Heuristic Optimization : Assignment 2

Kenzo Clauw

Vrije Universiteit Brussel

kclauw@vub.ac.be.

Introduction

In the following sections, two algorithms will be presented to solve the permutation flowshop scheduling problem. The first algorithm is a rather simple, but highly efficient method based on Iterated Greedy Search[2] with partial solution improvement, while the latter is a complicated Hybrid Genetic Algorithm [3] using a population. According to the literature, the IGS is a highly efficient algorithm for the permutation flowshop problem and in some cases is actually better then most of the complex algorithms which take a lot of effort to implement. During the last sections of our paper, we will statistically test if our results verify the assumptions in the literature. Due to the lack of computational resources, we couldn't perform decent parameter tuning on our algorithms. The details on how to run the application in described in the README.txt file.

Initial Heuristic

In our implementation we chose a more advanced constructive heuristic based on the simple RZ heuristic from the previous assignment. The RZ heuristic sorts each job according to it's total weighted completion times and constructs a partial sequence by inserting each value in the direction of the lowest WCT. Each insertion of a job into a new partial sequence requires a full computation of the WCT and the greedy improvement of each partial sequence ignores the details of the underlying machines. The Liu and Reeves(LR) constructive heuristic tries to resolve these issues by using an index function that calculates the total machine idle time and the WCT of the remaining jobs when inserting job X into the sequence .

The Index Function

The index function for choosing a job i in the unscheduled sequence U to append to the scheduled sequence S with k jobs is defined as following :

$$f_{ik} = (n - k - 2)IT_{ik} + AT_{ik}$$

When a Tie occurs, the value with minimum IT_{ik} is selected.

- The weighted total machine idle time focuses on the effect of appending job k to the last value in the scheduled sequence.

  $$IT_k = \sum_{j=2}^{m} w_{jk} max\{C(i, j-1) - C([k], j), 0)\}$$

  with weights :

  $$w_{jk} = \frac{m}{(j + k(m - j))/(n - 2)}$$

  By using a weight on the total idle time, we can reduce the importance of jobs reaching the end of the sequence.

- The Total Artificial Flow Time determines the effect of inserting jobs x on the remaining jobs in S. By calculating the average processing time, we create an Artificial job p as following :

  $$t(p, j) = \sum_{q=U, q \neq i}^{m} t(q, j)/(n - k - 1)$$

  The Total Artificial Flow Time is then defined as the sum of the completion times of job p and i, as if they were scheduled after the last element in S :

  $$AT_{ik} = C(i, m) + C(p, m)$$

LR(x) Algorithm

The algorithm of our initial solution contains the following steps :

1. Sort the jobs according to ascending order of the index function. In case of a tie, pick the value with the lowest IT_{i0}
2. Use each of the first x ranked jobs as the first job in S and construct a sequence by selecting jobs one by one using the index function.
3. The sequence with the minimum $Csum$ is the final solution.

Iterated Local Greedy Search

The Iterated Local Greedy Search algorithm start from an initial solution generated by the LR(x) heuristic. In each iteration the algorithm introduces a new candidate solution through destruction, followed by a reconstruction of the partially destroyed sequence. We further improve the new solution through a local search algorithm. After each iteration, an acceptance criteria determines if the new solution is better then the current solution and with a certain probability accepts a worse step.

Overview of the algorithm

Algorithm 1 Iterated Greedy Search

1: **procedure** SEARCH(d,t)
2: $temperature \leftarrow averageProcessing() * t$
3: $\pi_0 \leftarrow LR(nbJobs)$
4: $\pi \leftarrow BestInsertionSearch(\pi_0)$
5: $\pi_* \leftarrow \pi$
6: ***Repeat*:**
7: $currentWCT \leftarrow computeWCT(\pi_)$
8: $\pi_{'} \leftarrow Destruction(d, \pi)$
9: $\pi_{''} \leftarrow ConstructionWithPartialImprovements(d, \pi)$
10: $nextWCT \leftarrow computeWCT(\pi_{''})$
11: **if** $currentWCT < nextWCT$ **then**
12: $\pi \leftarrow \pi''$
13: **if** $currentWCT <$ *computeWCT*(π_*) **then**
14: $\pi_* \leftarrow \pi''$
15: **else if** $rand() \leq e^{\frac{currentWCT - nextWCT}{temperature}}$ **then**
16: $\pi \leftarrow \pi''$

