[INFO-H-413] Heuristic Optimization - Assignment 2

Alexandre Heneffe

May 2021

1 Implementation

In this section, we will describe the two Stochastic Local Search algorithms implemented. First, we implemented the **"Memetic Genetic algorithm"** for the permutation flow-shop scheduling problem (PFSP) with weighted sum of completion times objective. Then, we implemented the **"Tabu search"** algorithm to solve the same problem. In this genetic algorithm, the fitness value is simply the Weighted completion sum of completion times. the lower the fitness, the better. (even though we would normally consider a fitness value as better if it is higher). We want to minimize the WCTs produced by the algorithm.

1.1 Memetic Genetic Algorithm (MGA)

The memetic genetic algorithm we implemented is mainly based on the framework proposed in [1]. The paper presents a novel GA for the permutation flowshop scheduling problem with total flowtime minimization. We define a chromosome as a solution permutation (set of jobs defining a solution, as for the iterative improvement)

1.1.1 Population initialization

The population is generated randomly, namely, given N chromosomes, each of them is generated using "Uniform random picking". If we look at the results of iterative improvement of the previous assignment, we see that random initial solution give better quality solutions in average (comparing statistically significant relative percentage deviations) compared to the simplified RZ heuristic.

1.1.2 Elite selection

In order to generate an artificial chromosome (described in the next subsection), we select a subset of the current population (at each iteration). That is, we only keep a certain percentage of "elite" chromosomes (chromosomes with lower fitness value, namely lower WCT) is selected.

1.1.3 Artificial chromosome

In order to generate good offspring chromosomes for the next generation, we use the "crossover" operator (that will be described in the next section). But first, create an "artificial chromosome" which use the information of the elite population described hereabove. The paper presents the "Weighted Simple Mining Gene Structure (WSMGS)" method that uses the statistic sorting of the elite population to create a new chromosome with good genes. This methods first performs a voting phase, where we count the occurrences of each job at each position over all the elite chromosomes, resulting in a "dominance matrix". In order to take the WCT into account, we use WCTs as weights and weight the occurrences accordingly. The goal is to find the job positions that give good quality solutions. Then, the artificial chromosome receives job that whose positions's occurences are maximized in the dominance matrix.

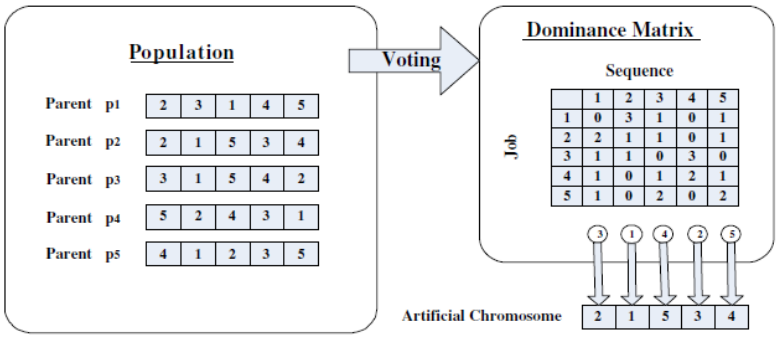


Figure 1: Procedure of simple mining gene structure - from [1]

1. Voting
   For each elite chromosome π in the population
   For each position j in π
   if ($\pi_{[j]}$ is job i) $M_{i,j}++$;
2. $count \leftarrow 1$;
3. while ($count \leqslant n$)
   3.1 Find the element $M^*_{i,j}$ with the maximum value in the "dominance matrix" (just select the first one if there is more than one such element);
   3.2 Add job i at position j in the artificial chromosome;
   3.3 Set all elements in row i and column j as -1;
   3.4 $count++$;

Figure 2: Artificial chromosome creation - from [1]

1.1.4 Selection

For the crossover operator (described hereunder), we need a mechanism to select parents. We use the rank roulette wheel selection mechanism. We first sort the chromosomes based on their WCTs, then we generate a random number (between 0 and 1) and the chromosome with high rank have more chance to be selected by this random number.

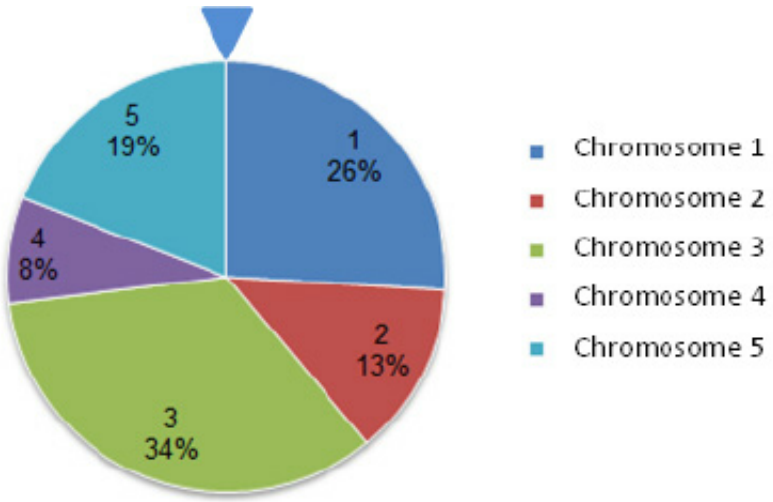


Figure 3: Roulette wheel selection mechanism

1.1.5 Crossover

We use the principle of "Common Job" (CJ), which is a job that is located in the same position in all considered chromosomes. The crossover operator works as follows : it generate a random crossover point, then, we pick 2 parents (father and mother, using the roulette wheel selection mechanism), then we search for Common jobs between father and mother (CJ(FM)), artificial chromosome and father (CJ(AF)), and artificial chromosome and mother (CJ(AM)). Then, we follow the transfers described in the following figures :

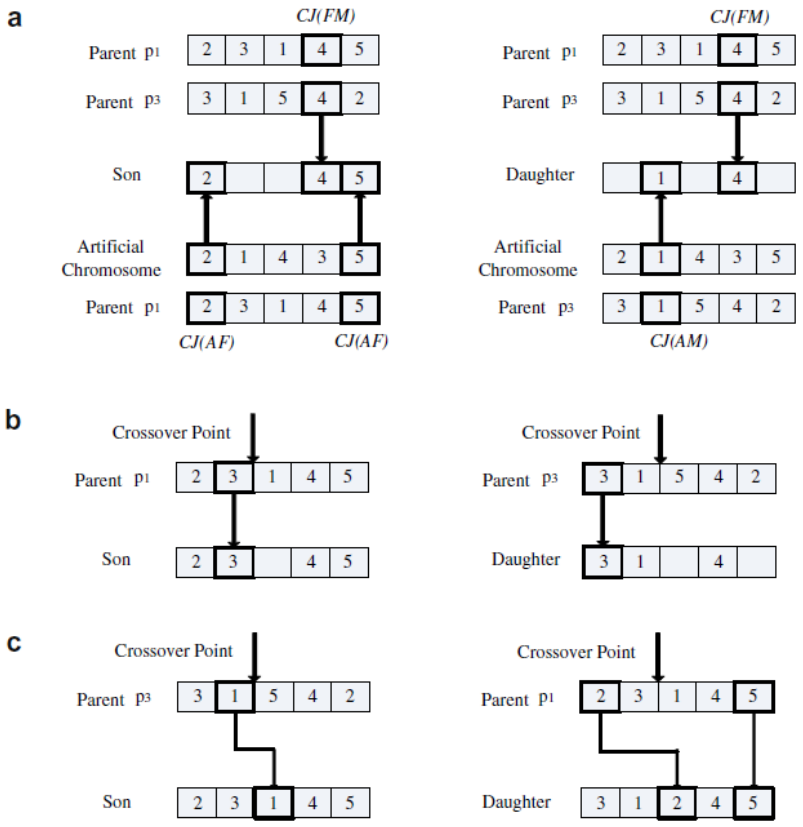


Figure 4: Crossover mechanism

After that, we obtain a son and a daughter. We apply a iterative improvement local search algorithm on both chromosomes in order to obtain local optimum solutions. We choose the **first improvement - insert** algorithm from previous assignment since it gave the best quality solutions compared to Best improvement and Transpose or Exchange neighbourhood. However, FI is slower then the other iterative improvement algorithms. Here, the initial solution given to the II is already defined, so we do not have to choose between random and Simplified RZ heuristic.

