3, 10, 17, 9],
[17, 2, 2, 2, 19, 17, 12, 9, 19, 7],
[4, 10, 10, 3, 17, 3, 16, 3, 12, 4]]

- Genetic Algorithm:
  Additional parameters: Population Size = 10 for 4x4 and Population Size = 100 for 10x10, termination criteria: max iteration = 100

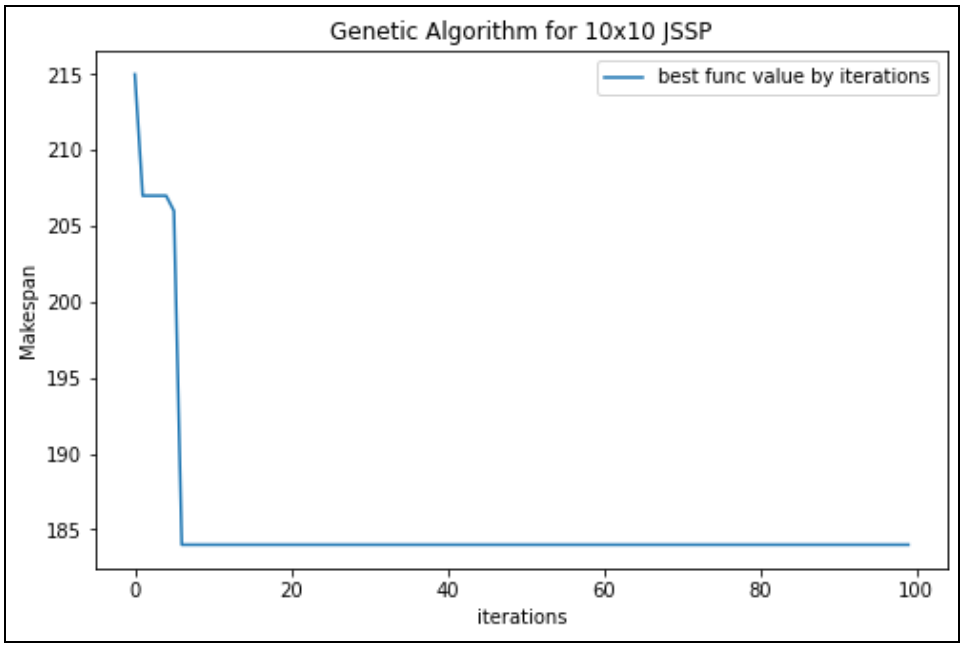
  Test 1 result:
  Minimum makespan by number of iterations: It finds the optimal at the first iteration.



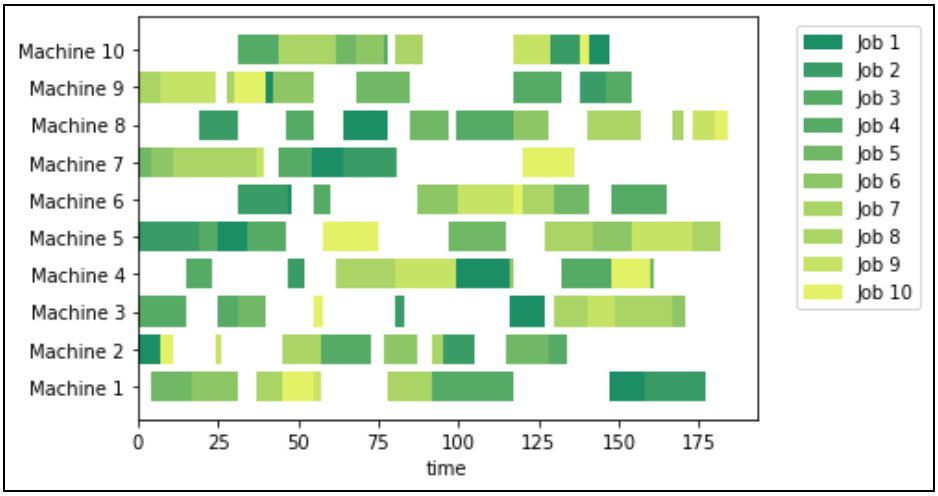
  Job arrangement visualized:



Test 2 result:
Minimum makespan by number of iterations: It finds a suboptimal solution very quickly, but may need more iterations to find the global minimum.