- **Destruction** During the Destruction phase, a partial sequence is constructed, by randomly removing jobs from the current solution. The parameter D, controls the number of jobs to be destroyed in the current sequence.
- **Construction** In the Construction Phase, a new solution is created by reinserting the removed jobs. Each of the partial sequences is evaluated and the sequence with the lowest WCT value is selected as the next solution.
- **Partial Improvement** Each partial solution in the construction phase is improved with local search, which according to [1] significantly improves the performance.
- **Local Search** We chose the Best Insertion Improvement as our local search algorithm because of the similarity's with our initial heuristic and the promising results during the last experiment.
- **Acceptance Criteria** The acceptance Criteria in our algorithm is based on Simulated Annealing, which compromises between exploration and exploitation by accepting a worse step in the search space with a certain probability.
- **Temperature** The temperature in the acceptance criteria is defined as the average processing time of each machine. Parameter t controls the exploration/exploitation trade off in the acceptance criteria.

$$temperature = \sum_{j=1}^{n} \sum_{i=1}^{m} \frac{processingtime(i,j)}{10mn} \times t$$

Algorithm 2 Destruction/Construction

1: **procedure** DESTRUCTION(d, π)
2: Initialize $removedJobsList$
3: **for** i = 1 to d **do**
4: $\pi_{'} \leftarrow$ remove a random job from π'
5: $removedJobsList \leftarrow$ add removed job to list
6: **procedure** CONSTRUCTION(d, π)
7: **for** i = 1 to d **do**
8: $\pi_{'} \leftarrow$ insert each job from the list to $\pi_{'}$
9: $\pi_{'} \leftarrow$ further improve with local search

parameters

IGS is controlled by the destruction parameter d and the amount of temperature t allowed in the acceptance criteria. When the temperature is high, we favor more exploration exploration. The destruction parameter determines how many genes in a given solution are destroyed. In the IGS our parameters are set as :
destruction = 2
temperature = 0.7

These parameters are the same as in the original paper. We set the amount of jobs to be removed from the solution rather low, this is because the partial updates of IGS favor minimal destruction.

Hybrid Genetic Algorithm

The inspiration behind Genetic Algorithms is based on the ideas of natural selection and survival of the fittest in evolution theory. In GA a population of individuals(candidate solution) is iteratively modified into a new generation of a solutions. In each iteration the algorithm randomly selects two parents from the current population and uses them to create new solutions.The individuals with the best fitness(completion time) have a bigger chance of transferring to the next generation. Eventually the population converges towards the optimal solution to the problem. In comparison to traditional algorithms, GA generate multiple solution and select the best one, the next population is determined non-deterministically. Our Hybrid Genetic Algorithm is a regular genetic algorithm but with an extra local search when generating the offspring population described as follow :

Algorithm 3 Hybrid Genetic Algorithm

```
1: procedure SEARCH(eliteProbability,mutationPprobability)
2:     Repeat:
3:     population ← initializethepopulation
4:     population ← evaluatethepopulation
5:     SelectEliteChromosomes(population)
6:     createArtificial(population, eliteWeights)
7:     offspring ← create offspring
8:     while i ¡ populationSize do
9:         father ← getrandomparentfrompopulation
10:        mother ← getrandomparentfrompopulation
11:        mutation(father, mutProbability)
12:        mutation(mother, mutProbability)
13:        localsearch(child1)
14:        localsearch(child2)
15:        add children to offspring
16:    population ← offspring
17:    population ← evaluatethepopulation
18:    SelectEliteChromosomes(population)
19:    createArtificial(population, eliteWeights)
```

- **Selection rule** The selection rules determines which solutions are selected as parents in the population
- **Crossover** The crossover combines multiple parents to create children for the next generation
- **Mutation** Through mutation we can apply random changes to the individual children

Initial Population

In most Genetic Algorithms, the initial condition of each solution in the population is generated at random. When generating solutions at random, the population gets filled with bad solution which in return get inherited by their offspring. To resolve this issue, we initialize one of our population members with the RZ(x) initial heuristic and the remaining solutions are generated at random.

