Assignment 1
Selection Constructive Hyper-Heuristics

Quinton Weenink, 13176545[1]

[1]Department of Computer Science – University of Pretoria
Pretoria, South Africa

1. Low-level construction heuristics

The following describes the implementation of the low-level heuristics used in this research.

1.1. First fit

Starting from the first bin, the first bin that is found to fit the item is chosen and the item is placed in that bin. If there is no bin that is found to fit the item an additional bin is created and the item is placed in that bin.

1.2. Last fit

Rather than starting at the begging of the bin list, last fit, starts at the end and finds the first bin that is found to fit the item. If no bin is found to fit the item a bin is created and the item is placed in that bin.

1.3. Next fit

Rather than searching from the beginning of the bin list the search is started at the last place an item was placed in. If no bin is found for it to be placed into a bin is created and the item is placed into it and the search position is placed at the first bin once again.

1.4. Best fit

Starting at the beginning of the bin list, best fit, attempts to find the first bin which, when the item is placed in it, leaves the least amount of space left over. If no bin is found to be able to contain the item a new bin is created and the item is placed inside it.

1.5. Worst fit

Rather than finding the best bin to place the item into, worst fit, intends to find the bin that leaves the most amount of space. If no bin can be found to contain the item a new bin is created and the item placed in it.

1.6. Random fit

This low-level heuristic was used as a control and comparison and was not used as part of the collection of heuristics.

Random fit randomly selects bins to attempt to place the item in. If the item fits it is placed in it. If no bin is found to contain the item a bin is created for the item to be placed into it.

2. Tabu search selection construction hyper-heuristic

A standard Tabu search algorithm was used where only solutions that improve on the solution as well as solutions that are not in the Tabu list are accepted as moves. The following move operators were used to navigate the search space:

- Add: A new random low-level heuristic is added at a random index in the current solution heuristic list.
- Change: A random low-level heuristic is selected from the current solution heuristic list and is randomly changed to an new randomly selected low-level heuristic.
- Remove: A heuristic at a random index of the current solution heuristic list is removed.
- Swap: Two randomly selected heuristics in the current solution heuristic list are swaped.

The above operators were selected randomly and applied to the current solution. The fitness of the final solution is used to determine weather the solution is an improvement over the previous solution.

The current position represents a collection of low-level heuristics to be applied to the problem instances

3. Genetic algorithm construction hyper-heuristic

The genetic algorithm used doesn't differ much from a standard Genetic algorithm except that each chromosome represents a collection of low-level heuristics to be applied to problem instances. The fitness of the resulting bin-packing problem is then measured for fitness. A population size of 25 was used for every test in this research. As well as a mutation rate of 50% and a crossover rate of 30% with the remaining 20% used for reproduction.

A tournament size of 3 was used in order to reduce selective pressure.

- Crossover: Two chromosomes are used to create two offspring by selecting random indexes in each of the chromosomes and allowing each of the offspring to have the section before the index of the one parent and the section after the index of the other parent.
- Mutate: Two randomly selected indexes of the chromosome are swaped to produce a mutated result.

4. Experimental setup

All initial solutions used both in Tabu search and the Genetic algorithm generated the initial solutions with a length between 5 and the length of the items for that particular problem. The heuristic solution list is not restricted in its length when any operators are applied to it, whether it be genetic or move operators.

For this research the stopping criteria of a iteration limit was used. 300 was chosen as the solutions appear to stagnate after that and little improvement can be seen.

In order to determine the fitness of the constructed solution the fitness function (1) is used. When multiple problem instances are used, such as in P_1, P_3 and P_5, the mean of all instances is calculated and that is used as the fitness of the solution.

$$f_{BPP} = \frac{\sum_{i=1}^{N} \left(\frac{F_i}{C}\right)^2}{N} \tag{1}$$

Table 1 lists the problem instance or a collection of problem instances on which the collection of heuristics will be applied. The resulting solution, resulting packed bins, is evaluated for its fitness using the above equation.

Table 1. Problems used to test the selective constructive hyper-heuristic.

Problem	***Data set***	***Instances***
P_1	1	$N1C1W1_F$
P_2	1	$N1C1W1_O, N1C1W1_P, N1C1W1_T,$ $N1C1W1_G, N1C1W1_K, N1C1W1_S$
P_3	2	$N1W2B2R0$
P_4	2	$N1W1B1R0, N1W1B1R1, N1W2B3R2$
P_5	3	$HARD6$
P_6	3	$HARD1, HARD3, HARD4$

5. Results

Below listed in tables 2, 3 and 4 are the results reported over 30 samples for stochastic problems. While those problems that are not stochastic were only ran once.