The whole process (from parents selection to local searchs) is repeated until N chromosomes are produced (N = population size).

1.1.6 Population update

After the reproduction phase (artificial chromosome generation, crossover operator, children local searchs), we update the population for the next generation. We choose to simply replace the current population by the new population composed of the same number of chromosomes produced by the reproduction phase.

The whole genetic algorithm framework can be summed by the following pseudocode (provided by [1]) :

```
{
    count ← 1;
    population ←PopulationInitialization(S);
    PopulationEvaluation(population);
    TFT_min ← MinTFT(population);
    π* ← MinTFTPermutation(population);
    E ← EliteSelection(P_e, population);
    π# ← WSMGS(E);
    while(count ⩽ COUNT AND within CPU time limitation)
    {
        Create offspringPopulation without any chromosome
        Repeat until covering the entire population
        {
            father←RandomSelect(population);
            mother←RandomSelect(population);
            son←Crossover(father, mother, π#, P_c);
            daughter ←Crossover(mother, father, P_c);
            Mutation(son, P_m);
            Mutation(daughter, P_m);
            //local search to improve both son and daughter
            RZ_FPE(son);
            RZ_FPE(daughter);
            Add(son,offspringPopulation);
            Add(daughter,offspringPopulation);
        }
        //population updated by generational schema in Section
        3.5
        population
        ←PopulationUpdate(population,offspringPopulation);
        PopulationEvaluation(population);
        E ←EliteSelection(P_e,population);
        π# ← WSMGS(E);
        if (MinTFT(population) < TFT_min)
        {
            TFT_min ← MinTFT(population);
            π* ← MinTFTPermutation(population);
            count ← 1;
        }
        else
            count + +;
    }
    return TFT_min;}
```

Figure 5: Whole hybrid genetic algorithm framework

The termination condition is simply a predefined time limit (depending of the number of jobs considered).

1.2 Tabu search

The Tabu search implemented is based on 3 different papers : [2], [3] and [4]. The general algorithm works as follows :

- Initialize the solution
- Find a neighbour that has not the tabu status and that gives the best WCT (and if the neighbour is tabu but it satisfies a certain aspiration criterion, then it is also accepted)
- Update the tabu list
- Terminates when the stopping criteria is satisfied

1.2.1 Initialisation

To generate the initial solution, we simply use "Uniform random picking". We saw in the previous assignment that random initial solutions lead to better quality solutions than Simplified RZ Heuristic. Additionally, we want to avoid getting trapped in a local optimum during the search.

1.2.2 Tabu list and neighbourhood

The method we used is common in [2] and [3]. We define a move by a pair of positions. The operator used to produce such moves is "Insert" (I-move). [2] states that Insert is better than Exchange (E-move). In the previous assignment, we also concluded that Insert neighbourhood is better than Exchange neighbourhood (and transpose) since it gave better quality solutions (but it is slower in terms of computation run-time). Given positions $a, b \in 0...n, a \neq b$ where n is the number of jobs, we can perform 2 different insertion depending on a and b :

- if $a < b$: $\pi_v = (\pi(1)...\pi(a-1), \pi(a+1), ...\pi(b), \pi(a), \pi(b+1), ...\pi(n))$
- if $a > b$: $\pi_v = (\pi(1)...\pi(b-1), \pi(a), \pi(b), ...\pi(a-1), \pi(a+1), ...\pi(n))$

where π is a job permutation.
Then, we define a tabu pair as a pair of consecutive jobs based on a move (a, b):

- if $a < b$: $\pi(a), \pi(a+1)$
- if $a > b$: $\pi(a-1), \pi(a)$

When considering a new move (a, b), we define it as tabu if one of those pairs belongs to the tabu list :

- if $a < b$: $(\pi(j), \pi(a)), j = a+1, .., .b$
- if $a > b$: $(\pi(a), \pi(j)), j = b, ..., a-1$

1.2.3 Diversification

In order to avoid getting trapped in a local optimum during the search, we introduce a diversification mechanism. When no better solution (that is not tabu and that satisfies the aspiration criterion) is found after considering all the neighbourhood of a given candidate solution, then we randomly pick one worsering solution among all neighbours.

1.2.4 Aspiration criterion

When a solution is considered as non-tabu, we might also consider an aspiration criterion. Namely, if the candidate neighbour solution is better than the overall best solution found, then it is accepted.

2 Parameters

2.1 SLS algorithms Parameters

2.1.1 Memetic Genetic Algorithm

- Probability of mutation (**Pm**) : **0.5**. We want the algorithm to avoid getting trapped in a local optimum, so a relatively big mutation probability introduces diversification.
- Probability of crossover (**Pc**) : **1.0**. The crossover is done at every generation so that a we exploit intensification. It is important for the parents to pass all its good genes to its offspring. All parents should participate in this transfer.
- Proportion of elite (**Pe**) : **0.4**. The paper tests 5 different values : 0.2, 0.4, 0.5, 0.6, 0.8. 0.6 was their optimal value, but in our case, the different of WCT between two chromosome is significant, thus we select less elite chromosomes to form the elite population. We only keep the 40% best ones.
- **COUNT** : **1000**. This parameter is not used for the benchmarks but it can be used as secondary termination criterion when running a single MGA. It defines the maximum number of steps where the algorithm keeps the same best solution.
- **Population size** : 20. Since we are limited by the maximum run-time, we keep a small population so that we have a tradeoff between population size and number of generation. With smaller population size, more generation are performed, and thus possibly better quality results are generated in the end.
- **Max time** : **350.0** seconds.

2.1.2 Tabu Search

- Tabu **tenure** : **7**. We want to have a relatively small tabu list size but not too small otherwise we would reject to much moves during the search. In [4], they choose 7 as tabu tenure as it is the traditional value used in the literature. They also state that a variable tabu list length can be implemented but do not give significantly better results.
- **Max time** : **150.0** seconds.

2.1.3 Time limit

For both MGA and TS, we have to define a maximum run-time limit. Since both SLS algorithms use the same termination criterion, namely the run-time, we will use the same maximum values for both of them. In the previous assignment, the Variable Neighbourhood Descent local search took in average (over the 4 algorithms : RTEIn RTIE, STEI, STIE)

- 0.08 seconds for 50 jobs
- 1.69 seconds for 100 jobs

instance	RTEI	RTIE	STEI	STIE
50	0.117659	0.146266	0.052422	0.024498
100	2.133100	3.434033	0.768017	0.446567

Figure 6: VND - average computation times

It is required to multiply those values by 500 to obtain the maximum run-time limit for the SLS algorithms. We get :

- 40 seconds for 50 jobs
- 847 seconds for 100 jobs

Since 40 seconds is not enough for the SLS algorithms in order to generate sufficiently good quality results, and 847 seconds is too long, we decided to use these max run-times :

- **150 seconds** for 50 jobs
- **350 seconds** for 100 jobs

3 Exercise 2.1

Some precisions about the benchmarks :

1. We make sure to have different seeds for each run (we run each SLS algorithm 5 times on each instance), so we have 1 different seed per run, but the same seed for the 2 SLS algorithms per instance (for convenience).
2. All the experiments were run on the "CECI" cluster (http://www.ceci-hpc.be/faq.html) and the code is written in C++.
3. We use the following abbreviations to describe the algorithms :
   - GA = Memetic Genetic Algorithm
   - TS = Tabu Search
   - GA{i} = Run i of Genetic algorithm on all instances
   - TS{i} = Run i of Tabu Search on all instances
   - BK = Best known solution for a given instance
   - GA{j} = target quality solution with j% for Genetic Algorithm (j = {0.1,0.5,1,2}% above BK)
   - TS{j} = target quality solution with j% for Tabu Search (j = {0.1,0.5,1,2}% above BK)