Crossover Operator

The crossover operator extracts the good genes/jobs from a chromosome/solution in the population through a voting process.In our operator, a weighted mining gene structure is used to statistically sort and evaluate the best genes in the population.

Dominance Matrix A dominance matrix extracts the good genes in the chromosomes by counting the occurrences of each job in a specific position of the sequence. Because not every job is useful during the voting process, we add an extra weight to the voting process which is based on the best chromosomes in the population. Chromosomes with a higher fitness(best WCT) are superior and deserve a higher value when voting.

Artificial Chromosome Based on the values in the dominance matrix, we create an artificial chromosome by selecting the job/sequence pairs with the maximum number of occurrences. The Artificial Chromosome represents the good genes in the current population and is considered in the crossover operator. The voting process of the dominance matrix is defined as follows :

1. Generate the elite chromosomes in the current population by sorting the solutions according to their fitness value(Weighted Completion Time) in descending order.
2. Each position P_i in the sorted list represents the rank/weight of chromosome. The better the fitness(WCT), the higher the rank(position).
3. Build the dominance matrix $M_{i,j}$ by counting each job/sequence pair weighted with the rank of job P_i in the population. The amount of chromosomes that influence the voting process is determined by the probability parameter P_e.

Operator

As our crossover operator, we selected the SJOX operator which is based on a one-point crossover. In the SJOX operator common jobs between the parent and the child are transfered to the next generation, unlike the original operator we further improved the performance by integrating the artificial chromosomes in the common jobs selection. The common jobs(CJ) between the parents father/mother $CJ(FM)$,artificial chromosomesCJ_{AM},CJ_{AF} and their children are transferred to the next generation. By involving the artificial chromosomes in the crossover operator, we mine better local and global genes from the population.

1. Select both of the parents in the current population by generating random numbers from a distribution between 1 and the population size.
2. Search the common jobs CJ(FM),CJ(AF),CJ(AM) between the parents ,children and the artificial chromosomes.
3. Transfer the common jobs between the father /mother , artificial chromosomes and their children with the positions unchanged.
4. All the unassigned jobs before the crossover point from the father or mother chromosome are inherited by their son or daughter.
5. The remaining jobs after the crossover point are copied in the same order from their father or mother.

To illustrate this process, consider the artificial chromosome generated with sequence 2 1 4 3 5. The jobs $_\pi 1$ and $_\pi 5$

are selected as the parents with the current index equal to 2.

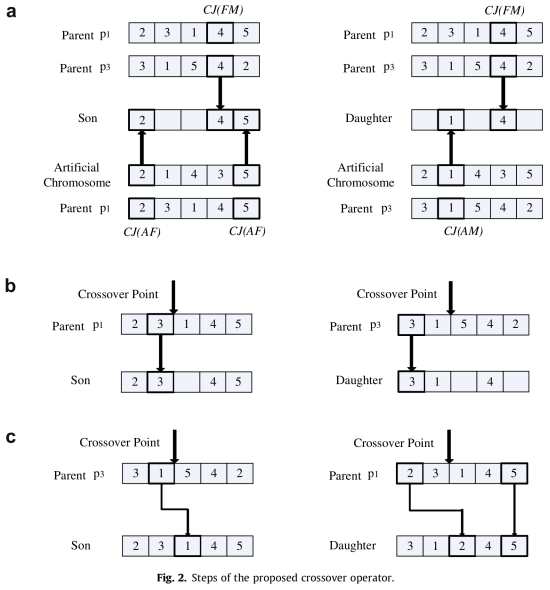

Fig. 2. Steps of the proposed crossover operator.

Mutation

The idea behind mutation is to destroy some of the genes in the parents thus favoring some exploration over exploitation in the search space. As in the originl paper, we selected the shift mutation as the our mutation procedure. The amount of mutation in each offspring population is controlled by the mutation probability parameter$_p m$. The shift mutation changes the solution by randomly moving the genes to the left or right.