As can be observed a problem with multiple instances results in worse performance than that of a single instance. This might be due to any progress made on the one problem instance influences the performance in the other problem instances, making the problem more difficult to solve.

The low-level heuristics do not perform as well as the genetic algorithm and the Tabu search. The random search algorithm does perform better than one would expect but not as well as the Genetic algorithm and Tabu search.

Table 2. P_1 and P_2 from $dataSet_1$ after 300 generations. Means and standard deviations are reported over 30 samples.

		Generations		Performance		
	Algorithm	***50***	***150***	**mean**	**std**	**max**
P_1	Genetic algorithm	0.958487	0.961035	0.961947	0.005402	0.974372
	Tabu search	0.936785	0.947666	0.951998	0.010554	0.974241
	Random fit			0.890694	0.001344	0.894811
	First fit			0.902004		
	Last fit			0.887967		
	Next fit			0.921481		
	Best fit			0.902004		
	Worst fit			0.902004		
P_2	Genetic algorithm	0.934160	0.936783	0.937591	0.005883	0.952751
	Tabu search	0.917052	0.921795	0.927364	0.006512	0.945407
	Random fit			0.881086	0.004181	0.887771
	First fit			0.894473		
	Last fit			0.872514		
	Next fit			0.911979		
	Best fit			0.894473		
	Worst fit			0.894473		

Table 3. P_3 and P_4 from $dataSet_2$ after 300 generations. Means and standard deviations are reported over 30 samples.

		Generations		Performance		
	Algorithm	***50***	***150***	**mean**	**std**	**max**
P_3	Genetic algorithm	0.894263	0.898187	0.899240	0.012025	0.921320
	Tabu search	0.867903	0.871277	0.873578	0.006854	0.900746
	Random fit			0.853457	0.000172	0.853734
	First fit			0.853734		
	Last fit			0.853102		
	Next fit			0.893269		
	Best fit			0.853734		
	Worst fit			0.853734		
P_4	Genetic algorithm	0.830595	0.831182	0.831561	0.001848	0.834504
	Tabu search	0.822406	0.826900	0.827313	0.003537	0.831412
	Random fit			0.814017	0.011785	0.819341
	First fit			0.819321		
	Last fit			0.785687		
	Next fit			0.799966		
	Best fit			0.819321		
	Worst fit			0.819321		

Table 4. P_5 and P_6 from $dataSet_3$ after 300 generations. Means and standard deviations are reported over 30 samples.

		Generations		Performance		
	Algorithm	***50***	***150***	**mean**	**std**	**max**
P_5	Genetic algorithm	0.882853	0.882995	0.883069	0.000302	0.883907
	Tabu search	0.878595	0.879668	0.880004	0.001951	0.882631
	Random fit			0.866889	0.000009	0.866905
	First fit			0.866911		
	Last fit			0.866831		
	Next fit			0.875611		
	Best fit			0.866911		
	Worst fit			0.866911		
P_6	Genetic algorithm	0.866240	0.866544	0.866793	0.000838	0.867700
	Tabu search	0.862724	0.863854	0.865016	0.001366	0.867283
	Random fit			0.859169	0.000389	0.859609
	First fit			0.859619		
	Last fit			0.857875		
	Next fit			0.851034		
	Best fit			0.859619		
	Worst fit			0.859619		

6. Comparison

All figures represent the mean fitness after 300 generations of the genetic algorithm and 300 iterations of the tabu search algorithm.

As can be observed in Fig. (1, 2, 3, 4, 5, 6) using a genetic algorithm as a selection constructive hyper-heuristic almost always outperforms the local search algorithm for one-dimensional bin packing problems.

Figures (5) and (6) as well as their corresponding table 4 show how when the problem sets become more difficult and that the difference between Genetic algorithm and Tabu search becomes negligible.

