A Comparison of Hybrid Natural Computation Methods for Solving the Job Shop Scheduling Problem

Hichame Moriceau - 40171260

Bsc Computing Science - Computational Intelligence course
School of Computing,
University of Napier Edinburgh

April 15, 2016

Abstract

In the last decades, natural computation methods have demonstrated their simplicity and global search abilities in a wide range of applications and the Job Shop Scheduling Problems is a notoriously difficult problem which has been extensively researched. Previous work on hybrid evolutionary approaches using a local search have shown a better ability to solve JSSPs. In this paper, a comparison of a Hybrid Genetic Algorithm using a tournament scheme with a neighborhood search with an Hybrid Artificial Immune System with the same local search is made. Firstly, the theory behind each method is explained, followed by a description of the approach used to convey the experiment. Subsequently, the obtained results with each evolutionary technique are displayed and discussed in the conclusion. Finally some paths worth exploring based on the literature and the obtained results are suggested.

1 Introduction

Job Shop Scheduling Problems (JSSPs) are NP-Hard combinatorial optimization problems, it is defined by finding an optimal repartition of operations or tasks. Each problem consist of a number of jobs and a number of machines where each job has a task for each machine and each task takes the amount of time required for the machine to process it. An optimal solution is defined by the shortness of makespan (schedule's length). In the context of Evolutionary Algorithms (EAs) the JSSP is therefore a minimisation problem.

2 Previous Work

In the last fifty years, a tremendous amount of research has been done in solving the JSSP. It is known from [1] and [2] that conventional approaches primarily include heuristic procedures such as a *priority rule* for selecting a job over a set of unscheduled jobs. More recently, a singular focus on using probability-based or constructive search methods such as *Simulated Annealing*, *Tabu Search*, *Genetic Algorithms* or *Hybrid Genetic Approaches* was observed in the literature. [2] shows that genetic algorithms perform poorly at fine-tuning around local or global optima, this justifies the interest in hybridization. Also, it is known that JSSPs defined by large number of machines and jobs but also jobs with identical number of jobs and machines are substantially harder.

3 Methodology

3.0.1 Scope

The first approach used is based on an elitist Genetic Algorithm using a tournament scheme. The second is based on an Artificial Immune System (H-AIS). Both algorithms include a local search within their main loop in order to encourage exploitation. Considering the amount of time available, the two algorithms will only be benchmarked on two problems of largely different complexity and two largely different population sizes.

3.0.2 Representation of a Single Individual

In the two algorithms, the chosen genotype for this project was a real encoded two dimensional array where each row corresponds to a machine and each element in this row correspond to a task.

3.0.3 Search Space of Feasible Solutions

The search space of JSSPs contain unfeasible solutions. In order to optimize the search, all the search operations (initial population, crossover, mutation etc.) only produce feasible solution, doing so excludes parts of the search space that are not authentic solutions.

3.1 Neighborhood Search

In order to increase the exploitation capabilities of the algorithm, one solution is to introduce a local search within the behaviour of the algorithm. Neighborhood search in the context of the JSSP can take several meanings. In this project, the logic of the neighborhood search is as follows. Firstly, the algorithm randomly select a machine, then it selects two neighbor jobs and swap them.

3.2 Elitist Genetic Algorithm with Neighborhood Search

3.2.1 Algorithm Description

Figure 1 illustrates the structure of the optimization algorithm. It is the template used for creating the hybrid GA using a tournament scheme with neighborhood search (H/GA-tnmt/NH).

Firstly, a population of random solution is generated, then, the main loop is performed. This loop is constituted of 5 evolution phases. In the *evaluation phase*, the fitness value of each individual is calculated, then, parents are chosen in the *selection phase*. In the canonical elitist GA algorithm the two fittest individuals are selected as parents.

The selected individuals then produce a mutated offspring, by first applying a crossover operation to produce the new individual and applying a mutation operation on the latter. The *crossover phase* uses the genotypes of the selected parent to produce a novel combination of these. In this project, attention was only paid to using a one point, two

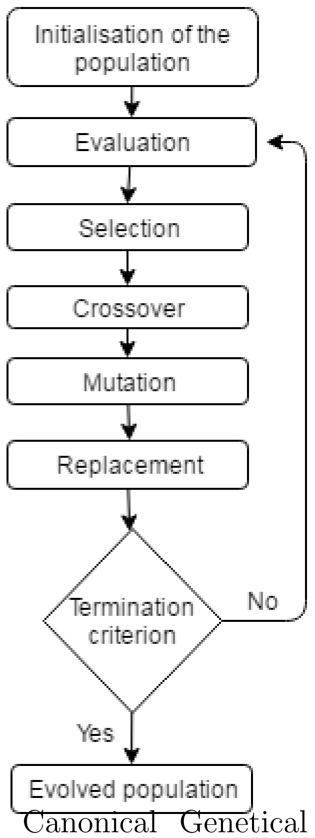

Figure 1: Canonical Genetical Algorithm. Sources: [3] and [4].

point or three parents crossover. It was experienced that the three parents crossover produced better results, therefore it is the one that was selected.