Parameters In the HGA our parameters are set as :
Population size = 50
Mutation probability = 0.2
Mutation probability = 1.0
Number of elite chromosomes = 0.8

These parameters are the same as in the original paper. Setting the population size larger then the amount of jobs, is more then enough to reach a good performance in the genetic algorithm. The number of elite chromosomes is a critical parameter in the experiment, if the amount of elite chromosomes in the voting is to big, we might get stuck in a local minimum because there is no exploration. According to the original paper, it is better to allow maximum crossover, this allows all of the good genes to transfer to the offspring population.

Relative Percentage Deviation

To determine the relative average percentage deviation, we ran each algorithm per instance with a different seed in each run. When performing our experiments, each algorithm ran 5 times on each instance, with a different random seed in each run.

The RVD is described by the following formula :

$$RPD = \frac{(currentWCT - bestWCT)}{bestWCT}$$

The computational results containing the relative deviation and the best score per instance is gathered in tables 2 and 3.

On average the instances of size 100 seem to be better when using IGS, the smaller instances of 50 seems to be the same for both algorithms.

Instance	IGS	HGA
size 50	1.5017	1.6394
size 100	2.2392	2.6740

Table 1: Table : Average deviation over all instances

To determine if their is a significant difference between the algorithms, we use a wilcoxon test with significance level of 0.05 on both instance sizes.

Instance	P-value
size 50	0.003144
size 100	4.405e-12

Table 2: P-values of the wilcoxon test

In both cases the P-value is lower then the significance level, indicating that there is a difference between the algorithms. When looking at the RPD between each instance in table 2 and 3, we notice that there seems to be a difference between both algorithms.

		IGS		HGA	
Instance	Best	Score	Relative	Score	Relative
50.20.01	595260	604817	1.6056166	606596	1.90438
50.20.02	622342	633786	1.838892	634369	1.93254
50.20.03	592745	598579	0.9842338	603259	1.77378
50.20.04	666621	680633	2.101972	675357	1.31049
50.20.05	653748	664335	1.619432	669246	2.37064
50.20.06	643294	652260	1.3938582	654487	1.73995
50.20.07	565375	577953	2.22486	572328	1.2298
50.20.08	532097	538962	1.290293	542767	2.00527
50.20.09	572797	587688	2.59984	580930	1.41987
50.20.10	600785	611031	1.70547	608214	1.23655
50.20.11	604272	612880	1.424557	613976	1.6059
50.20.12	584199	589082	0.835914	592455	1.41322
50.20.13	551945	559845	1.431446	560796	1.6036
50.20.14	572954	581592	1.507766	583154	1.78025
50.20.15	595368	603471	1.361s006	603478	1.36218
50.20.16	551243	559152	1.4347582	556639	0.978879
50.20.17	647036	656540	1.468944	657096	1.55478
50.20.18	630186	638617	1.337858	643526	2.11684
50.20.19	465508	472940	1.59662	468937	0.736615
50.20.20	565240	574497	1.637818	574811	1.69326
50.20.21	617905	624631	1.0886464	628150	1.65802
50.20.22	642666	654828	1.8925234	661455	2.9236
50.20.23	602536	614251	1.944314	614267	1.94694
50.20.24	552363	559633	1.3161618	561614	1.6748
50.20.25	587942	595101	1.2176702	596650	1.4811
50.20.26	600296	609702	1.5669948	608066	1.29436
50.20.27	563816	570891	1.254841	575946	2.15141
50.20.28	549809	556696	1.2526892	559670	1.79353
50.20.29	545184	552747	1.387386	553209	1.47198
50.20.30	509210	512933	0.7311716	514392	1.01765