   For exercise 2.1.5, we will consider solutions close to best known solutions for some instances. Hereunder, we have all the best solutions depending on the percentage of quality solution above the best known solutions. For instance, 0.1% means that we add this percentage of the best known to the best known. We will only consider the 2 first instances, namely 50_20_01 and 50_20_02.

instances	best	0.1	0.5	1	2
50_20_01	595260	595855	598236	601212	607165
50_20_02	622342	622964	625453	622342	634788
50_20_03	592745	593337	595708	598672	604599
50_20_04	666621	667287	669954	673287	679953
50_20_05	653748	654401	657016	660285	666822

Figure 7: best known solution and its closest best solutions, {0.1,0.5,1,2}% above

3.1 All Weighted sum of completion times (WCT) over 5 runs

instance	GA1	GA2	GA3	GA4	GA5	TS1	TS2	TS3	TS4	TS5
100_20_01	1802774	1803106	1804606	1803407	1801413	1811855	1810396	1815365	1809119	1812728
100_20_02	1817916	1820232	1823489	1821005	1823016	1836185	1850508	1829273	1833964	1838673
100_20_03	1684602	1685280	1683920	1688245	1689325	1706461	1691036	1694770	1691144	1699254
100_20_04	1957459	1958627	1963533	1960739	1953636	1970735	1969563	1983796	1980861	1975087
100_20_05	1546766	1545754	1548783	1543087	1551474	1568881	1554838	1563517	1561351	1561220
100_20_06	1677456	1676440	1669071	1667231	1670506	1691829	1681918	1680975	1681191	1688261
100_20_07	1784534	1782716	1783978	1784041	1784344	1785609	1793219	1800415	1793648	1806952
100_20_08	1767459	1768061	1766077	1770186	1771679	1784430	1783485	1765894	1777957	1786976
100_20_09	1856230	1855598	1850173	1854943	1856714	1875540	1870474	1865322	1890559	1882883
100_20_10	1670144	1673436	1672662	1670670	1674466	1690359	1706547	1682044	1682888	1691628
100_20_11	1827462	1828170	1828189	1828948	1832218	1834521	1831432	1841699	1848548	1831865
100_20_12	1712989	1711176	1713406	1715821	1709507	1777982	1732056	1724434	1741782	1748437
100_20_13	1729619	1729080	1723649	1729863	1723619	1734822	1731747	1726717	1738724	1723947
100_20_14	1719543	1717852	1718069	1723409	1719023	1725714	1744792	1732306	1733586	1723744
100_20_15	1458123	1453493	1459578	1450680	1458716	1462195	1467377	1474654	1456079	1474143
100_20_16	1858434	1860687	1864338	1858496	1864576	1874309	1874017	1873212	1868029	1863086
100_20_17	1780666	1785479	1787628	1784259	1776320	1794016	1805431	1801968	1803984	1811397
100_20_18	1829183	1831154	1831366	1827596	1837272	1845455	1849487	1852606	1856805	1846814
100_20_19	1683245	1687906	1683909	1682112	1683911	1707893	1704788	1694047	1697818	1702537
100_20_20	1811391	1814501	1810331	1812747	1813586	1818771	1820536	1809999	1815705	1819160
100_20_21	1724994	1727649	1725194	1725477	1723249	1738183	1726282	1731361	1727914	1730157
100_20_22	1751736	1747569	1744629	1750409	1745017	1758036	1782147	1771409	1776524	1770787
100_20_23	1788700	1786920	1791750	1783195	1787394	1814473	1817102	1808755	1835677	1807014
100_20_24	1707822	1712481	1702468	1705105	1707289	1727501	1729343	1714389	1734897	1714985
100_20_25	1724921	1725958	1729140	1725695	1727699	1742801	1744095	1762911	1759834	1750481
100_20_26	1568138	1573049	1574034	1575573	1577479	1606004	1587093	1578008	1590847	1587099
100_20_27	1867481	1876009	1878467	1879392	1876453	1877625	1895496	1902806	1892735	1919492
100_20_28	1746214	1747966	1743067	1743780	1743030	1755646	1750027	1757957	1753906	1757568
100_20_29	2010704	2001733	2006072	2009585	2010762	2032817	2021688	2024634	2029023	2020656
100_20_30	1777965	1772446	1775116	1781662	1776817	1800385	1792675	1813574	1795843	1802818

Figure 8: SLS - WCT - 100 jobs - left 5 = GA, right 5 = TS

50_20_01	596204	595943	596259	596761	596138	598590	596894	599440	595754	597552
50_20_02	623274	622822	622342	622444	622822	623287	627620	625081	633704	623564
50_20_03	592635	591316	592830	593413	591389	592525	593412	594808	593187	594788
50_20_04	666980	666916	667374	667222	667170	669646	666724	669400	667757	667283
50_20_05	654328	656953	655288	654759	653837	657042	655835	656906	660082	659107
50_20_06	643304	644241	643977	643540	644046	645710	646298	645003	645724	646085
50_20_07	565391	565375	565810	565379	565739	569684	570628	570101	569147	570579
50_20_08	532409	529836	531014	533216	531535	532965	531726	533291	535916	536957
50_20_09	571670	572143	571264	572528	572246	573692	571043	575074	575376	572434
50_20_10	601060	601223	601017	600906	601070	602588	602061	603114	601055	601985
50_20_11	604319	604941	605285	604398	604140	607218	605745	605603	605315	605862
50_20_12	584146	584384	584238	583673	583938	585736	584818	585377	587258	584679
50_20_13	551357	551379	552533	551317	552584	553685	554005	552375	552018	554000
50_20_14	573120	574566	573573	573024	573573	576370	577164	575079	574313	578242
50_20_15	595449	596109	595146	595888	595904	596652	597516	595920	598716	597039
50_20_16	551703	551363	551858	551106	551636	552623	551495	551862	552781	556281
50_20_17	646402	646975	646006	645882	645270	648645	646697	653087	653508	649861
50_20_18	631165	630934	630934	631128	630948	630929	631816	633197	631903	632446
50_20_19	465614	465614	466083	465614	465732	468096	465690	465614	465814	466382
50_20_20	565944	565395	566010	566162	565689	565622	566448	566680	566421	567331
50_20_21	618606	618054	618144	618156	619218	619771	621212	617842	622573	621249
50_20_22	643210	643125	643703	643105	642539	644510	653560	648208	649036	646294
50_20_23	603131	602728	602639	602994	602481	603917	604420	605407	604276	603697
50_20_24	553004	553228	553633	553569	553015	553105	554276	555123	554801	555092
50_20_25	589136	589322	587440	589639	588820	593854	588249	588429	591189	589168
50_20_26	601765	601964	602452	601197	601839	603986	602816	602956	603077	602082
50_20_27	563212	563953	564361	563840	563606	563681	564608	564228	563183	563826
50_20_28	550096	550229	549908	550101	550020	550929	551629	551814	550110	552396
50_20_29	545288	545288	545622	545630	545300	545894	545157	549232	549049	546872
50_20_30	509219	510224	509445	510250	509395	510284	509760	510365	510009	510526