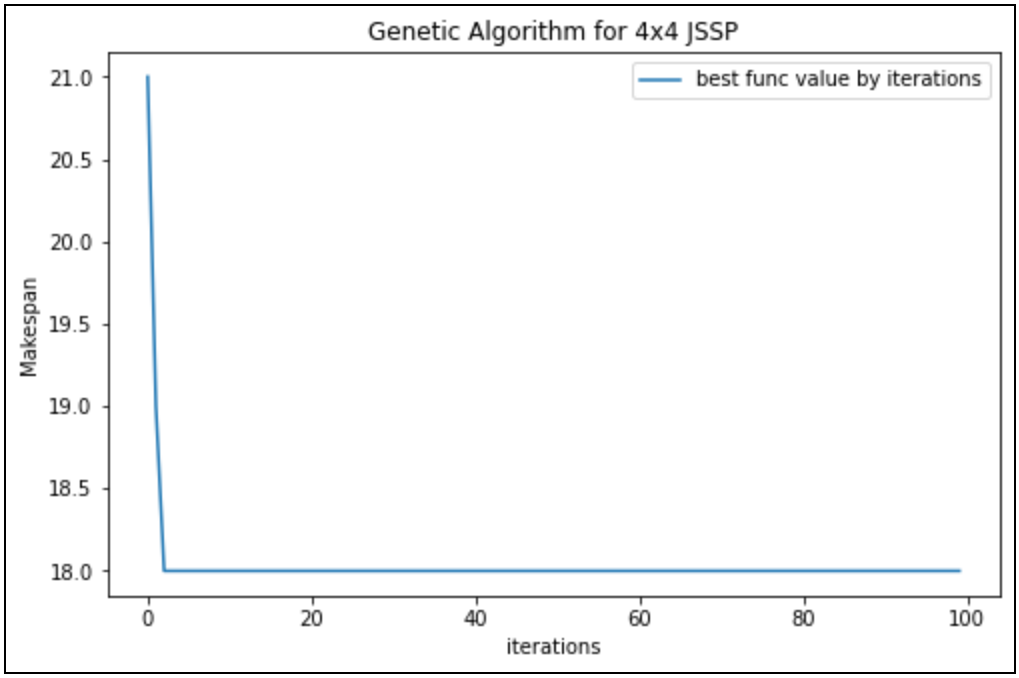


Job arrangement visualized:

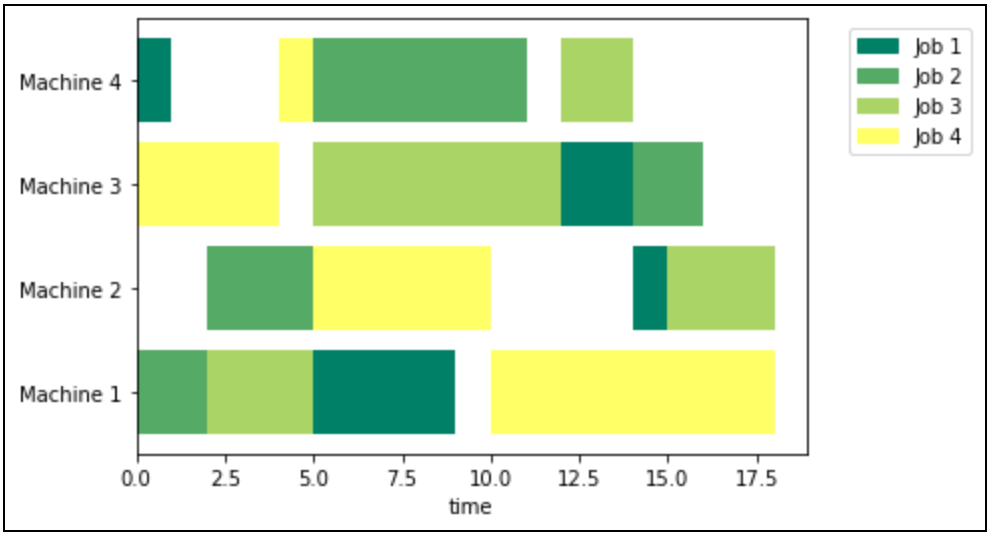


- Modified Genetic Algorithm
  Additional parameters: max_iterations = 100, population_size = 10 for JSSP 4x4 and 100 for JSSP 10x10, probability to do local search $\epsilon_{LS} = 0.95$, max_crossover = 10, best_k = 2

  Test 1 result:
  Minimum makespan by number of iterations: It finds the optimal very quickly.