Table 3: Table : Results of instances size 50

		IGS		HGA	
Instance	Best	Score	Relative	Score	Relative
100.20.01	1792110	1828876	2.051546	1834852	2.385032
100.20.02	1810660	1855148	2.457048	1862590	2.868004
100.20.03	1679880	1717349	2.230472	1711569	1.88641
100.20.04	1943480	1996873	2.747268	1992785	2.536964
100.20.05	1539090	1573214	2.217182	1576607	2.437624
100.20.06	1660530	1697835	2.246548	1705505	2.708474
100.20.07	1783580	1807951	1.3664324	1821285	2.113996
100.20.08	1754470	1802171	2.71885	1807506	3.022886
100.20.09	1852180	1878240	1.40699	1880545	1.531416
100.20.10	1660070	1694671	2.08431	1702398	2.54975
100.20.11	1816300	1853766	2.062776	1867945	2.843428
100.20.12	1711620	1747209	2.079244	1747867	2.11769
100.20.13	1712830	1754940	2.458528	1762078	2.87523
100.20.14	1709350	1744997	2.08539	1741450	1.877922
100.20.15	1449140	1485246	2.491558	1494117	3.1037
100.20.16	1847320	1885732	2.079346	1898819	2.78778
100.20.17	1777650	1811691	1.914954	1826640	2.755862
100.20.18	1824010	1864720	2.231886	1862881	2.131096
100.20.19	1675710	1719969	2.641232	1723483	2.850924
100.20.20	1801770	1826955	1.750832	1850367	2.697172
100.20.21	1709120	1754495	2.654876	1773716	3.779478
100.20.22	1740800	1781071	2.31335	1791834	2.931628
100.20.23	1776600	1821845	2.546718	1828476	2.91996
100.20.24	1692610	1735009	2.504948	1747119	3.220386
100.20.25	1713800	1754480	2.373672	1762213	2.824868
100.20.26	1565920	1598629	2.08879	1606365	2.582826
100.20.27	1862160	1902744	2.179382	1917666	2.980742
100.20.28	1732590	1770727	2.201168	1782596	2.886222
100.20.29	1992230	2038563	2.325666	2049658	2.882608
100.20.30	1770450	1817637	2.665266	1825898	3.131882

Table 4: Table : Results of instances size 100

Correlation

To determine the correlation between the instances of the algorithms, we create a scatter plot of the data. The instances are represented by the dots while the X and Y values indicate the ARDP of IGS and the HGA. When looking at the scatter plot of the instances size 50, we notice a positive correlation but the results have high variance. Nevertheless, it seems like the ARDP of IGS is increasing with the HGA indicating that the results are correlated.

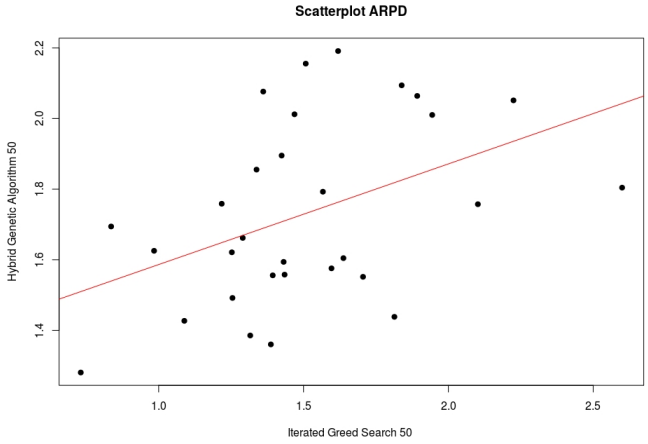

Table 5: scatter plot of instances size 50

The results of the instances in the scatterplot of the instances size 100 are more spread around the correlation line but they follow a negative correlation. A negative correlation might indicate that there is a difference in performance between the algorithms on both instances.

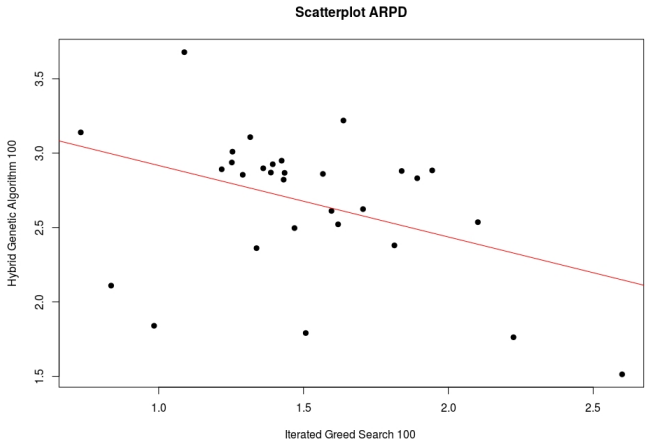

Table 6: scatter plot of instances size 100

Quality-runtime distribution

Before running the quality-runtime distribution, we determine the runtime needed for each algorithm on 5 selected instances to reach the best score.