Figure 9: SLS - WCT - 50 jobs - left 5 = GA, right 5 = TS

3.2 All relative percentage deviations (RPD) over 5 runs across all instances

instance	GA1	GA2	GA3	GA4	GA5	TS1	TS2	TS3	TS4	TS5
100_20_01	0.595053	0.613578	0.697279	0.630374	0.519109	1.101.774	1.020.361	1.297.632	0.949105	1.150.487
100_20_02	0.400738	0.528647	0.708526	0.571339	0.682403	1.409.707	2.200.744	1.027.968	1.287.045	1.547.115
100_20_03	0.281092	0.321452	0.240493	0.497952	0.562243	1.582.315	0.664095	0.886373	0.670524	1.153.297
100_20_04	0.719277	0.779375	1.031.809	0.888046	0.522568	1.402.381	1.342.077	2.074.423	1.923.405	1.626.310
100_20_05	0.498736	0.432983	0.629788	0.259699	0.804631	1.935.624	1.023.202	1.587.107	1.446.374	1.437.863
100_20_06	1.019.313	0.958128	0.514354	0.403546	0.600772	1.884.880	1.288.022	1.231.233	1.244.241	1.670.009
100_20_07	0.053488	-0.048442	0.022315	0.025847	0.042835	0.113760	0.540430	0.943888	0.564483	1.310.398
100_20_08	0.740338	0.774650	0.661567	0.895769	0.980866	1.707.638	1.653.776	0.651137	1.338.695	1.852.753
100_20_09	0.218661	0.184539	-0.108359	0.149176	0.244793	1.261.217	0.987701	0.709542	2.072.099	1.657.668
100_20_10	0.606842	0.805147	0.758522	0.638527	0.867192	1.824.562	2.799.701	1.323.679	1.374.520	1.901.004
100_20_11	0.614546	0.653526	0.654572	0.696361	0.876397	1.003.193	0.833122	1.398.392	1.775.478	0.856962
100_20_12	0.079983	-0.025940	0.104346	0.245440	-0.123450	3.877.146	1.193.957	0.748647	1.762.190	2.151.003
100_20_13	0.980191	0.948722	0.631645	0.994436	0.629893	1.283.957	1.104.430	0.810763	1.511.767	0.649043
100_20_14	0.596309	0.497382	0.510077	0.822476	0.565888	0.957323	2.073.420	1.342.967	1.417.849	0.842074
100_20_15	0.619885	0.300385	0.720289	0.106270	0.660806	0.900879	1.258.471	1.760.630	0.478836	1.725.368
100_20_16	0.601628	0.723589	0.921226	0.604985	0.934110	1.460.981	1.445.175	1.401.598	1.121.029	0.853453
100_20_17	0.169662	0.440413	0.561303	0.371783	-0.074818	0.920654	1.562.794	1.367.986	1.481.394	1.898.405
100_20_18	0.283606	0.391665	0.403287	0.196600	0.727079	1.175.706	1.396.758	1.567.755	1.797.962	1.250.212
100_20_19	0.449660	0.727811	0.489285	0.382047	0.489404	1.920.559	1.735.264	1.094.282	1.319.321	1.600.933
100_20_20	0.533975	0.706583	0.475144	0.609234	0.655800	0.943572	1.041.531	0.456718	0.773406	0.965162
100_20_21	0.928782	1.084.125	0.940484	0.957042	0.826683	1.700.466	1.004.142	1.301.313	1.099.630	1.230.867
100_20_22	0.628217	0.388844	0.219956	0.551988	0.242245	0.990119	2.375.172	1.758.330	2.052.160	1.722.599
100_20_23	0.681076	0.580885	0.852752	0.371215	0.607565	2.131.769	2.279.748	1.809.918	3.325.284	1.711.922
100_20_24	0.898730	1.173.986	0.582414	0.738209	0.867241	2.061.373	2.170.199	1.286.711	2.498.331	1.321.923
100_20_25	0.648909	0.709418	0.895087	0.694072	0.811005	1.692.204	1.767.709	2.865.620	2.686.078	2.140.331
100_20_26	0.141642	0.455260	0.518162	0.616443	0.738160	2.559.773	1.352.112	0.771942	1.591.844	1.352.496
100_20_27	0.285743	0.743706	0.875703	0.925377	0.767550	0.830487	1.790.179	2.182.734	1.641.910	3.078.790
100_20_28	0.786337	0.887458	0.604702	0.645854	0.602566	1.330.725	1.006.412	1.464.109	1.230.297	1.441.657
100_20_29	0.927303	0.477003	0.694799	0.871134	0.930214	2.037.265	1.478.645	1.626.519	1.846.825	1.426.843
100_20_30	0.424468	0.112740	0.263549	0.633285	0.359626	1.690.813	1.255.331	2.435.765	1.434.268	1.828.236

Figure 10: SLS - RPD - 100 jobs - left 5 = GA, right 5 = TS

50_20_01	0.158586	0.114740	0.167826	0.252159	0.147499	0.559419	0.274502	0.702214	0.082989	0.385042
50_20_02	0.149757	0.077128	0.000000	0.016390	0.077128	0.151846	0.848087	0.440112	1.825.684	0.196355
50_20_03	-0.018558	-0.241082	0.014340	0.112696	-0.228766	-0.037115	0.112527	0.348042	0.074568	0.344668
50_20_04	0.053854	0.044253	0.112958	0.090156	0.082356	0.453781	0.015451	0.416879	0.170412	0.099307
50_20_05	0.088719	0.490250	0.235565	0.154647	0.013614	0.503864	0.319236	0.483061	0.968875	0.819735
50_20_06	0.001554	0.147211	0.106172	0.038241	0.116898	0.375567	0.466972	0.265664	0.377743	0.433861
50_20_07	0.002830	0.000000	0.076940	0.000707	0.064382	0.762149	0.929118	0.835905	0.667168	0.920451
50_20_08	0.058636	-0.424923	-0.203534	0.210300	-0.105620	0.163128	-0.069724	0.224395	0.717726	0.913367
50_20_09	-0.196754	-0.114177	-0.267634	-0.046963	-0.096195	0.156251	-0.306217	0.397523	0.450247	-0.063373
50_20_10	0.045773	0.072905	0.038616	0.020140	0.047438	0.300107	0.212389	0.387659	0.044941	0.199739
50_20_11	0.007778	0.110712	0.167640	0.020852	-0.021844	0.487529	0.243764	0.220265	0.172604	0.263127
50_20_12	-0.009072	0.031667	0.006676	-0.090038	-0.044677	0.263095	0.105957	0.201644	0.523623	0.082164
50_20_13	-0.106532	-0.102546	0.106532	-0.113779	0.115772	0.315249	0.373226	0.077906	0.013226	0.372320
50_20_14	0.028973	0.281349	0.108037	0.012217	0.108037	0.596208	0.734788	0.370885	0.237192	0.922936
50_20_15	0.013605	0.124461	-0.037288	0.087341	0.090028	0.215665	0.360785	0.092716	0.562341	0.280667
50_20_16	0.083448	0.021769	0.111566	-0.024853	0.071293	0.250343	0.045715	0.112292	0.279006	0.913935
50_20_17	-0.097985	-0.009428	-0.159187	-0.178352	-0.272937	0.248672	-0.052393	0.935188	1.000.253	0.436606
50_20_18	0.155351	0.118695	0.118695	0.149480	0.120917	0.117902	0.258654	0.477795	0.272459	0.358624
50_20_19	0.022771	0.022771	0.123521	0.022771	0.048119	0.555952	0.039097	0.022771	0.065735	0.187752
50_20_20	0.124549	0.027422	0.136225	0.163117	0.079435	0.067582	0.213715	0.254759	0.208938	0.369931
50_20_21	0.113448	0.024114	0.038679	0.040621	0.212492	0.301988	0.535196	-0.010196	0.755456	0.541184
50_20_22	0.084647	0.071421	0.161359	0.068309	-0.019761	0.286930	1.695.126	0.862345	0.991184	0.564523
50_20_23	0.098749	0.031865	0.017094	0.076012	-0.009128	0.229198	0.312678	0.476486	0.288779	0.192686
50_20_24	0.116047	0.156600	0.229921	0.218335	0.118038	0.134332	0.346330	0.499671	0.441376	0.494059
50_20_25	0.203081	0.234717	-0.085383	0.288634	0.149334	1.005.541	0.052216	0.082831	0.552265	0.208524
50_20_26	0.244713	0.277863	0.359156	0.150093	0.257040	0.614697	0.419793	0.443115	0.463271	0.297520
50_20_27	-0.107127	0.024299	0.096663	0.004257	-0.037246	-0.023944	0.140471	0.073073	-0.112271	0.001774
50_20_28	0.052200	0.076390	0.018006	0.053109	0.038377	0.203707	0.331024	0.364672	0.054746	0.470527
50_20_29	0.019076	0.019076	0.080340	0.081807	0.021277	0.130231	-0.004952	0.742502	0.708935	0.309620
50_20_30	0.001767	0.199132	0.046150	0.204238	0.036331	0.210915	0.108010	0.226822	0.156910	0.258440