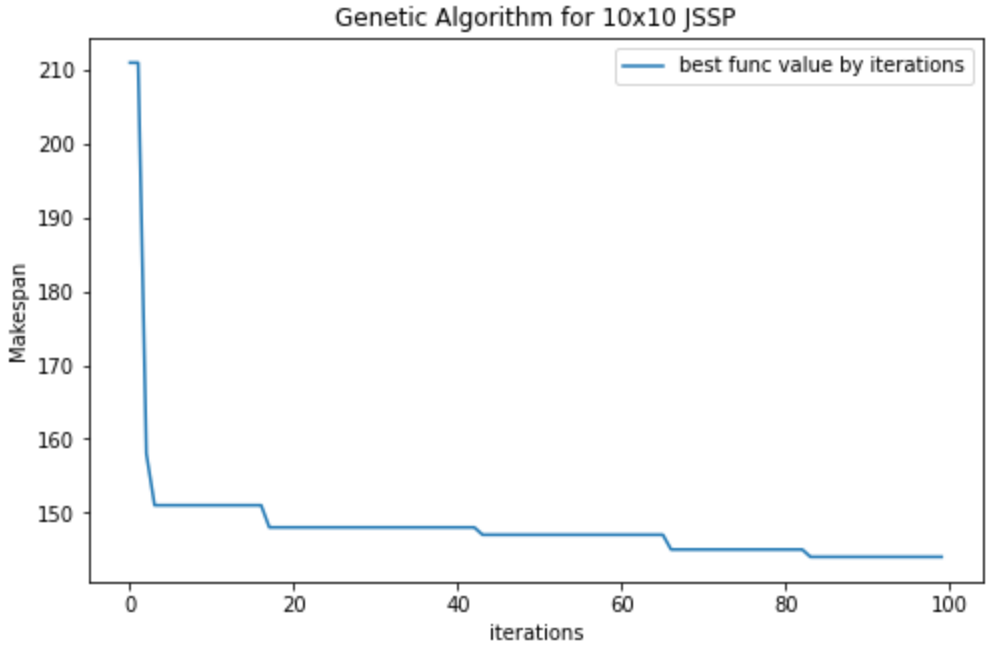


  Job arrangement visualized:

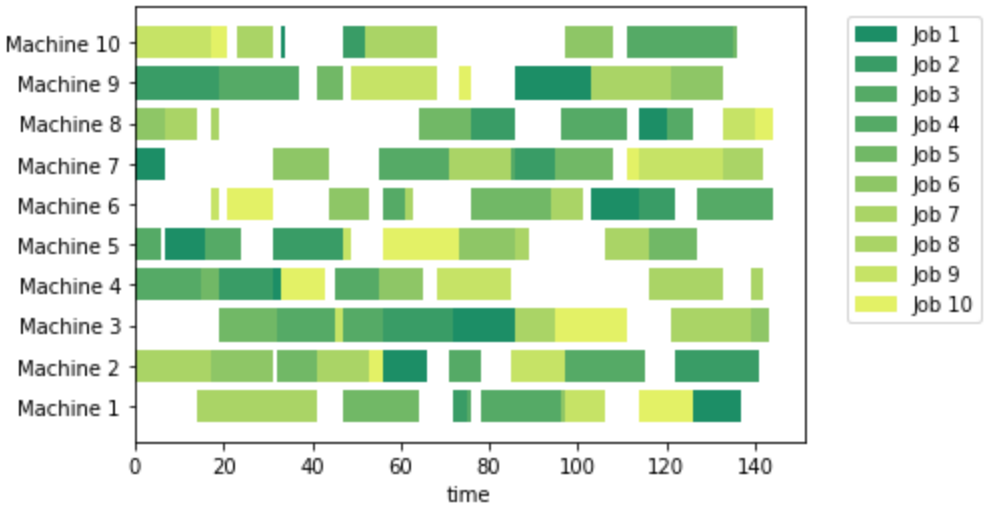


Test 2 result:

Minimum makespan by number of iterations: It seems can find even a better solution if with more iterations.

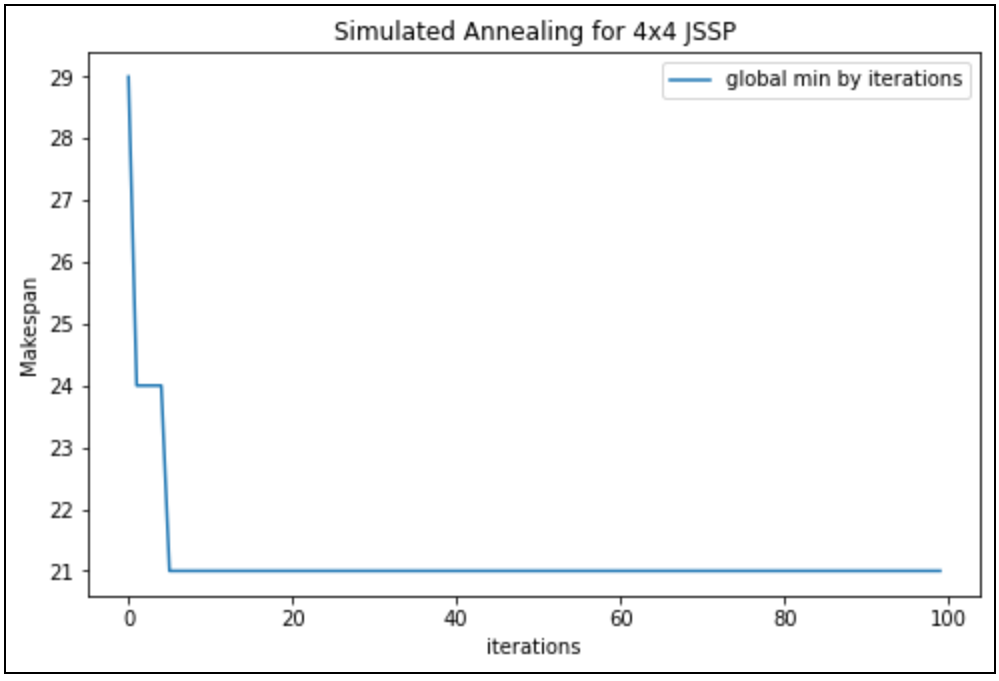


Job arrangement visualized:

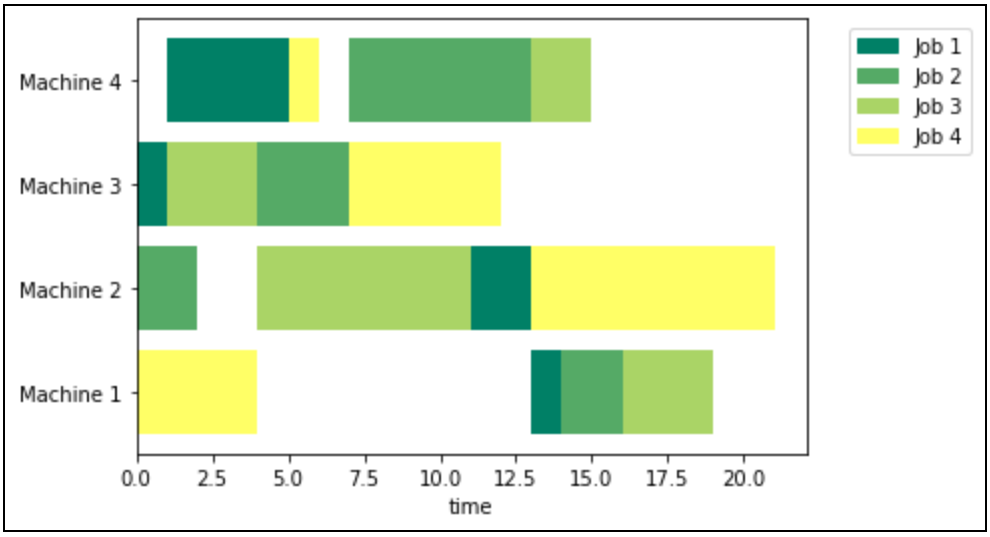


- Simulated Annealing:
  Additional Parameters: Temperature T=200, decreasing speed y=0.8, and max_iterations=100

  Test 1 result:
  Minimum makespan by number of iterations: It stuck at local minimum.

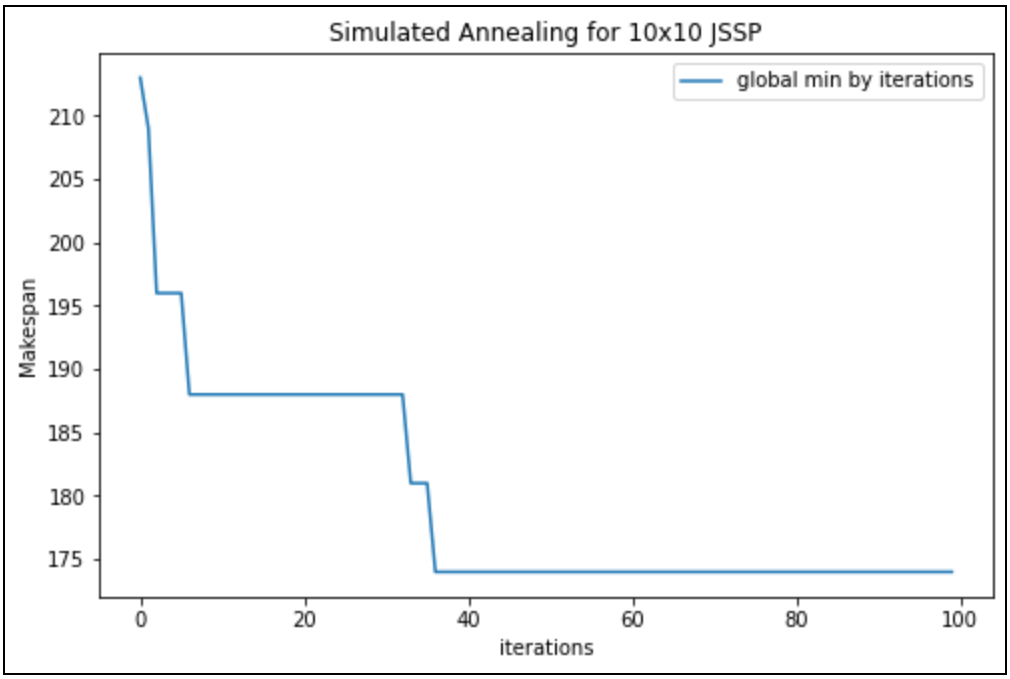


  Job arrangement visualized:

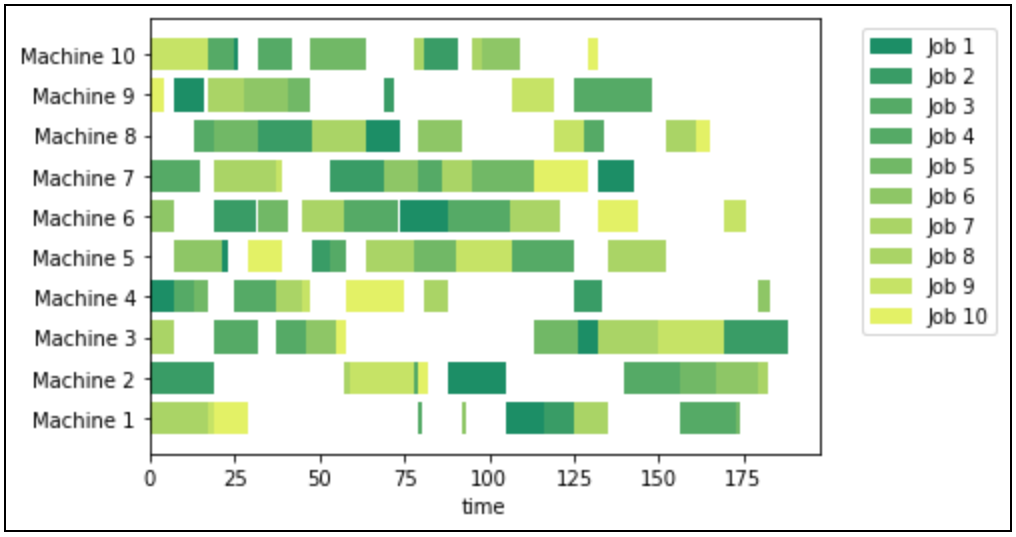


Test 2 result:

Minimum makespan by number of iterations: It may need more iterations to find the global minimum.

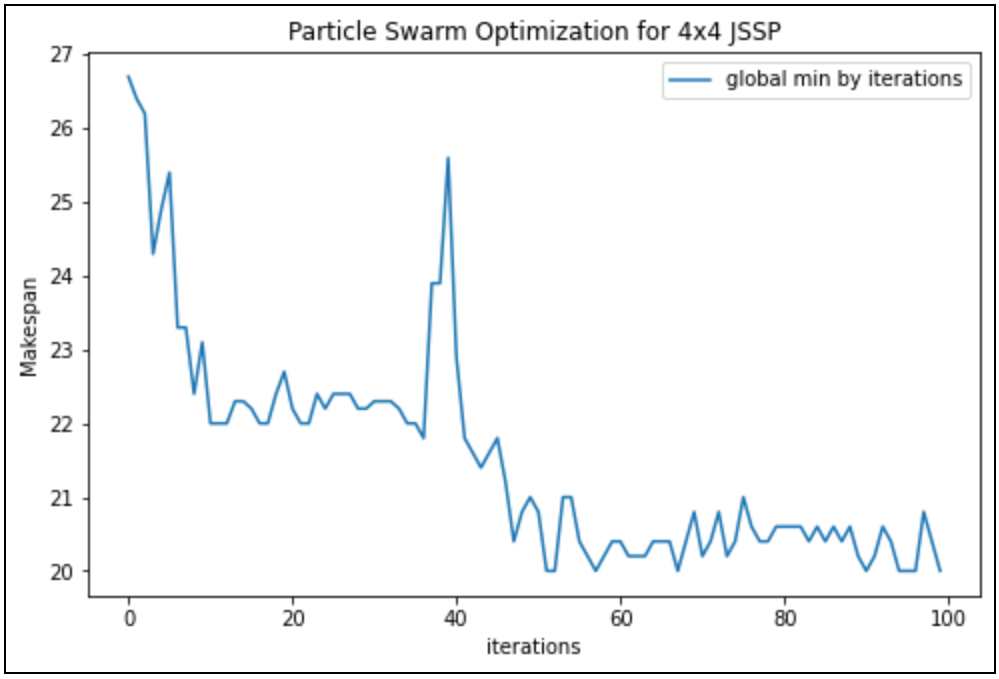


Job arrangement visualized:

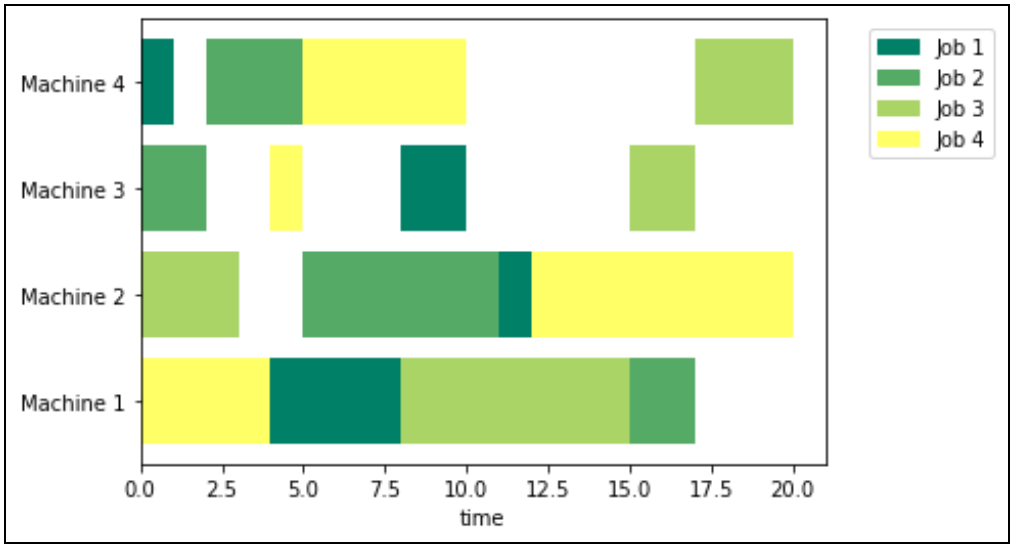


- Particle Swarm Optimization:
  Additional parameters: #Particles = 10 for test 1 and 100 for test 2, linearly decreasing inertia weight, termination criteria: max iteration = 100

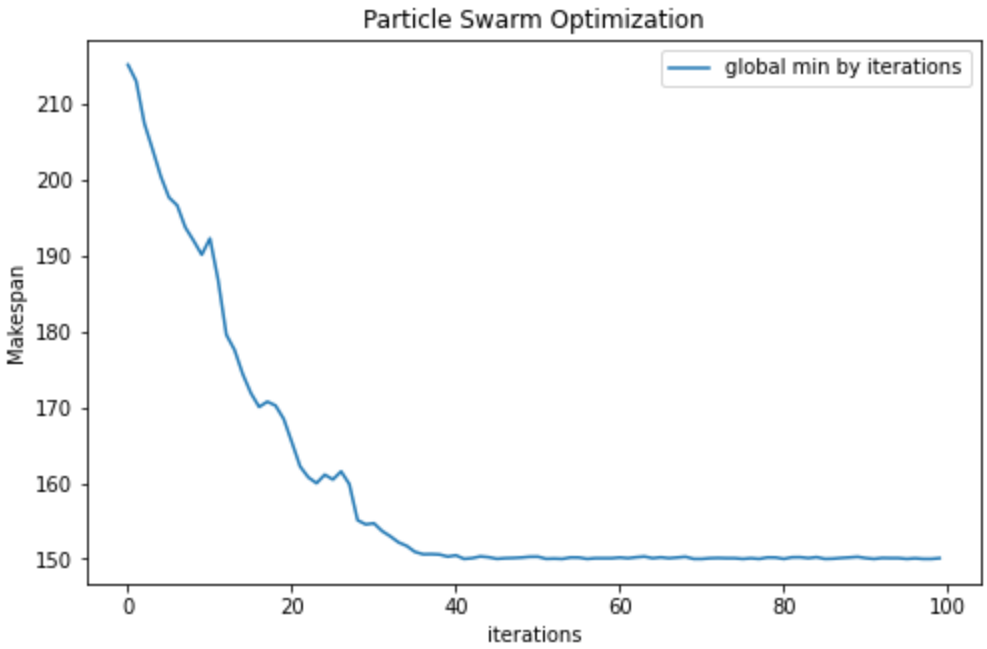
  Test 1 result:
  Minimum makespan by number of iterations: It may need more iterations to converge.



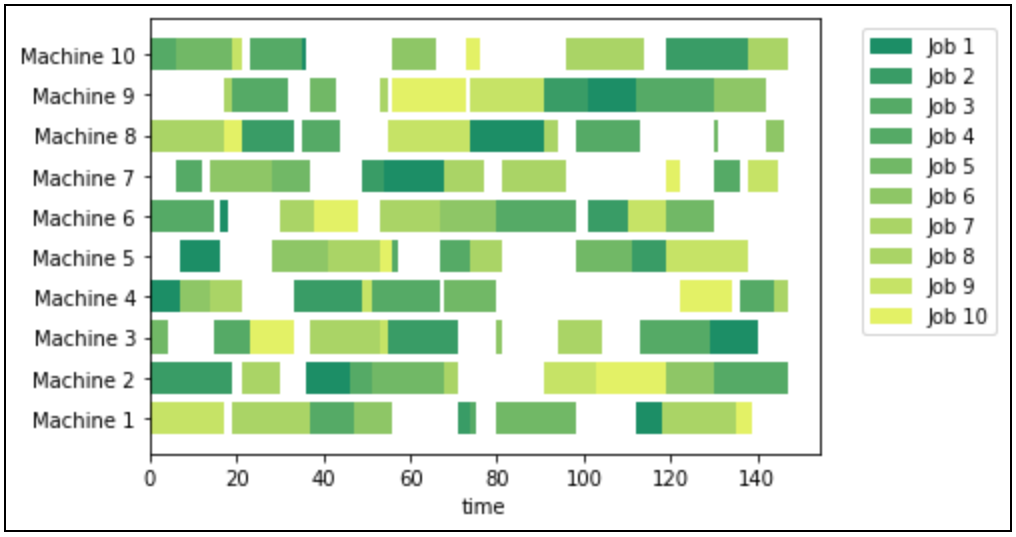
  Job arrangement visualized:



Test 2 result:
Minimum makespan by number of iterations:



Job arrangement visualized:



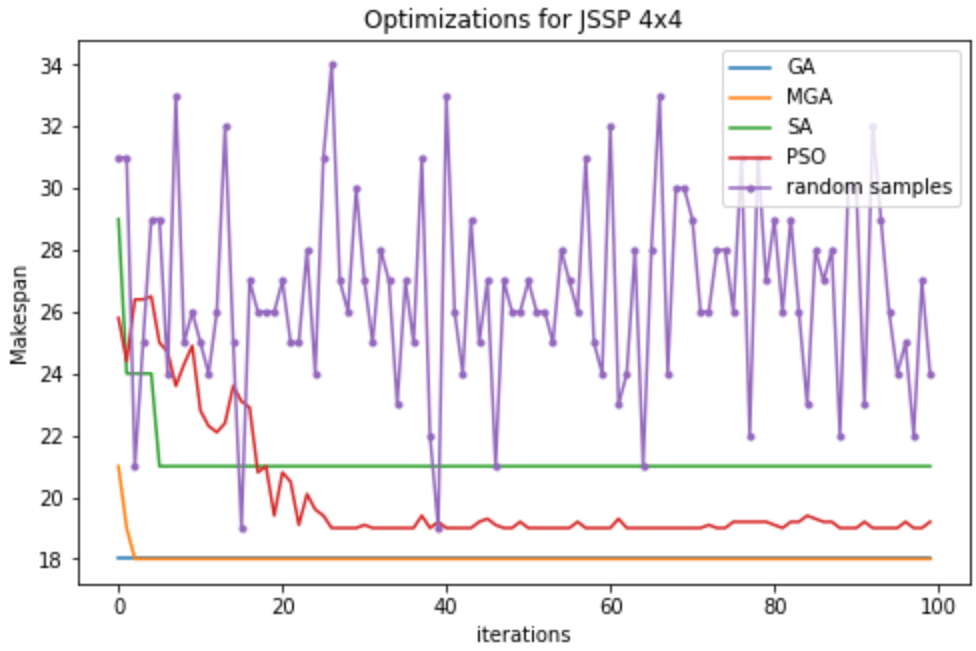
- Summary:

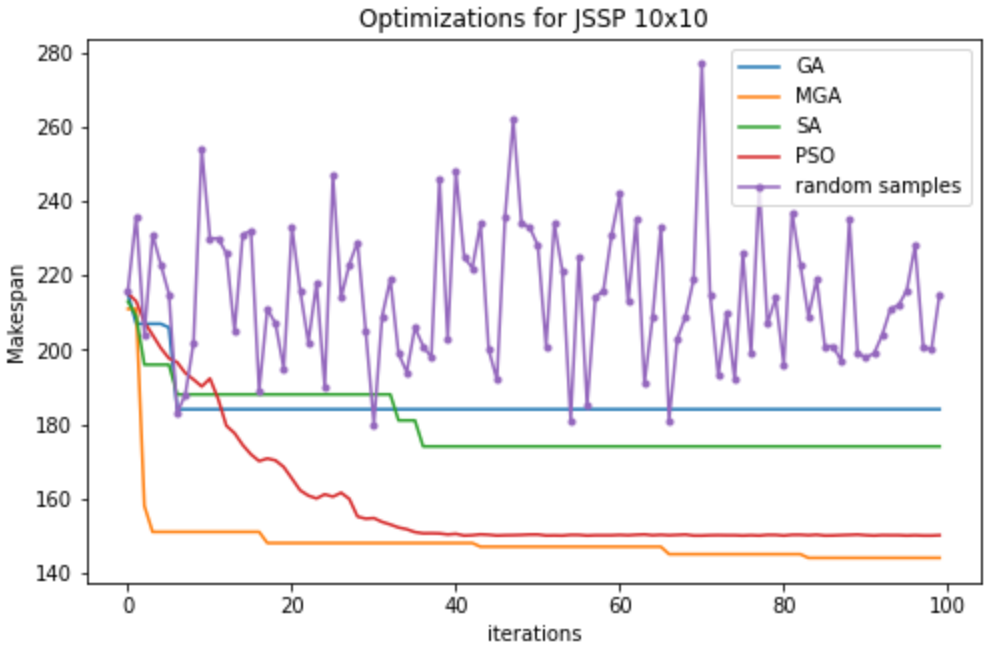
4x4 JSSP

	GA	MGA	SA	PSO
Best makespan	18	18	20	20
Time used	1.280	15.90	9.46	0.25

10x10 JSSP

	GA	MGA	SA	PSO
Best makespan	184	144	176	150
Time used	11.36	1547.34	299.98	8.57





5. Discussion and Conclusion

- Genetic Algorithm

The original genetic algorithm is able to find the optimal if the problem size is small. With a large enough population size and large enough iterations, it will eventually converge but it will take a very long time. The modified genetic algorithm is able to find the optimal for both small size problem and large size problem, but it takes longer time each iteration. The good

performance may be due to the modified crossover section that ensures the children's improvement in performance. Meanwhile, the massive search on the best k children makes it more likely to turn a suboptimal solution into the optimal solution.