Instance	IGS	HGA
50.20.01	33 sec	56 sec
50.20.02	32 sec	59 sec
50.20.03	29 sec	62 sec
50.20.04	39 sec	66 sec
50.20.05	47 sec	63 sec

Table 7: Runtime best scores

We then calculated the runtime distributions by running the algorithms on the 5 instances for 25 iterations with 10x the best time as a limit and a cutoff of 2 percent compared to the best solution score. The results of the run time scores of both algorithms are then sampled in a cumulative distribution and visualized through a plot. When looking the plot of the first instance, we notice that IGS seems to reach the best solution faster then HGA.

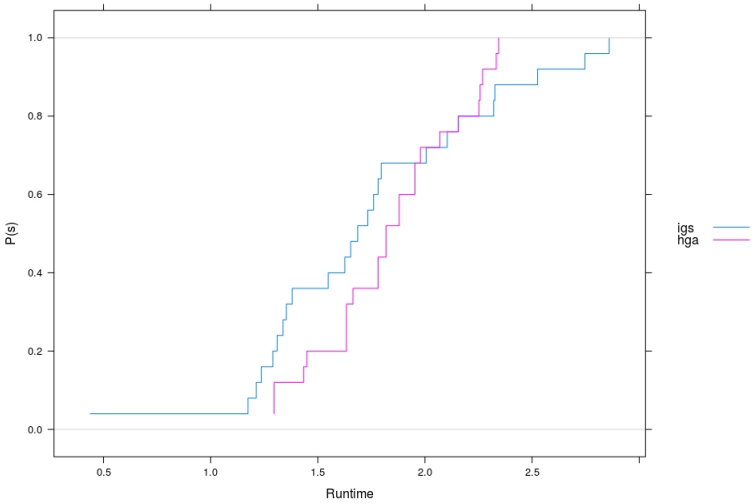

Table 8: Qualtity-runtime distribution on instance 50.20.1 with 2 percentage cutoff

In general we notice that IGS has a higher chance to reach the best solution over the 5 instances, except for instance 5. We conclude that IGS is alot faster then HGA and even gets better scores on instances of size 100. From these results, we learned that in practice when working with optimization, it is advised to start with a simple algorithm such as IGA and then implement a more complicated algorithm if the results are not satisfied.

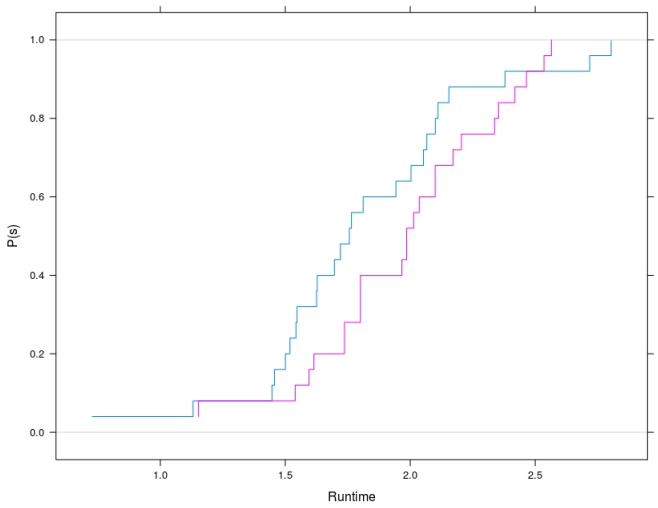

Table 9: Qualtity-runtime distribution on instance 50.20.2 with 2 percentage cutoff

