Population and Stochastic methods on Job-Shop Scheduling Problem

Yuting Lu, yutil15@uci.edu. Tingyin Ding, tingyind@uci.edu
Zezhong Zhang, zezhongz@uci.edu, University of California, Irvine

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 Pi,0, Pi,1, ... , Pi,m-1,*
- *A finite set of m machines, M0, M1, ... , Mm-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 = {J1,J2, ... , Jn}*
- *Machine set M = {M1,M2, ... , Mm}*
- *Operations O = {O1, O2, ..., On} with Oi = {oi,1, oi,2, ... , oi,m} and processing time {ti,1, ti,2, ... ,ti,m}*
- *With an ordered sequence Seq = {Seq1, Seq2, ... , Seqn} 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 we used random search as a baseline algorithm to evaluate the performance of the algorithms above.

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
https://github.com/hannesfrank/jobshop

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.

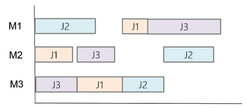
- Processing time

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]*
We represent the sequences as random Chromosomes and implement selection, cross over, and mutation methods to operate these chromosomes. The rest result will be kept after n iterations.

Modified Genetic Algorithm *[Viana, Monique Simplicio, 20]*
We used the improved version of genetic algorithm with local search strategies and multi-crossover operator.

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 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
The algorithm starts with a given temperature T, decreasing speed y. It will move to the neighbor state if a random number p is less than probability $p = e^{-cost/T}$. In our case, the neighbour will be a swap of an operation pair of one job.

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].

2) In each iteration, new velocity and position is assigned to each particle and the optimum value will be stored.

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.

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.

Results

Basic parameters:
Test 1 #jobs = 4, #machines = 4, time cost matrix 1
Test 2 #jobs = 10, #machines = 10, time cost matrix 2

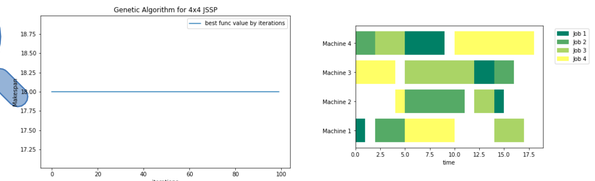
```
[[1, 4, 2, 1],
 [2, 3, 6, 2],
 [3, 7, 2, 3],
 [4, 1, 5, 8]]
```

matrix 1

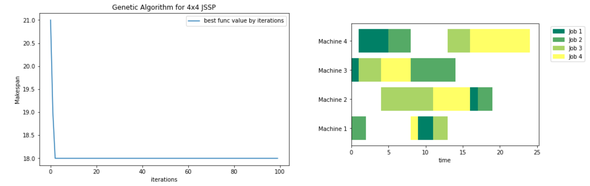
```
[[ 6,  6, 13, 10, 16,  7, 18, 15, 16, 17],
 [ 4, 13,  9,  6, 17, 12, 18, 13, 11,  1],
 [ 7, 14, 13,  9, 10, 13,  1, 11, 12,  4],
 [17,  2, 18, 16, 14,  3,  7, 15, 18,  3],
 [ 7,  9,  8, 12,  2,  9,  3, 10, 17,  9],
 [17,  2,  2,  2, 19, 17, 12,  9, 19,  7],
 [ 4, 10, 10,  3, 17,  3, 16,  3, 12, 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

matrix 2

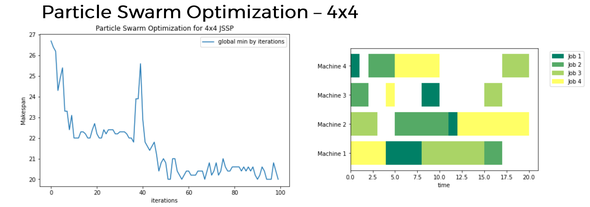
Genetic Algorithm – 4x4



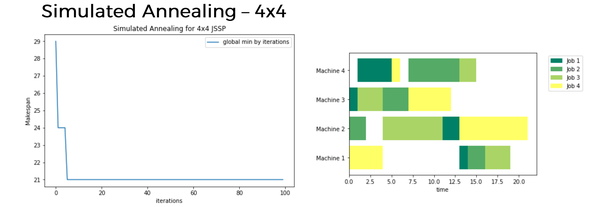
Modified Genetic Algorithm – 4x4



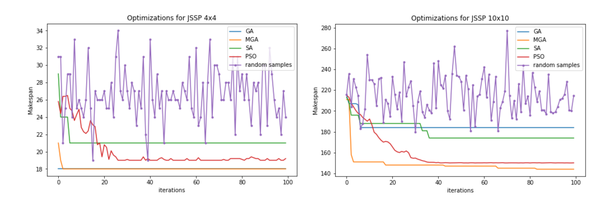
Particle Swarm Optimization – 4x4



Simulated Annealing – 4x4



Summary



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 fail 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.

10x10 JSSP

	GA	MGA	SA	PSO
Best makespan	184	143	176	147
Time used	11.36	1547.34	299.98	8.57

4x4 JSSP

	GA	MGA	SA	PSO
Best makespan	18	18	20	20
Time used	1.280	15.9	9.46	0.25