# Applying the Particle Swarm Optimization + Hill-climbing in the Flexible Job-Shop problem

**João Baptista Cardia Neto**
*Computer Science Graduate Programa*
*São Carlos Federal University - UFSCAR, Brasil*
*joao.cardia@fatec.sp.gov.br*

**Edilson Reis Rodrigues Kato**
*Computing Department*
*São Carlos Federal University - UFSCAR, Brasil*
*kato@dc.ufscar.br*

***Abstract—The Flexible Job-Shop problem is a very interesting and important problem. In this paper it is studied an approach of the PSO (Particle Swarm Optimization) in the Flexible Job- Shop problem, the studied and applied approach is derived from a Travelling Salesman Problem solution with a few minor alterations, trying to reach the optimum values discovered by a series of other works. The goal of this paper is to indicate machines configuration that better supports the its restriction and productivity necessity.***

***Keywords-PSO, Job-Shop***

## I. INTRODUCTION

The Flexible Job-Shop problem has attracted the attention of several researchers in the fields of production management and combinatorial optimization, this is due the fact that is NP-Hard and it has a great impact in increasing the profit and product quality in different industries [1]. There are several approaches into utilizing PSO to solve the aforementioned problem, in [1] the authors proposed a hybrid intelligent algorithm integrating PSO and AIS (Artificial Immune System) it also adjust the fitness in each iteration to prevent premature convergence. In [2] the authors modifies the particle position to a based on preference list-based representation, the particle movement based on a swap operator and a particle movement based on a Tabu search, in their paper the results show that the PSO outperforms other meta-heuristics.

In [3] the authors utilizes a Pareto approach to solve the problem utilizing PSO and a local search algorithm to achieve the optimum. The majority of works that combines PSO with a local search utilizes the hierarchical representation, separating the routing and the scheduling. While the main meta-heuristics search for a more efficient routing the local search focus on finding the better scheduling for a given routing.

The current work utilizes the velocity calculation from [4] and another set of optimizations directly in the scheduling in an attempt to reach the optimum found in other works.

## II. FLEXIBLE JOB-SHOP PROBLEM

In the current work the studied problem consists in a set of N Jobs (each job with a set of operations) that needs to be processed and finished in M Machines. The main purpose is to evaluate the time taken to finish all the operations from all the jobs and this work tries to find the best combination of a set of operations inside a machine in order to minimize the amount of time taken to conclude all operations.

## III. PARTICLE SWARM OPTIMIZATION

The Particle Swarm Optimization (PSO) was originally proposed by Kennedy and Eberhart in [5] and, since then, has been largely applied in a series of problems.

It is similar to genetic algorithm but, specially in the sense that the system is initialized with a set of random solutions. The major diversion from the previous is how these solutions are guided in direction to the solution.

Each particle receives an stochastic velocity that helps to guide them to explore and exploit the solution space. For this to happen each particle records its best position ($P_{best)}$ and, combining with the best globally solution ($t_{tbest}$) found by the swarm, it tries to scape local optimums.

PSO algorithm has the following steps:

- Step 1 - Generate initial random population for each partcle $X_i$;
- Step 2 - Calculate the fitness for each particle $X_i$;
- Step 3 - Evaluates the system, if reached the stopping conditions, stop;
- Step 4 - Updates $P_{best}$ and $t_{tbest}$;
- Step 5 - Calculates new $X_i$ positions;
- Step 6 - Go to step 2.

A new position for a particle $X_i$ is defined by [4]:

$$X_i(k+1) = X_i(k) + V_i(k+1) \tag{1}$$

being *k* the current iteration, *i* the particle number and $V_i(k+1)$ the velocity for the current particle in the next iteration.

$V_i(k+1)$ is defined by:

$$V_i(k+1) = wV_i(k) + c_1r_1(P_i - x_i(k)) + c_2r_2(P_g - x_i(k)) \tag{2}$$

being w the inertia coefficient with values in the interval [0, 1]. Both $c_1$ and $c_2$ are defined as stochastic acceleration terms, the last two terms ($r_1$ and $r_2$) are random values in the interval [0, 1].

The stopping condition can be defined as a number of iterations or if the algorithm found a fitness value that satisfies the problem constraints.

Classical PSO (the algorithm described above) is only applicable for continuous problem and, the Job-Shop and a big range of applications, are discrete. To solve this problem is necessary the adjustment of the way that new particles positions are generated.