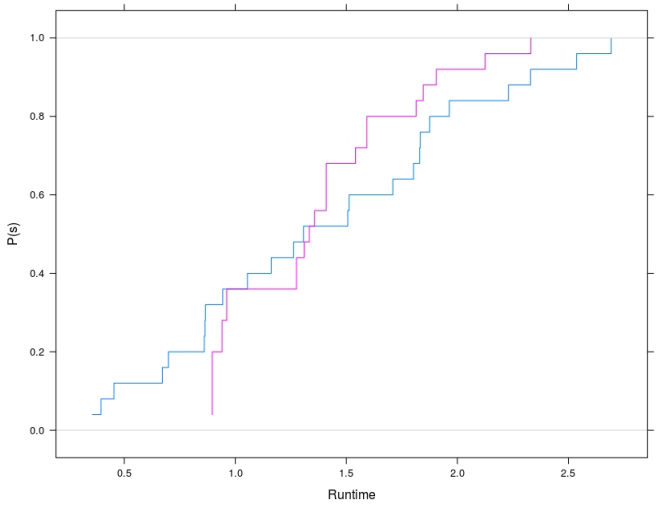

Table 10: Qualtity-runtime distribution on instance 50.20.3 with 2 percentage cutoff

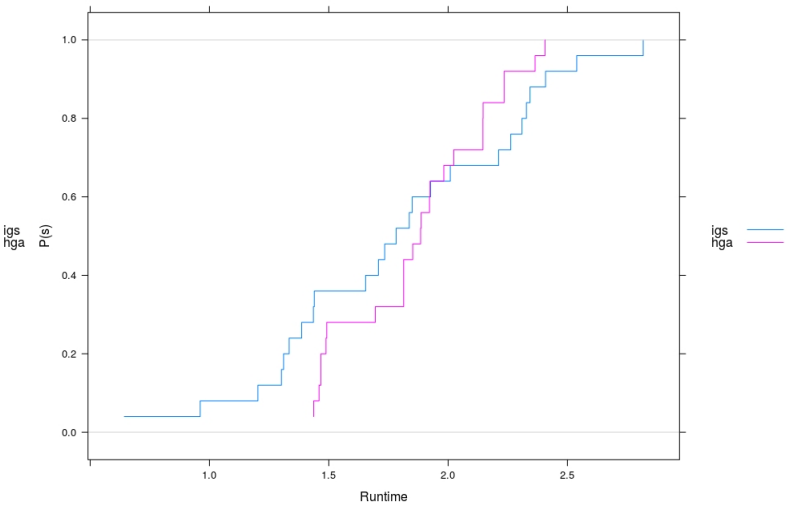

Table 11: Qualtity-runtime distribution on instance 50.20.4 with 2 percentage cutoff

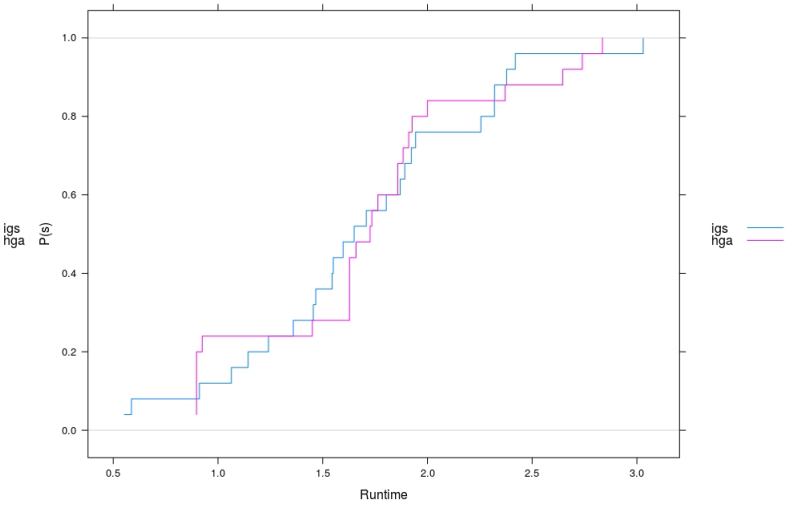

Table 12: Qualtity-runtime distribution on instance 50.20.5 with 2 percentage cutoff

References

J. Dubois-Lacoste, F. Pagnozzi, and T. Stützle. An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem. *Computers & OR*, 81:160–166, 2017.

Q.-K. Pan and R. Ruiz. Local search methods for the flowshop scheduling problem with flowtime minimization. *European Journal of Operational Research*, 222(1):31–43, 2012.

Y. Zhang, X. Li, and Q. Wang. Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. *European Journal of Operational Research*, 196(3):869–876, 2009.