The *mutation operation* signifies that the genotype of the offspring may or may not be randomly altered. In this project, a random machine is selected and two of its tasks are swapped.

3.3 Clonal Selection with Neighborhood Search

The *Clonal Selection Algorithm* (CLONALG) algorithm is an evolutionary method for minimizing a cost function, the Algorithm 1 is used as a template for the Hybrid AIS/neighborhood (H-AIS-NH). Although it performs a multimodal search, CLONALG belongs to global search algorithms. In order to compensate this lack of local search ability and achieve a fair comparison with the Genetic Algorithm, the same neighborhood search is introduced within the main loop of the algorithm.

Algorithm 1 Clonal Selection Pseudocode

```
 1: $pop\_size \leftarrow 80$
 2: $selection\_size \leftarrow pop\_size \times 30/100$
 3: $nb\_r\_cells \leftarrow pop\_size \times 10/100$
 4: $clone\_rate \leftarrow 5$                      ▷ Scaling factor
 5: $mutation\_rate \leftarrow 0.7$ ▷ Mutation decay factor
 6: $pop \leftarrow create\_rand\_cells(pop\_size)$
 7: repeat
 8:     for eachindividual i in pop do
 9:         $compute\_affinity(i)$
10:     end for
11:     $pop_{select} \leftarrow select(pop, selection\_size)$
12:     $pop_{clones} \leftarrow empty\_list$
13:     for eachindividual i in pop_select do
14:         $population_{clones} \leftarrow clone(i, clone\_rate)$
15:     end for
16:     for eachindividual i in pop_select do
17:         $hypermutate(i, mutation\_rate)$
18:         $compute\_affinity(i)$
19:     end for
20:     $pop \leftarrow select(pop, population_{clones}, pop\_size)$
21:     $pop_{rand} \leftarrow create\_rand\_cells(nb\_r\_cells)$
22:     $replace(pop, population_{rand})$
23: until stopping criteria
24: return $pop$
```

3.3.1 Algorithm Description

Initially, a population of random individuals (referred to as cells). Then, the optimization loop composed of five distinct phases is initiated.

Firstly, an *evaluation of the affinities* of all cells is performed. In this specific use case of an AIS, the affinity corresponds to the single-valued measure indicating the quality of the individual, As regards to the JSSP, it corresponds to the fitness of the solution (i.e., the makespan).

The *reproduction or cloning* phase is then performed, by producing copies of the current cell population. The number of clones generated for each cell is inversely proportional to its affinity, ensuring that the population thrives to learn from the existing best solutions.

Subsequently, the *hypermutation of the clones* is done by mutating the clones is performed by mutating individuals using a mutation rate inversely proportional to its affinity.

Finally an *injection of randomly generated cells* is made into the population to improve diversity and similarly as in EAs *cells replacement* is achieved.

This process is achieved, until a given criteria is met (e.g. a certain number of generations was performed) the main optimization loop is performed.

4 Experimental Plan

4.0.2 Benchmarking Architecture

The software uses the following (hard-coded) experimentation settings to run the algorithms.

- Number of replicates
- Number of generations
- Population size
- Tournament size
- Selection pressure
- Problems to be solved

Then the software produces the following metrics for each iteration of training.

- Problem ID
- Number of Jobs
- Number of Machines
- Optimization algorithm (0=H/GA-tnmt/NH, 1=H/AIS/NH)
- Generation
- Fitness of best individual
- Lower Bound of current problem
- Mean of the fitness of individuals
- Variance of the fitness of the individuals

This process allows an automated run of the algorithms on various problems using different settings.