## IV. DISCRETE PSO

Inspired in [4] in this work the velocity of each particle is represented as a set of permutations. With this in mind, the velocities are calculated in the following form:

- Get two particles, *a* and *b*;

- Create the permutation Pab as given:

$$P_{ab} = \begin{bmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \end{bmatrix} \qquad (3)$$

- The velocity is the permutation array;

Another operation that has to be rewritten is the multiplication. In the current work, when multiplying a constant to a velocity vector, the constant dictates the chance that a single permutation has to happen.

The formula for the new position of a particle is given by [4]:

$$\begin{aligned} V_i(k+1) = c_1 R_1^k (P_i^k - x_i(k)) + c_2 R_2^k (P_g^K - x_i(k)) + \\ w(c_1 R_1^{k-1} (P_i^{k-1} - x_i(k-1))) + \quad (4) \\ c_2 R_2^{k-1} (P_g^{k-1} - x_i(k-1)) \end{aligned}$$

In the above formula the $R_1^{k-1}$ and $R_2^{k-1}$ are the random values utilizes in the $k - 1$ iteration of the system.

The new position for a particle $x_i(t + 1)$ is defined as the application of the permutation defined in $V_i(k + 1)$ to $x_i(k)$.

## V. HILL-CLIMBING

Hill-Climbing search is one of the simpler ways to perform a local search in order to find the a local maxima. It algorithm is very simple and consists in, given one solutions, looks at the neighbours of that solutions and find the one with better value. It is not maintained a search tree or any way of looking forward. The following algorithm shows how to implement a Hill Climbing Search, this is based on [6].

```
function hillClimbing (item, iterations)
    bestItem <— item
    while iterations > 0
        neighbours <— getNeighbours(item)
        for n in neighbours
            if value(n) > value(bestItem)
                bestItem <— n

        iterations <— iterations − 1

    return bestItem
```

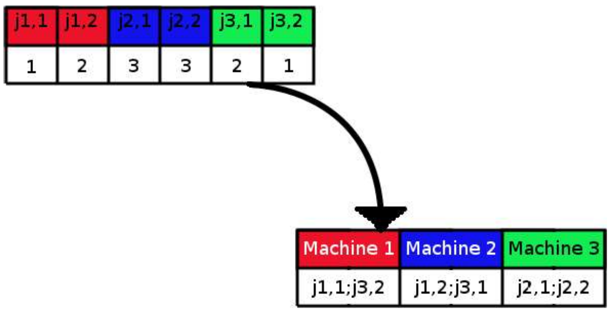


Fig. 1. Both representations for the Job-Shop problem utilized in the current work. The Vector with 6 positions is the router vector and the other is scheduling vector. This is a example for a solution in a 3X3 problem.

## VI. PROBLEM MODELING

In the current work a hierarchical modeling is utilized. There are two vectors defined for each particle, one is know as routing and the other scheduling. Both of them describes different sides of the same problem.

For each particle the routing vector shows in which machine each of the operations from each job will be made. The scheduling shows the order of each one of these operations in the machine. The Figure 1 shows both the representations.