Figure 11: SLS - RPD - 50 jobs - left 5 = GA, right 5 = TS

3.3 Average relative percentage deviations of the 5 runs for per instance size

instance	GA	TS
100_20_01	0.611079	1.103.872
100_20_02	0.578331	1.494.516
100_20_03	0.380646	0.991321
100_20_04	0.788215	1.673.719
100_20_05	0.525167	1.486.034
100_20_06	0.699223	1.463.677
100_20_07	0.019209	0.694592
100_20_08	0.810638	1.440.800
100_20_09	0.137762	1.337.645
100_20_10	0.735246	1.844.693
100_20_11	0.699081	1.173.429
100_20_12	0.056076	1.946.589
100_20_13	0.836977	1.071.992
100_20_14	0.598426	1.326.727
100_20_15	0.481527	1.224.837
100_20_16	0.757108	1.256.447
100_20_17	0.293669	1.446.246
100_20_18	0.400447	1.437.679
100_20_19	0.507642	1.534.072
100_20_20	0.596147	0.836078
100_20_21	0.947423	1.267.284
100_20_22	0.406250	1.779.676
100_20_23	0.618699	2.251.728
100_20_24	0.852116	1.867.707
100_20_25	0.751698	2.230.389
100_20_26	0.493933	1.525.633
100_20_27	0.719616	1.904.820
100_20_28	0.705383	1.294.640
100_20_29	0.780091	1.683.219
100_20_30	0.358734	1.728.882

Figure 12: SLS - avg RPD - 100 jobs

50_20_01	0.168162	0.400833
50_20_02	0.064081	0.692417
50_20_03	-0.072274	0.168538
50_20_04	0.076715	0.231166
50_20_05	0.196559	0.618954
50_20_06	0.082015	0.383961
50_20_07	0.028972	0.822958
50_20_08	-0.093028	0.389779
50_20_09	-0.144344	0.126886
50_20_10	0.044974	0.228967
50_20_11	0.057027	0.277458
50_20_12	-0.021089	0.235297
50_20_13	-0.020111	0.230385
50_20_14	0.107722	0.572402
50_20_15	0.055629	0.302435
50_20_16	0.052645	0.320258
50_20_17	-0.143578	0.513665
50_20_18	0.132628	0.297087
50_20_19	0.047991	0.174261
50_20_20	0.106150	0.222985
50_20_21	0.085871	0.424725
50_20_22	0.073195	0.880022
50_20_23	0.042919	0.299965
50_20_24	0.167788	0.383154
50_20_25	0.158077	0.380276
50_20_26	0.257773	0.447679
50_20_27	-0.003831	0.015821
50_20_28	0.047617	0.284935
50_20_29	0.044315	0.377267
50_20_30	0.097524	0.192219

Figure 13: SLS - avg RPD - 50 jobs - left = GA, right = TS

3.4 Overall Average relative percentage deviations over 5 runs

instance	GA1	GA2	GA3	GA4	GA5	TS1	TS2	TS3	TS4	TS5
50	0.046463	0.063622	0.064188	0.069421	0.039321	0.320026	0.302051	0.367633	0.433879	0.392536
100	0.547140	0.577587	0.569169	0.566484	0.597379	1.523.094	1.454.823	1.372.856	1.523.878	1.511.839

Figure 14: SLS - overall avg RPD - 50 and 100 jobs - left = GA, right = TS

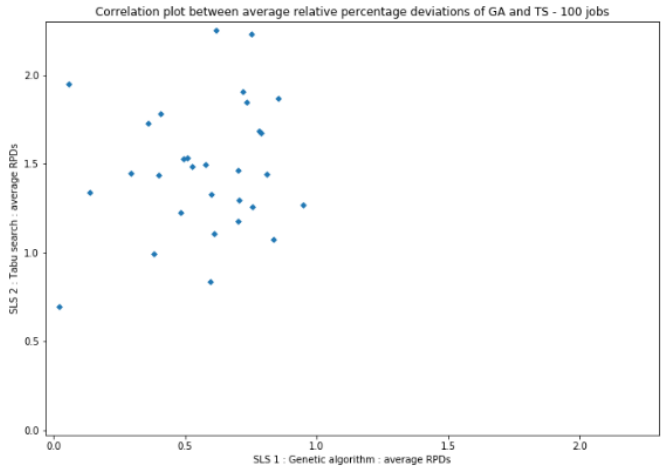
Instance size	GA	TS
50	0,056603	0,363225
100	0,5715518	1,477298

Figure 15: SLS - overall avg RPD - 50 and 100 jobs

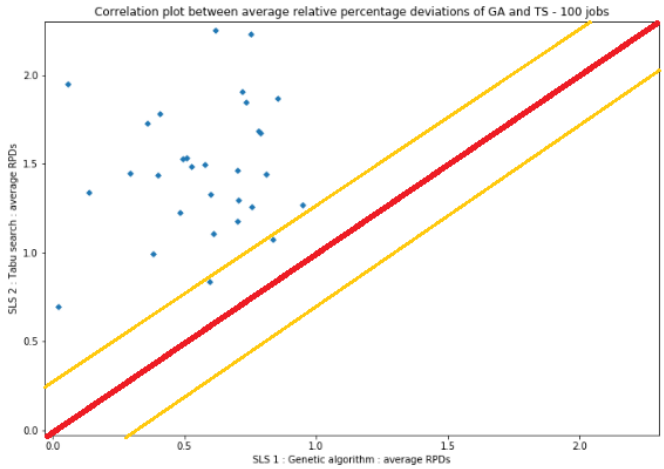
Considering all the results above, we can already make some observations.

- Overall, all the Weighted Sums of completion time are close to the best known solutions provided, which means that two SLS algorithms are both performant.
- If we compare those values to the precedent assignement, we clearly see that the SLS algorithms perform better than Iterative improvement algorithms.
- We can see that the SLS algorithms produce better quality solutions for instances with 50 jobs compared to instances with 100 jobs. It can be explained by the fact that we fix the runtime of each experiment. The time needed for an instance of 100 jobs is probably bigger than that of 50. Hence, in order to have more accurate results, we should have run the SLS algorithm during a longer period of time (for instance 1000 seconds instead of 350 seconds.). We will see that this difference affects the statistical tests a bit.
- Sometimes, Genetic Algorithm manages to find solutions with higher quality than the best known solutions for some instances. Indeed, some relative percentage deviations are negative (which means that the WCT found by the algorithm is lower than the best WCT).