4.0.3 Mean of the Fitness Values

Since the two approaches are constructive population-based optimization algorithms, the mean of the fitness values of the entire population should tend to decrease over time (since JSSPs are minimization problems). Example of this can be observed on Figure 3

4.0.4 Variance of the Fitness Values

The variance plot 4 shows how quickly the H/GA-tnmt/NH converges. Considering that the H-AIS maintains a 10% subset of its population as random individuals, a much noisier curve can be observed 4

4.0.5 Using Multiple Replicates

Considering the randomness of EAs induced by the initial random population and the probability of mutation but also because of the stochasticity of each algorithm, a single run of the algorithm is not enough to make any conclusion.

Using multiple replicate and averaging the obtained results paints a more accurate picture of the general behaviour of each algorithm. This process is achieved by default in all experiments.

5 Results

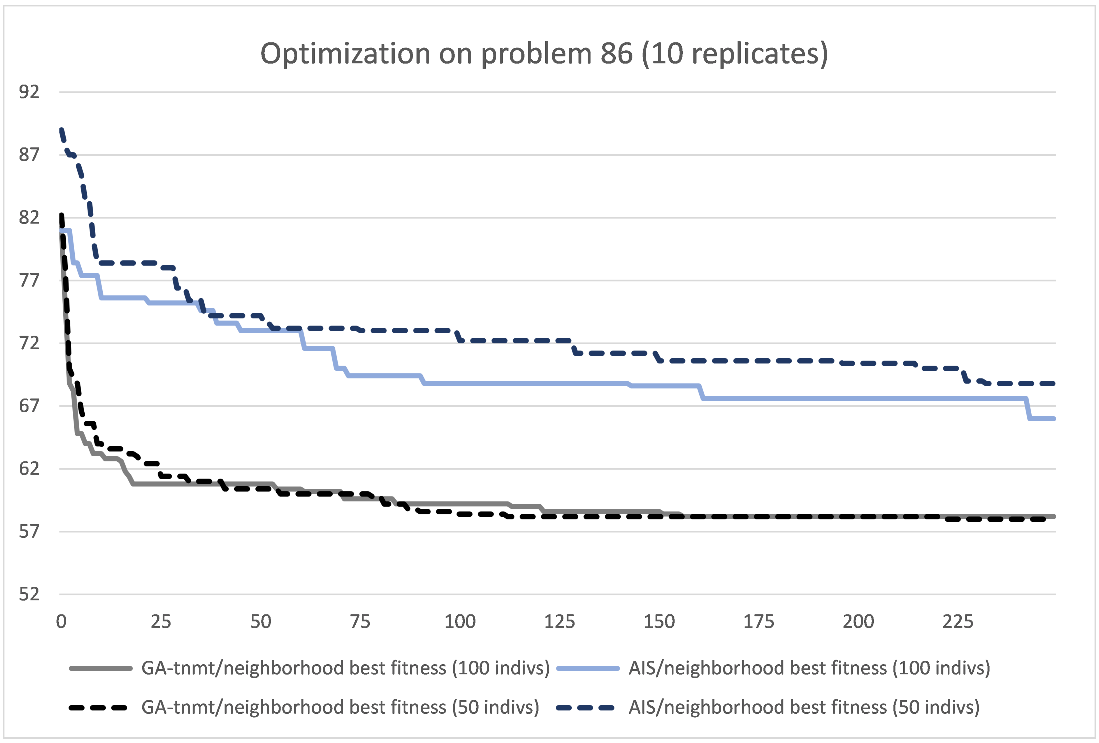

Figure 2: Performances of the two algorithms with different population sizes (Lower bound = 52).