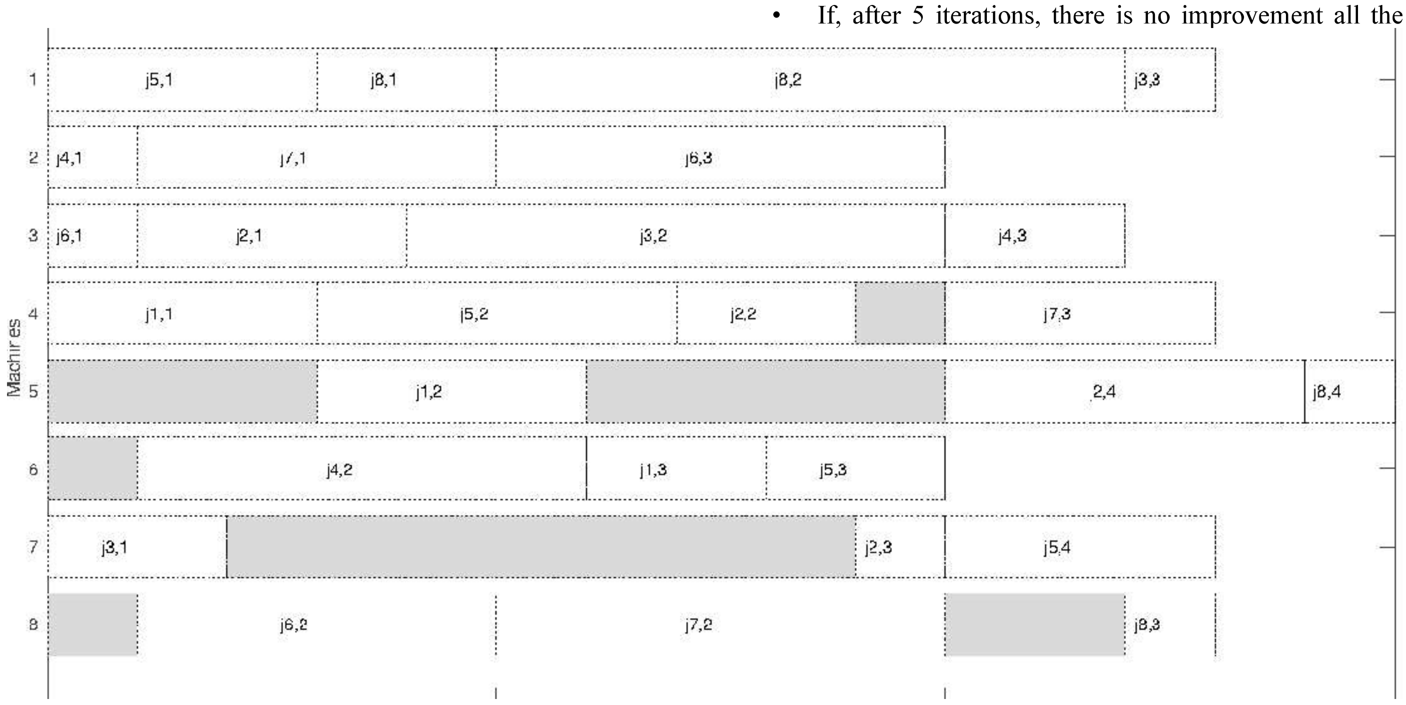
## VII. IMPLEMENTED SYSTEM

For the current work a set of optimizations are implemented trying to reach the optimum found in other works. This set is described as follows:

- Workload equalization: If there is a new $G_{best}$ and the difference in the quantities of operations from two ma- chines is bigger than one an operation from the machine with more items is moved to the one with less operations. This is repeated until there is no machine with more than two operations than the other machine. If the equalized ttbest has a fitness smaller than the original, the equalized becomes the new $G_{best}$;
- A permutation is only applied to generate a new particle if the fitness from the new element is better or equal from the old one;
- For each particle $x_i(_t)$ eight neighbours are generated.

The new particle $x_i(t + 1)$ is the one with best fitness from the eight neighbours;

- If, after 5 iterations, there is no improvement all the



new $x_i(t+1)$ receives new random values - utilizing the same algorithm from the initial population;

- If a new particle already was generated in an older iteration, it goes trough a process of N random mutations, until is a never seen before particle.

For each slot for each operation in the initial population is made a roulette utilizing the processing cost. Smaller the cost for a operation in a machine the higher the chance to it be in the population.

TABLE I - COMPARISON OF RESULTS FOUND IN LITERATURE

| Proposed System | Makespan |
|---|---|
| PSO + SA [8] | 15 |
| Proposed Work | 15 |
| PSO + TS 2009 | 14 |
| MOPSO 2011 | 14 |

### A. *Applying the Hill-Climbing*

The Hill-Climbing is utilized as an auxiliary meta-heuristics. After calculating the new $x_i$ the scheduling for the position is used as the input for the algorithm. If the output from the search has a better fitness than the original scheduling than it is utilized.

## VIII. EXPERIMENTS AND RESULTS

For the current work two experiments were made, utilizing the benchmark given in [7]. The current work was evaluated on the 8X8 and 10X10 problem, with 50 particles and 30 iterations. The values of w = 0.9, c1 = 0.5 and $c_2$ = 1.5. There were made 1000 iterations for the Hill-Climbing.

The minimum Makespan in the experiment with 8X8 was 15. The Figure 2 shows the Gantt chart for the result, Figure 3 shows the convergence curve for the implemented system. Finally, Figure 4 shows the Boxplot for the experiment.

The Table I compares the results of the current work with others found in the literature.

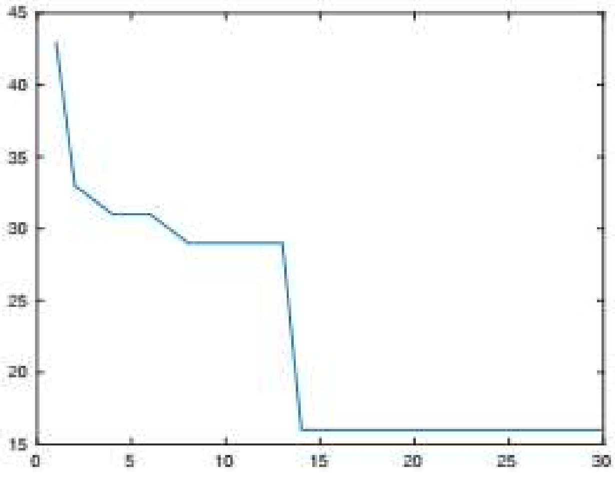


Fig. 3. The convergence history for the best solution. It is possible to see that the algorithm converges around the iteration 14 15.