However, GA seems still much slower than other algorithms like PSO. In this case, optimizing its parameters (especially population size and number of max iterations) may be necessary in order to speed up, whereas finding the optimal parameters also needs a long time and the parameters would become too specific to generalize to other situations.

Moreover, it is possible to stop early for GA by stopping when there is only one chromosome. However, it is hard to tell whether it is a premature point. Trade-off between premature terminations and time cost should be considered.

- Simulated Annealing

The Simulated Annealing is supposed to find the global optimum given a temperature T. As long as the temperature is high, the algorithm is more flexible to move to neighbor states to explore the global optimum. In our experiment, the Simulated Annealing does not succeed to find the global optimum. The random factors have a great impact and the algorithm performs dangling between different neighbors instead of finding the optimum. One possible reason for the failure is that the neighbor heuristic is not designed properly in our experiment. In our experiment, a swap of an operation pair of one job is defined as a neighbor, which might not represent the neighbor relationship properly and lead to the relative random results.

- Particle Swarm Optimization

In our test case, the algorithm converges quickly (4*4 = 16 dimensions at 20th iteration). When relying on a good particle population, the algorithm is able to find the global minimum in a fairly small amount of time.

When the population is drawn badly, a delay in convergence may appear. Sometimes it may also cause failure in convergence or stuck at local minimum (such as 19 or 20 in our case). To ensure convergence and avoid sticking at local minimum, advanced strategy in picking parameters, c1, c2, and w should be applied. We utilized the linearly decreasing inertia weight in our case, but it still failed to avoid local minimums in some rare population distributions.

Our future modification on PSO may focus on changing the momentum coefficients and inertia weight. Candidate strategies of picking inertia such as natural exponent and Sigmoid decreasing [Bansal, Singh, et al] will be tested in the future.

- Overall conclusion

The job shop scheduling problem is a complex model that needs specific strategies when performing optimization on relating problems. Our results are only based on 2 cases, while none of the algorithms we conducted in this paper performs outstandingly in terms of running time and accuracy. As described above, each of our algorithms has potential enhancements, and more mature optimization strategies such as termination criteria and parameter choices can be

implemented. We might also need a larger number of test cases to derive more convincing results. Our future work will utilize more of these strategies and focus on different variations of JSP.

6. Work Division

Yuting Lu: Responsible for the simulated annealing algorithm and the random search algorithm; responsible for making the poster.

Tingyin Ding: Responsible for constructing the JSP problem in python; responsible for the genetic algorithm and the modified genetic algorithm; responsible for conducting the experiments for the algorithms.

Zezhong Zhang: Responsible for the PSO algorithm; responsible for conducting the experiments for the algorithms.

7. References

Kochenderfer, M.J. and Wheeler, T.A. *Algorithms for Optimization*. MIT Press, 2019, ISBN:9780262351409. https://books.google.com/books?id=oxyNDwAAQBAJ

Z. Liu, "Investigation of Particle Swarm Optimization for Job Shop Scheduling Problem," Third International Conference on Natural Computation (ICNC 2007), Haikou, 2007, pp. 799-803, doi: 10.1109/ICNC.2007.453.

Wu, Cheng-man, "Solving Job shop scheduling problem with genetic algorithm," Github, 2018. https://github.com/wurmen/Genetic-Algorithm-for-Job-Shop-Scheduling-and-NSGA-II/blob/master/implementation%20with%20python/GA-jobshop

M. Gen, Y. Tsujimura, E. Kubota, Solving job-shop scheduling problem using genetic algorithms, Proc. of the 16th Int. Conf. on Computer and Industrial Engineering, Ashikaga, Japan (1994), pp. 576-579, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=400072&tag=1

Viana, Monique Simplicio, et al. "A Modified Genetic Algorithm with Local Search Strategies and Multi-Crossover Operator for Job Shop Scheduling Problem." MDPI, Multidisciplinary Digital Publishing Institute, 22 Sept. 2020, www.mdpi.com/1424-8220/20/18/5440/htm.

J. C. Bansal, P. K. Singh, M. Saraswat, A. Verma, S. S. Jadon and A. Abraham, "Inertia Weight strategies in Particle Swarm Optimization," 2011 Third World Congress on Nature and

Biologically Inspired Computing, Salamanca, 2011, pp. 633-640.doi: 10.1109/NaBIC.2011.6089659

Hannesfrank, Richard Kwasnicki, Job Shop Scheduling, Github. https://github.com/hannesfrank/jobshop