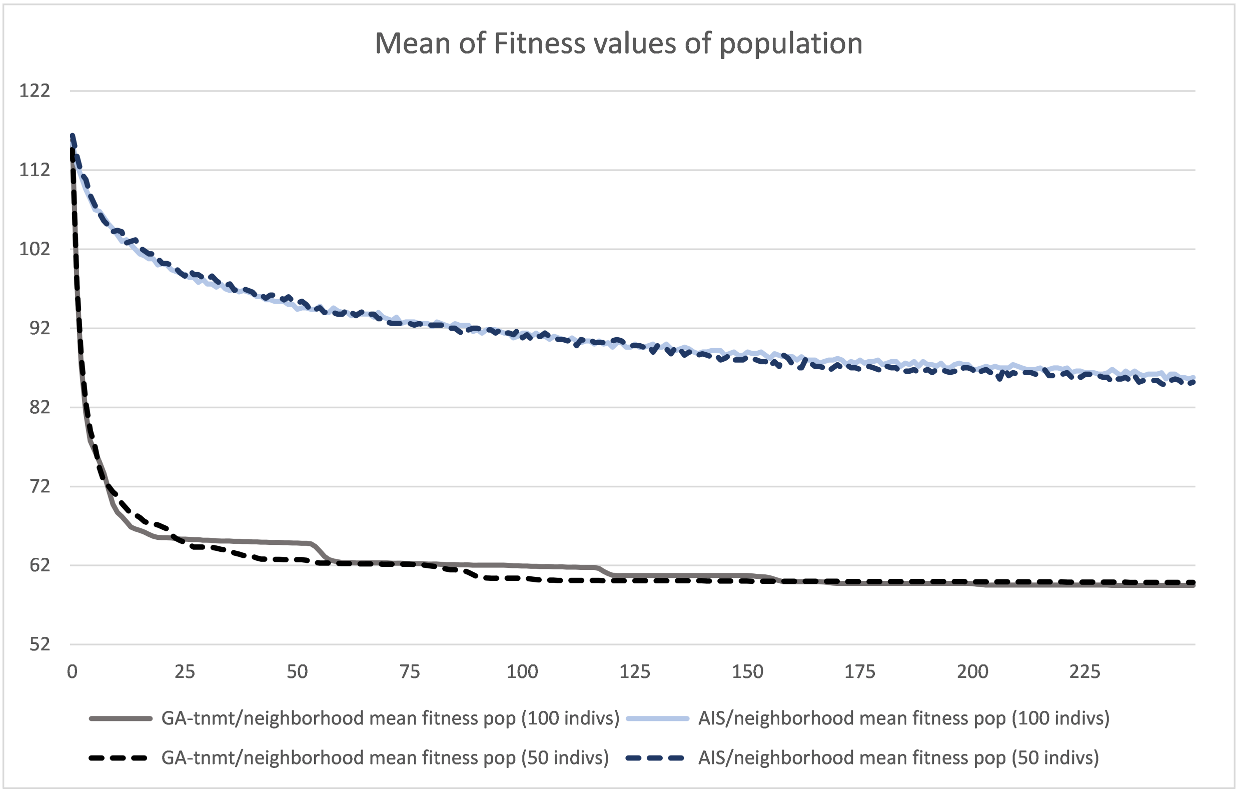

Figure 3: Mean of the fitness of all individual (Lower bound = 52).

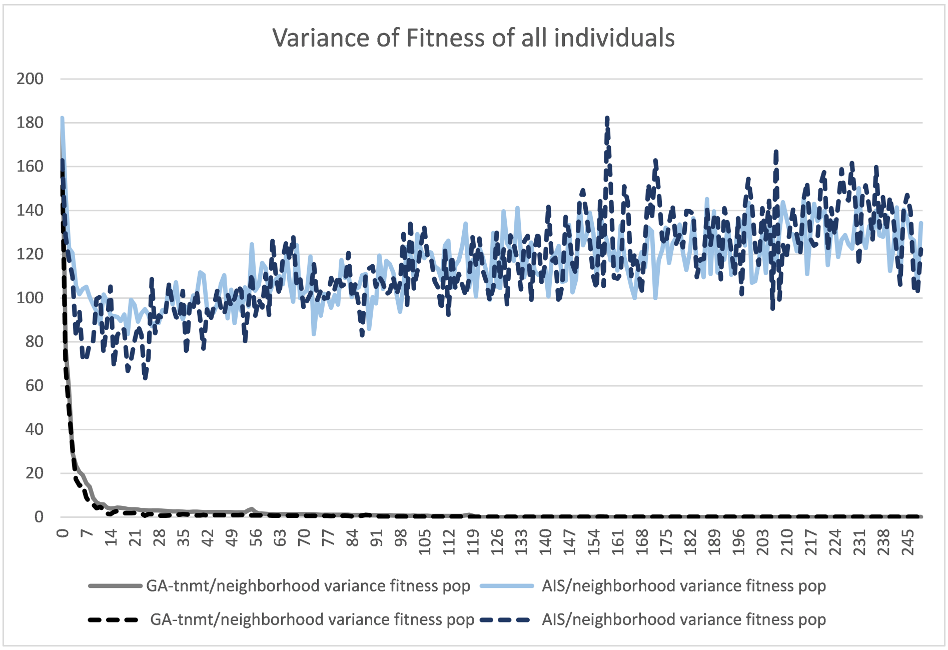

Figure 4: Variance of the fitness of all individual (Lower bound = 52).