For the 10X10 the minimum makespan was 9. But, from the 30 executions, the values wont went above 10, this means that the system is stable and can perform at its best in almost all executions. The Figure 5 shows the machine configuration for a 10X10 solution, the Figure 6 shows the conversion curve of the same problem and Figure 7 ilustrates the Boxplot for the current executions.

The Table II shows the results with other works in the literature.

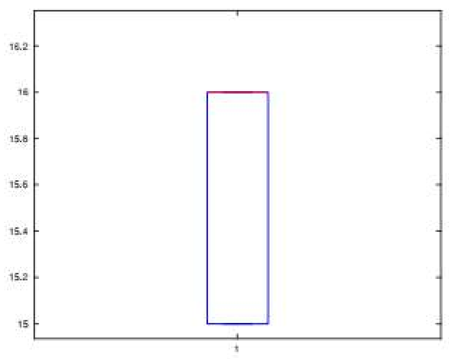


Fig. 4. Boxplot from the 30 consecutive executions of the proposed work. All of them returned a value between 16 and 15.

TABLE II
COMPARISON OF RESULTS FOUND IN LITERATURE

| Proposed System | Makespan |
|---|---|
| PSO + SA [8] | 7 |
| Proposed Work | 9 |
| PSO + TS 2009 | 7 |
| MOPSO 2011 | 7 |

## IX. CONCLUSION

The results shows that, utilizing PSO+Hill Climbing to solve the Job-Shop problem proved to be effective. All result are in the 10% range of the best solutions found in the literature.

There are a few more thing that could be changed to reach the optimum, things such as utilizing the level approach as in [8] or utilizing another kind of local search, such as Tabu List search.

Looking at the results it show clearly that improvement in the scheduling is possible and, one of the extensions of this work, would be trying to develop new heuristics that focus on creating solutions with less idle time.

Looking at the convergence history it possible to see that, one of the feature that helped the algorithm to escape local optimum is the reset combined with the mutation applied.

For the future work it would be interesting to utilize the critical path algorithm to generate the initial population and do a grid search to find the best parameters to achieve best results with the proposed work.

## REFERENCES

[1] H. W. Ge, L. Sun, Y. C. Liang, and F. Qian, "An effective pso and ais-based hybrid intelligent algorithm for job-shop scheduling," IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 38, no. 2, pp. 358–368, March 2008.
[2] D. Sha and C.-Y. Hsu, "A hybrid particle swarm optimization for job shop scheduling problem," Computers Industrial Engineering, vol. 51, no. 4, pp. 791 – 808, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360835206001471
[3] G. Moslehi and M. Mahnam, "A pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search," International Journal of Production Economics, vol. 129, no. 1, pp. 14 – 22, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925527310002938
[4] X. Shi, Y. Liang, H. Lee, C. Lu, and Q. Wang, "Particle swarm optimization-based algorithms for tsp and generalized tsp," Information Processing Letters, vol. 103, no. 5, pp. 169 – 176, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/ pii/S0020019007000804
[5] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm the- ory," in Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on, Oct 1995, pp. 39–43.
[6] S. J. Russell and P. Norvig, Artificial Intelligence: A

Modern Approach, 2nd ed. Pearson Education, 2003.
[7] I. Kacem, S. Hammadi, and P. Borne, "Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems," IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 32, no. 1, pp. 1–13, Feb 2002.
[8] W. Xia and Z. Wu, "An effective hybrid optimization approach for multi- objective flexible job-shop scheduling problems," Computers Industrial Engineering, vol. 48, no. 2, pp. 409 – 425, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360835205000197