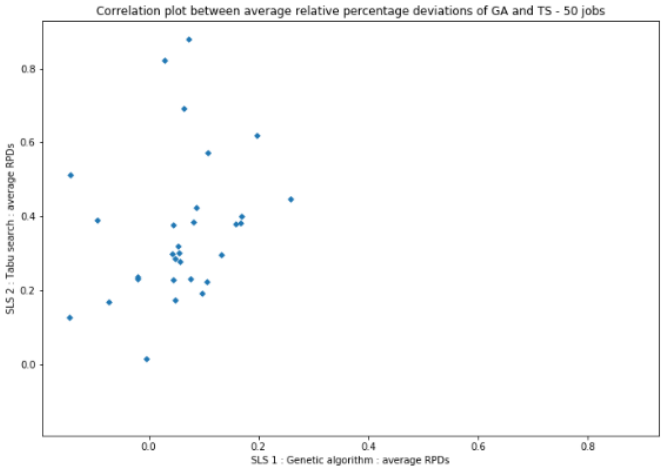
3.5 Correlation plot



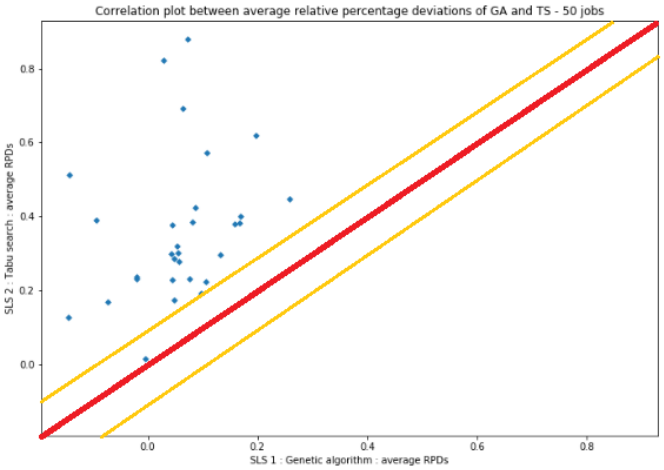
(a) Correlation plot of the average relative percentage deviations of the two SLS algos - **100** jobs



(b) With median Lines



(c) Correlation plot of the average relative percentage deviations of the two SLS algos - **50** jobs



(d) With median Lines

Hereabove, we have the correlation plots of the average relative percentage deviations for the two SLS algorithms, separating between the instances sizes (first 100 jobs, then 50 jobs). In order to evaluate these plots, we will use those metrics :

- **Qualitative Empirical correlation coefficient** : We get the ratio between the overall average relative percentage deviations of GA and TS for each instance size (fig 15) :
  - 50: 0.056603 / 0.363225 = **0.16**
  - 100: 0.5715518 / 1.477298 = **0.39**
- Statistical **Spearman's rank order test** :

instance_size	pvalue	rho
50	0.0488987142348892	0.363737486095662
100	0.5514141615468360	0.112791991101224

Figure 16: SLS - Spearman's correlation test - pvalues and Spearman coefficients

The **Spearman correlation coefficient** is a value within the range [-1,1], where

- -1 : a perfectly negative association between the two sets
- 0 : no association between the two variables
- 1 : a perfectly positive association between the two variables

Thus, according to the two Spearman coefficients (rho) (0.36 and 0.11), we conclude that there is a small positive association between GA and TS. It is a positive correlation, but the coefficients are closer to 0 than 1, so it is a small correlation.

To analyse the p-values, we take a significance level of $\alpha = 0.05$ and the null hypothesis :

- $\alpha = 0.05$
- Null hypothesis : there is no statistical correlation between GA and TS
- The alternative hypothesis : there is a significant correlation between GA and TS

The p-value of instance size 50 (0.04) is below the significance level $\alpha = 0.05$, which means that we reject the null hypothesis and accept the alternative hypothesis. It means that we have a significant correlation between GA and TS.

The p-value of instance size 100 (0.55) is above the significance level, which means that we cannot reject the null hypothesis and we cannot tell if there is a significant difference between GA and TS for 100 jobs. However, as stated before, this difference with respect to instance of 50 jobs is due to the running time that is probably too low for the instances with 100 jobs.

Quantitatively, we can clearly see visually that GA performs better than TS. Indeed, some relative percentage deviations of GA are negative, whilst TS has no negative RPDs. We can also see that the set of points tends to bend towards Tabu search, which means that a lot of pairs (x,y) are in favor of GA, in the sense that $x < y$ (where x and y are both RPDs).

3.6 Statistical tests

instance_size	t-test	wilcoxon
50	0.000000001783753601	0.000000001862645149
100	0.000000000000531311	0.000000001862645149

Figure 17: SLS - p-values based on average relative percentage deviations

Here we have the p-values resulting of 2 statistical tests : paired Student t-test and Wilcoxon-test between 2 sets of average relative percentage deviations.

We have that :

- $\alpha = 0.05$
- Null hypothesis : there is no statistical difference between GA and TS
- The alternative hypothesis : there is a significant difference between GA and TS

Considering the two p-values, we can see that they are both below alpha=0.05, hence we can reject the null hypothesis, which means that there is a significant different between Genetic Algorithm and Tabu Search.

3.7 Run-time distributions

3.7.1 All timings

BK	GA01	GA05	GA1	GA2	BK2	TS01	TS05	TS1	TS2
1.000.000.000	961.635.799	42.587.483	26.915.713	26.915.722	1.000.000.000	536.377.854	335.923.015	60.052.308	4.484.055
268.798.494	34.516.116	18.284.930	18.284.948	18.284.953	1.000.000.000	586.755.673	43.600.704	4.645.720	1.953.963
139.500.792	137.579.863	44.545.127	32.396.446	32.396.462	1.000.000.000	598.729.252	141.984.302	13.763.280	7.161.789
1.000.000.000	1.000.000.000	25.135.822	25.135.845	25.135.852	912.401.032	302.059.828	80.545.009	47.182.082	1.963.846
1.000.000.000	61.607.739	36.425.316	18.233.005	18.233.262	1.000.000.000	1.000.000.000	605.950.976	17.081.582	5.643.921
590.522.318	31.300.993	26.904.626	26.904.646	26.904.652	935.348.814	81.194.609	69.398.801	23.000.585	20.077.609
1.000.000.000	1.000.000.000	63.198.698	31.332.678	31.332.694	1.000.000.000	1.000.000.000	154.986.242	91.562.200	2.766.692
212.026.620	78.992.522	24.643.290	18.913.139	18.913.155	1.000.000.000	1.000.000.000	34.036.667	26.812.614	6.548.187
1.000.000.000	1.000.000.000	27.247.764	27.247.782	27.247.789	1.000.000.000	524.455.763	480.787.501	16.162.510	3.692.231
1.000.000.000	671.361.801	22.690.718	19.288.008	19.288.023	1.000.000.000	1.000.000.000	132.656.525	28.275.379	1.926.698
1.000.000.000	145.903.307	35.069.967	19.775.773	19.775.788	1.000.000.000	1.000.000.000	41.098.895	8.213.543	2.532.673
1.000.000.000	72.137.881	28.331.276	28.331.292	28.331.298	1.000.000.000	1.000.000.000	186.850.359	16.574.859	2.321.450
264.580.722	263.681.102	60.750.100	19.056.493	19.056.507	1.000.000.000	1.000.000.000	294.332.423	28.078.720	7.042.606
212.841.920	212.841.936	94.985.422	33.253.697	33.265.991	1.000.000.000	270.119.214	143.787.598	35.990.246	4.644.940
902.742.182	902.755.001	44.502.893	28.355.395	28.355.410	1.000.000.000	1.000.000.000	80.095.460	14.364.831	3.099.501
1.000.000.000	338.609.274	39.425.587	20.346.426	18.561.124	1.000.000.000	1.000.000.000	437.388.865	9.730.420	2.433.360
1.000.000.000	1.000.000.000	78.802.502	29.897.021	29.897.038	1.000.000.000	1.000.000.000	752.649.504	73.080.274	12.801.034
1.000.000.000	1.000.000.000	34.997.570	20.370.762	20.370.777	796.163.016	128.191.972	75.296.165	43.175.439	9.964.285
311.224.455	100.214.612	19.513.909	19.513.924	19.513.929	821.154.654	817.152.612	319.389.030	43.525.687	6.601.413
1.000.000.000	204.406.392	63.599.399	30.094.310	30.094.323	1.000.000.000	1.000.000.000	39.843.011	9.054.327	1.979.179
1.000.000.000	1.000.000.000	23.884.999	19.941.032	19.941.046	68.658.203	49.477.768	46.268.067	46.154.372	1.698.738
831.289.474	379.707.439	65.587.793	31.492.450	31.492.458	1.000.000.000	1.000.000.000	208.064.087	58.242.587	1.719.239
259.039.101	259.039.342	53.395.642	20.137.676	20.137.691	675.120.474	670.711.265	367.821.427	69.999.292	3.911.933
1.000.000.000	1.000.000.000	29.005.708	20.784.547	18.940.995	374.327.628	364.190.026	35.298.000	8.704.067	5.288.085
1.000.000.000	1.000.000.000	55.408.626	31.252.124	31.252.134	1.000.000.000	444.729.032	111.244.755	79.956.468	2.467.075