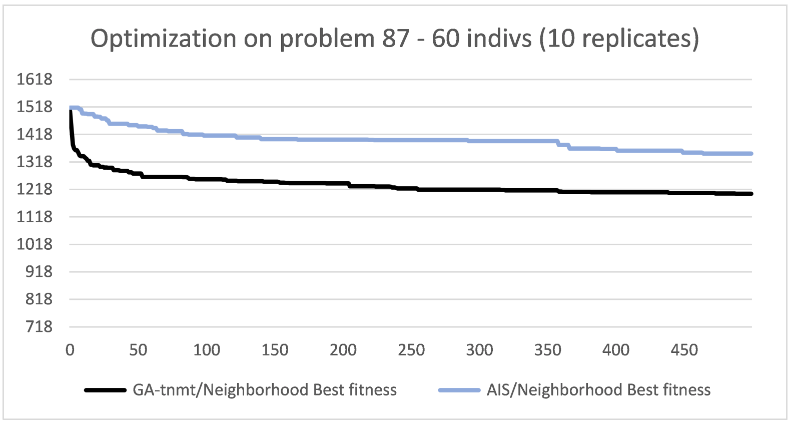

Figure 5: Performances of the two algorithms (Lower bound = 718).

Algorithm	Problem	NB Jobs	NB Machines	Pop. size	Fitness	LB
H/AIS/NH	86	6	6	50	68.8	52
H/GA-tnmt/NH	86	6	6	50	58	52
H/AIS/NH	86	6	6	100	66	52
H/GA-tnmt/NH	86	6	6	100	58.8	52
H/AIS/NH	20	20	15	50	2312	1213
H/GA-tnmt/NH	20	20	15	50	2195	1213
H/AIS/NH	20	20	15	100	1983	1213
H/GA-tnmt/NH	20	20	15	100	1706	1213

Table 1: Results of each algorithms after 250 generations (10 replicates).

6 Conclusions

The experiment confirms that the combinatorial nature of the JSSP makes it a highly complex problem. The results obtained during this project let believe that JSSPs become more tractable when approached using a combination of a global and a local search. EAs provide a good framework for solving JSSPs but the complexity of the problem is such that a canonical GA, a H-GA nor a H-AIS will not produce solutions close to the lower bound. It was observed that the GA-based approach tends to yields better results faster than the AIS-based algorithm. However, the AIS shows a better ability to maintain a diverse population. A tradeoff between speed of convergence and diversity of the population is noticed.

Essentially, it was observed that EA approaches are worth using on JSSPs of small complexity (less than 10 jobs and 10 machines) but does not scale to difficult problems (20 jobs and 15 machines). However it was observed that larger population tend to yield better results on these problems. More generally, the two algorithms appear to stagnate at obtaining better solutions after a relatively low number of generations (100 to 150).

The algorithms's tendency to remain stuck at a local optima that is relatively far from the lower bound suggests that hybrid exploration method can only lead thus far. The results obtained and the discussed inner workings of lead me to assert that even an hybrid approach alone is not enough to generate satisfying results on problems as hard as JSSPs. The exploration operations used were and the mutation, crossover and a neighborhood search returned poor results. I would suggest triggering an extremely diverse search methods to escape the current local optima.

6.1 Further Study

6.1.1 A Different Neightborhood Search

The method used for the neighborhood search in this project was aimed only at the best individual in the population, it would be worth exploring whether applying a local search on the entire population yiels better results.

6.1.2 Multiple Diverse Local Search Methods

It has been observed that approaching JSSPs by means of global and local search produced more interesting results. As observed in the literature, several well known priority rules have shown to produce better performances. Because of the permutative nature of JSSPs, it is likely that introducing multiple different priority rules within the main loop of a GA/neighborhood or AIS/neighborhood search algorithm would yield much better results. Doing so will undoubtedly result in a much longer search, therefore, using an extra coefficient to regulate the amount of inner searches is likely to be necessary.

References

[1] Runwei Cheng, Mitsuo Gen, and Yasuhiro Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms, part i: Representation. *Comput. Ind. Eng.*, 30(4):983–997, September 1996.

[2] Runwei Cheng, Mitsuo Gen, and Yasuhiro Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms, part ii: hybrid genetic search strategies. *Computers and Industrial Engineering*, 36(2):343–364, 1999.

[3] Randy L. Haupt and Sue Ellen Haupt. *Practical Genetic Algorithms*. John Wiley and Sons, Inc., New York, NY, USA, 1998.

[4] M. Negnevitsky. *Artificial Intelligence: A Guide to Intelligent Systems*. Addison Wesley/Pearson, 2011.