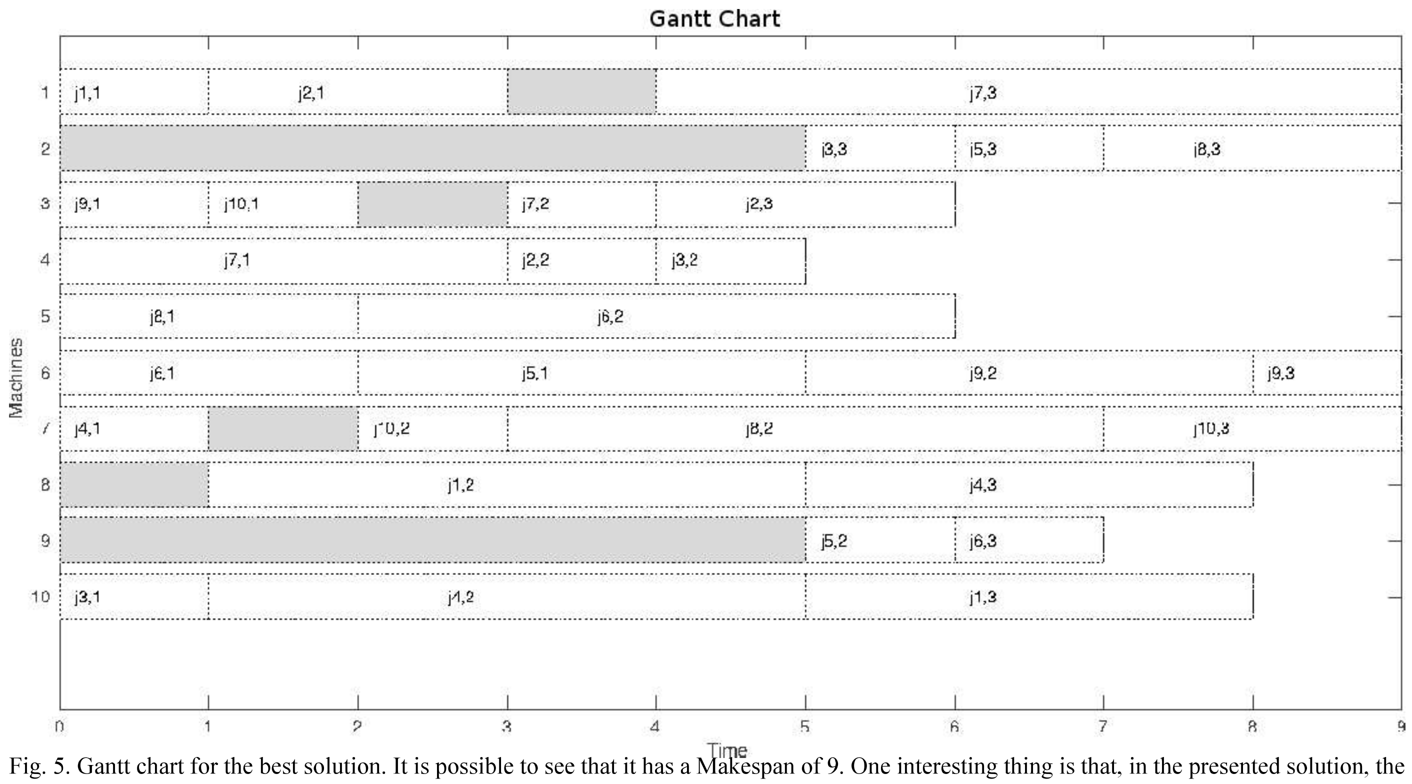


Fig. 5. Gantt chart for the best solution. It is possible to see that it has a Makespan of 9. One interesting thing is that, in the presented solution, the machines have similar workloads.

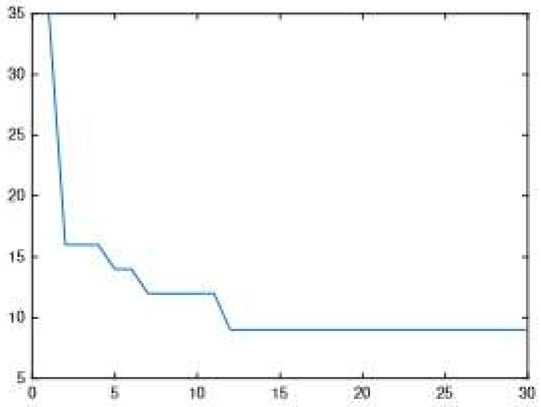


Fig. 6. The convergence history for the best solution. It is possible to see that the algorithm converges around the iteration 11 12.

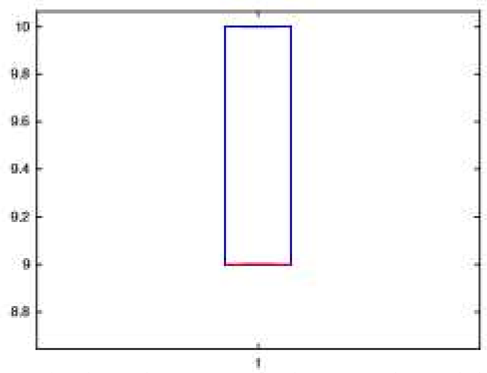


Fig. 7. Boxplot from the 30 consecutive executions of the proposed work. All of them returned a value between 9 and 10.