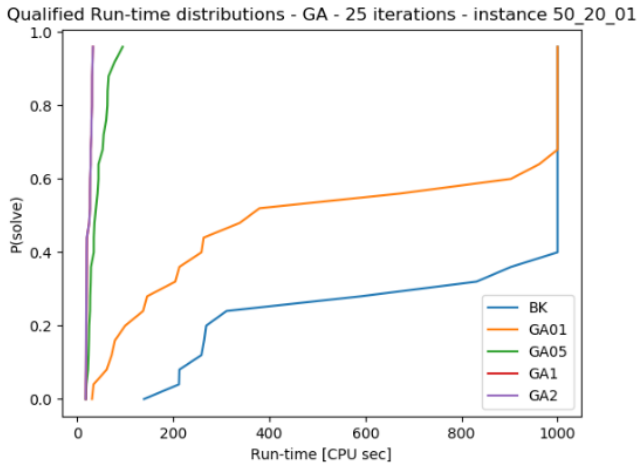
Figure 18: Timings - instance 1 (50_20_01) - 25 iterations

BK	GA01	GA05	GA1	GA2	BK2	TS01	TS05	TS1	TS2
89.715.790	89.728.279	33.225.744	23.565.720	23.565.740	1.000.000.000	62.367.135	47.027.553	41.346.525	3.808.210
228.652.465	98.075.056	63.597.693	32.399.491	32.399.516	1.000.000.000	33.119.015	10.853.871	7.772.081	2.100.077
788.852.296	105.066.876	46.778.473	23.902.480	23.902.495	1.000.000.000	550.853.248	533.792.979	494.215.960	34.904.455
120.278.287	117.240.913	47.121.811	44.316.652	39.644.373	1.000.000.000	1.000.000.000	243.622.023	128.164.769	6.534.536
336.193.547	77.779.838	36.624.451	23.684.348	23.684.362	1.000.000.000	1.000.000.000	1.000.000.000	1.000.000.000	3.554.207
108.034.822	76.081.796	21.860.855	21.861.084	21.861.092	1.000.000.000	285.844.594	271.196.217	16.337.792	11.846.325
87.973.297	58.713.343	42.521.252	38.798.735	32.770.299	1.000.000.000	908.077.137	287.334.919	8.795.736	6.751.109
57.707.514	44.504.585	27.569.159	23.966.391	23.966.407	1.000.000.000	59.517.982	16.231.518	3.990.725	2.493.108
122.406.385	80.577.197	69.361.162	39.082.590	39.082.599	1.000.000.000	491.999.535	273.346.477	83.601.684	5.596.545
107.484.907	82.190.451	23.900.380	23.900.395	23.900.400	1.000.000.000	1.000.000.000	1.000.000.000	729.921.733	34.971.402
83.638.238	83.638.423	33.668.302	24.596.984	24.596.998	1.000.000.000	1.000.000.000	12.313.334	10.122.643	4.575.069
90.153.472	44.920.058	23.637.737	23.637.752	23.637.757	1.000.000.000	148.924.101	25.545.886	16.146.900	1.307.511
179.028.609	138.230.129	27.376.836	22.493.218	22.493.234	486.286.628	485.887.094	154.833.248	49.127.254	2.470.520
315.958.105	285.719.198	38.779.127	38.779.145	38.779.151	58.072.857	57.700.402	21.638.565	2.094.883	1.830.234
348.273.839	71.081.227	39.322.234	23.701.817	23.701.832	618.149.676	617.939.945	579.243.277	438.921.645	12.184.249
133.896.688	113.967.050	64.020.005	51.541.978	38.007.667	301.749.540	291.698.014	43.924.206	21.804.515	3.627.765
1.000.000.000	121.187.323	62.580.647	44.572.667	37.031.645	1.000.000.000	1.000.000.000	114.853.419	14.106.406	7.489.813
61.859.719	37.151.023	26.318.131	25.287.658	25.287.673	1.000.000.000	38.142.104	32.110.537	27.065.458	18.769.068
446.927.907	146.194.914	36.208.375	36.208.385	36.208.391	1.000.000.000	391.524.350	364.884.907	360.040.296	4.208.691
64.929.665	58.834.556	39.398.055	39.398.066	39.398.072	1.000.000.000	529.006.310	211.983.113	75.671.905	2.360.403
58.178.885	58.178.895	23.922.341	23.922.355	23.922.364	1.000.000.000	1.000.000.000	1.000.000.000	239.781.823	1.205.907
234.360.688	129.787.337	82.314.240	52.403.937	35.786.270	670.641.447	667.317.760	264.994.678	75.707.442	12.615.710
70.689.954	70.689.963	65.976.028	33.362.334	33.362.350	1.000.000.000	1.000.000.000	39.273.848	27.164.751	7.961.071
39.839.783	39.839.797	36.195.062	36.195.072	36.195.077	1.000.000.000	187.676.068	91.246.505	73.719.784	3.931.197
165.310.542	89.714.244	54.790.108	36.533.148	36.533.164	595.168.120	595.117.344	383.626.068	18.807.632	6.148.481

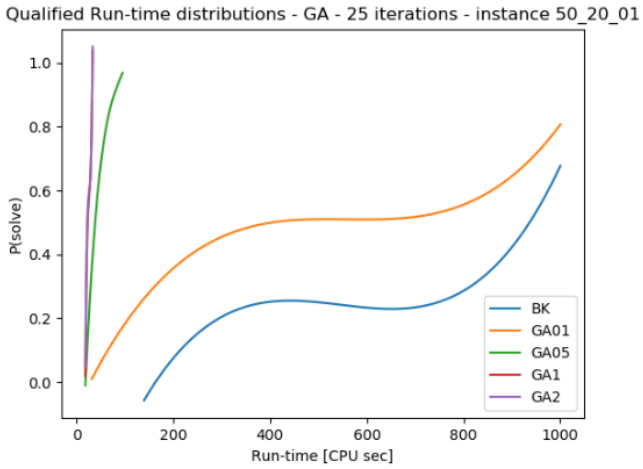
Figure 19: Timings - instance 2 (50_20_02) - 25 iterations

Hereabove are all the timings needed to reach sufficiently high quality solutions for the **2 first instances** of 50 jobs, namely 50_20_01 and 20_20_02. As stated at the beginning of the section, we have the 5 percentages of solutions above the best known : 0.1%,0.5%,1%,2%. (see table 7). The values in the tables above (fig 18 and fig 19) are expressed in seconds and have 6 digits after the comma, which means that 1.000.000.000 is 1000 seconds.