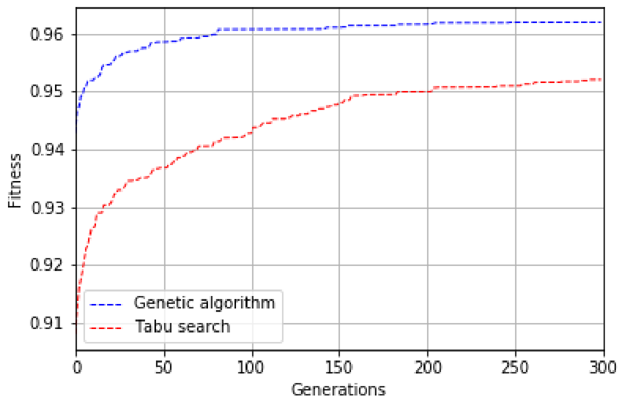

Figure 1. P_1 **mean fitness over 30 samples for 300 generations**

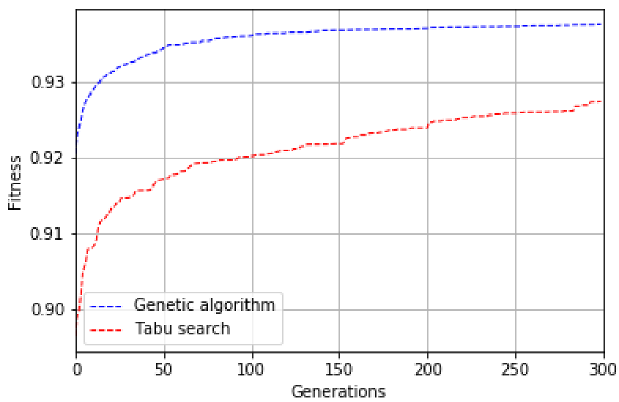

Figure 2. P_2 **mean fitness over 30 samples for 300 generations**

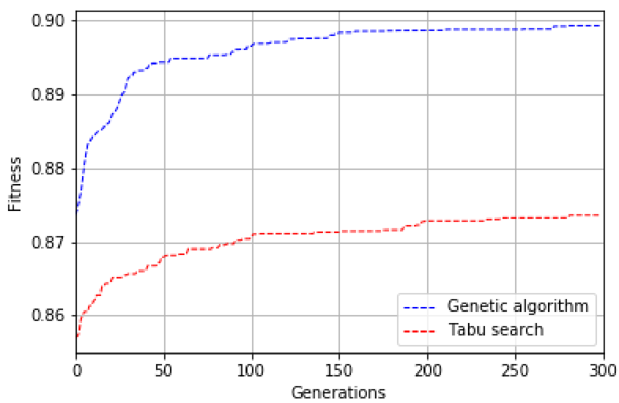

Figure 3. P_3 **mean fitness over 30 samples for 300 generations**

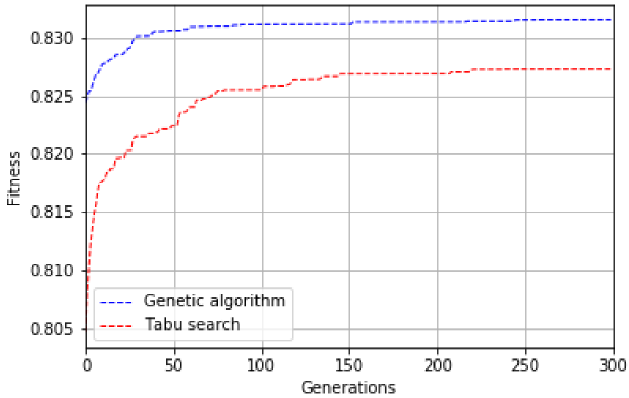

Figure 4. P_4 **mean fitness over 30 samples for 300 generations**

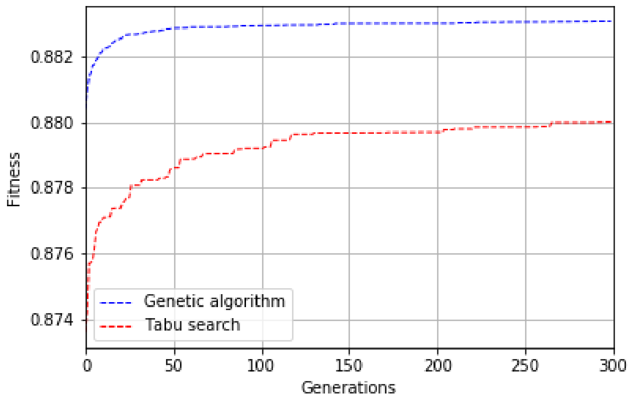

Figure 5. P_5 **mean fitness over 30 samples for 300 generations**

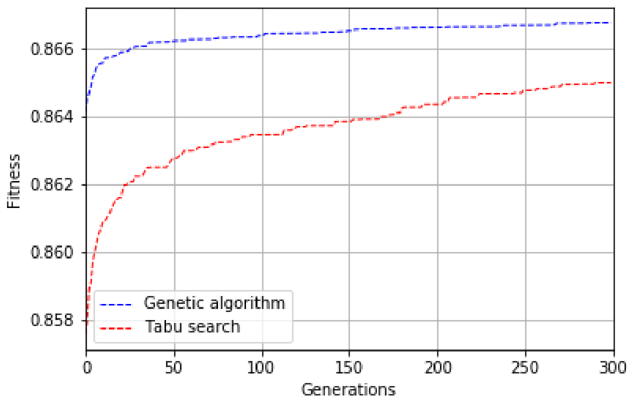

Figure 6. P_6 **mean fitness over 30 samples for 300 generations**