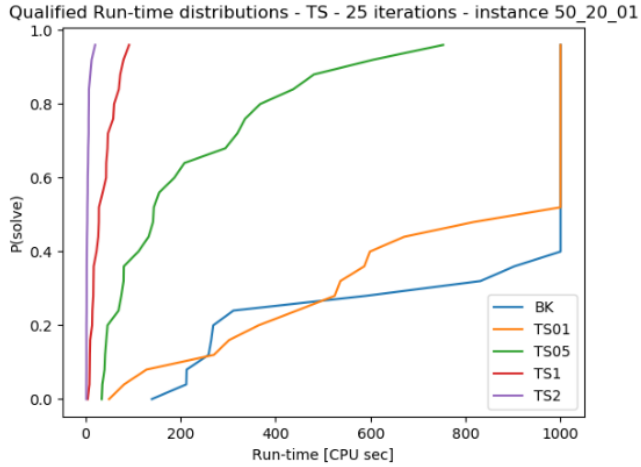
3.7.2 RTDs instance 1 (50_20_01)



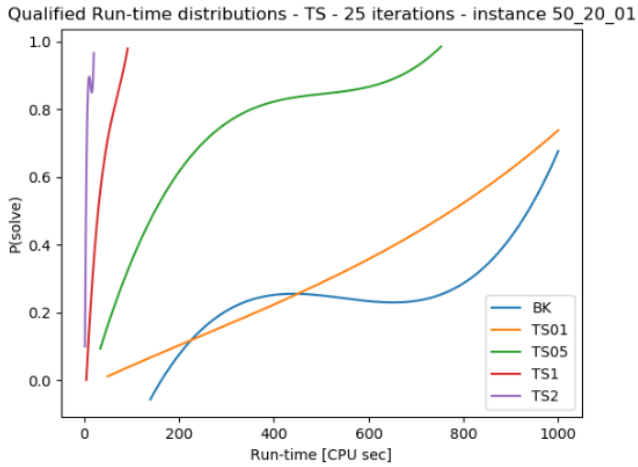
(a) Run-time distribution plot - Genetic Algorithm - 25 iterations - 50_20_01



(b) Run-time distribution plot (smooth) - Genetic Algorithm - 25 iterations - 50_20_01



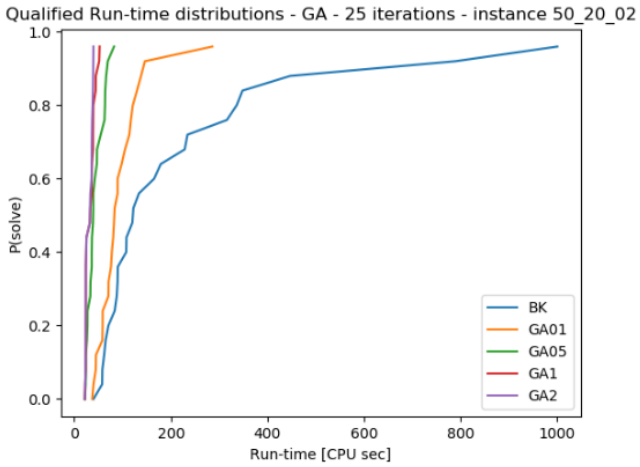
(c) Run-time distribution plot - Tabu Search - 25 iterations - 50_20_01



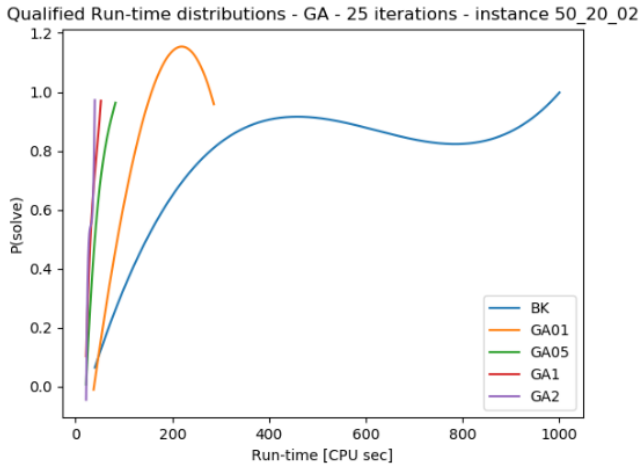
(d) Run-time distribution plot (smooth) - Tabu Search - 25 iterations - 50_20_01

Each plot is accompanied by its smoother version (polyfit).

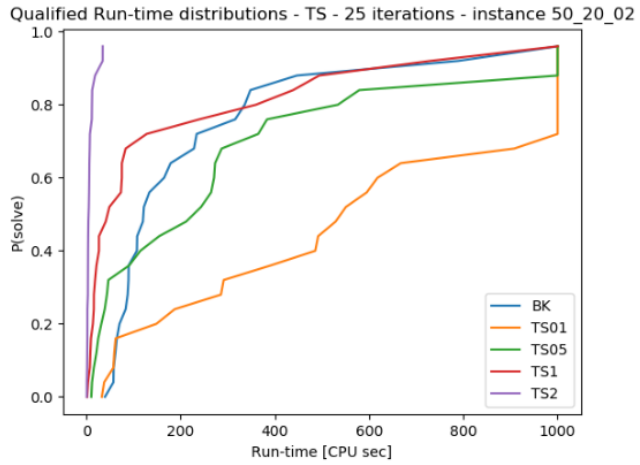
3.7.3 RTDs instance 2 (50_20_02)



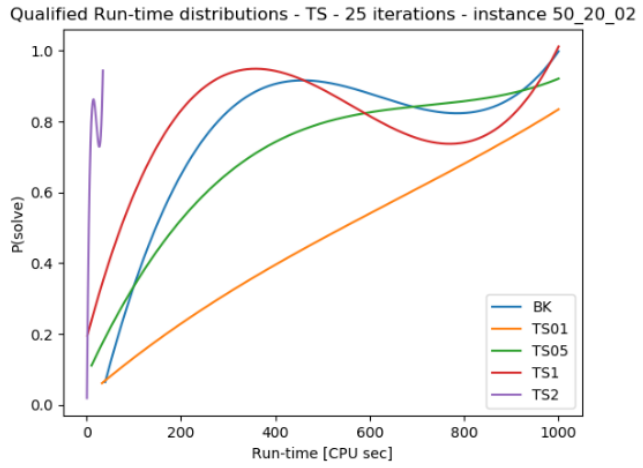
(e) Run-time distribution plot - Genetic Algorithm - 25 iterations - 50_20_02



(f) Run-time distribution plot (smooth) - Genetic Algorithm - 25 iterations - 50_20_02



(g) Run-time distribution plot - Tabu Search - 25 iterations - 50_20_02



(h) Run-time distribution plot (smooth) - Tabu Search - 25 iterations - 50_20_02

From those plots, we can make some observations:

- With sufficiently long run-times, GA and TS manage to find solutions that are better than the best known solutions (BK).
- The plots for instance 1 and instance 2 are globally very similar, except for some percentages (0.5% in green).
- We can see that Genetic Algorithm quickly finds the "easiest" solutions at the beginning of the algorithm (1% and 2% in purple and red, they are often merge in the plot due to the proximity of the values). It is expected since the first generation uses iterative improvement to generate the first offspring children and starts from a random solution. After the first generation, we already have good quality results. Then, for higher quality target solutions, GA takes way more time, but still manage to reach those targets.
- Tabu Search seems to have a more linear tendency across all the percentages. TS struggles to find solutions very close to the best known solutions (TS05, 0.5%).

- Overall, we can see that GA performs better than Tabu search in terms of run-time but both manage to find sufficiently high quality solutions if we let them run for a sufficient time.

For sufficiently long run-times, even if we increase the mean solution quality, the solution quality variability decreases.

3.7.4 Conclusions

1. According to the correlation plot, there is a statistically significant correlation between GA and TS in terms of average relative percentage deviations, even though this correlation is very low.
2. According to the Wilcoxon statistical test, there is a statistical difference between GA and TS
3. If we look at the average relative percentage deviations, we can conclude that Genetic Algorithm performs better than Tabu Search.
4. Genetic Algorithm finds high quality solutions quicker than Tabu Search, but they both manage to find those solutions if we give them enough time to run.

References

[1] Qian Wang Yi Zhang Xiaoping Li. "Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization". In: *European Journal of Operational Research 196 (2009) 869–876* (2008).

[2] Czestaw Smutnicki Eugeniusz Nowicki. "A fast tabu search algorithm for the permutation flow-shop problem". In: *European Journal of Operational Research 91 (1996) 160-175* (1994).

[3] Jarosław Pempera Józef Grabowski. "The permutation flowshop problem with blocking.Atabu search approach". In: *Omega 35 (2007) 302 – 311* (2005).

[4] M. Al-Fawzan M. Ben-Daya. "A tabu search approach for the flow shop scheduling problem". In: *European Journal of Operational Research lo9 (1998) 88-95* (1994).