Revisiting Simulated Annealing: a Component-Based Analysis: Supplementary Material

Alberto Franzin, Thomas Stützle

Preprint submitted to Elsevier *November 7, 2018*

1. SA algorithms for the QAP

1.1. Default settings of the existing SA algorithms

Algorithm	Temperature components	Exploration
`BR1`	**IT3**($l = 10^4$, $k = 1$) **AC1** **CS2**($\alpha = 0.5$) **TL4**($k = 2$) **TR4**($\alpha = 1$) **SC5**($n = 10$)	**NE1**
`BR2`	**IT3**($l = 10^4$, $k = 1$) **AC1** **CS2**($\alpha = 0.5$) **TL9**($k = 1.1$) **TR4**($\alpha = 1$) **SC5**($n = 1$)	**NE1**
`CBR1`	**IT3**($l = 10^4$, $k = 1$) **AC1** **CS2**($\alpha = 0.99$) **TL5**($k = 2$) **TR1** **SC2**($n = 50 \times \|N(s)\|$)	**NE1**
`CBR2`	**IT3**($l = 10^4$, $k = 1$) **AC1** **CS2**($\alpha = 0.99$) **TL9**($k = 1.1$) **TR1** **SC2**($n = 50 \times \|N(s)\|$)	**NE1**
`CLM1`	**IT5**($l = 10^4$, $k = 1$) **AC1** **CS5**(a,b based on the termination) **TL1**($k = 1$) **TR1** **SC2**($n = 50 \times \|N(s)\|$)	**NE1**
`CLM2`	**IT5**($l = 10^4$, $k = 1$) **AC1** **CS5**(a,b based on the termination) **TL1**($k = 1$) **TR1** **SC2**($n = 50 \times \|N(s)\|$)	**NE2**
`Q87`	**IT5**($l = 10^4$, $k = 1$) **AC1** **CS6**(a,b,m based on the termination) **TL1**($k = 1$) **TR1** **SC2**($n = 50 \times \|N(s)\|$)	**NE2**
`Jaj`	**IT6**($l = 10^4$, $k = 1$, $p = 0.8$) **AC1** **CS2**($\alpha = 0.9$) **TL6**($n = 10$, max $= 100$) **TR1** **SC4**($n = 100$)	**NE1**
`Bin`	**IT2**($k = 0.005$) **AC1** **CS2**($\alpha = 0.9$) **TL4**($k = 100$) **TR2**($n = 1$) **SC1**($n = 10s$)	**NE2**
`Tam`	**IT6**($l = 10^4$, $k = 1$, $p = 0.6$) **AC1** **CS2**($\alpha = 0.95$) **TL2**($k = 2$) **TR1** **SC1**($n = 50 \times \|N(s)\|$)	**NE1**

Table 1: Default settings for the ten algorithms of Section 5 of the main paper.

1.2. Results for longer runtimes

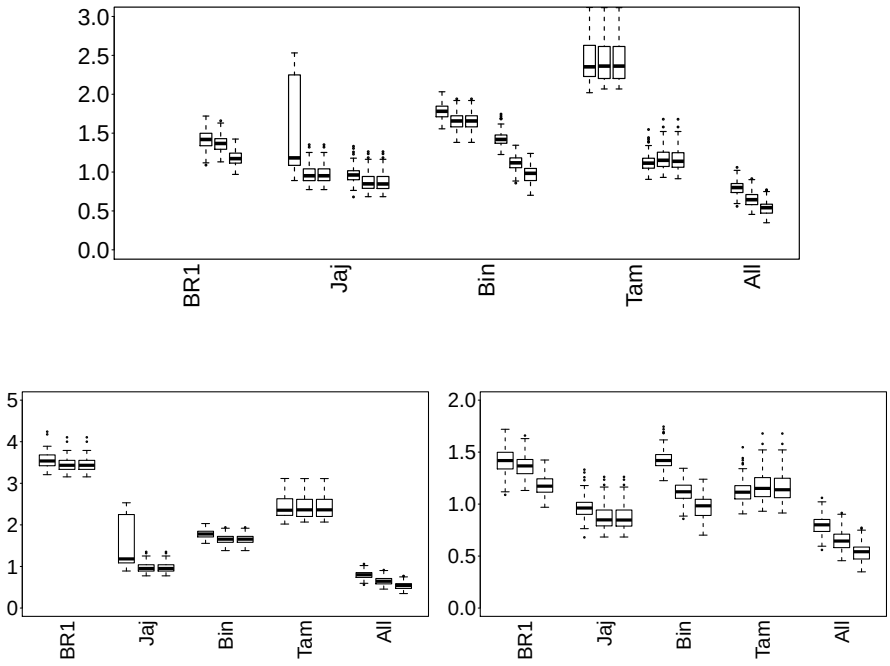

Figure 1: Overview of some of the SA algorithms for solution quality and anytime behaviour on the random QAP scenario, for results obtained in 10, 30 and 100 seconds. The results are reported in terms of average percentage deviation from the best known solutions. The first plot is a comparison of four of the existing algorithms with the three runtimes in the default and the tuned settings, and the automatically designed ones with the tree runtimes; the second and third plot are respectively a direct comparison of the non-tuned algorithms (including the automatically designed ones as a reference point) and the tuned ones, inclusing the automatically designed ones.

In Figures 1 and 2 we show the results obtained by some of our SA algorithms on the random and structured scenarios respectively, when longer runtimes are used, namely of 30 and 100 seconds.

It is interesting to note that even a much longer runtime does not necessarily result in an improved final solution quality. For example, in both the nontuned and the tuned settings, `Tam` is not able to improve over the results found in the ten seconds threshold. The reason for this is that the algorithms has already converged before the ten seconds limit; as it is clear from Figure 6 (see Subsection 1.3 of this supplementary material) in the nontuned setting `Tam` converges very rapidly, and, while the tuning exploits the entire ten seconds interval, it is not able to improve after that. For `Jaj`, instead, in the default settings the ten

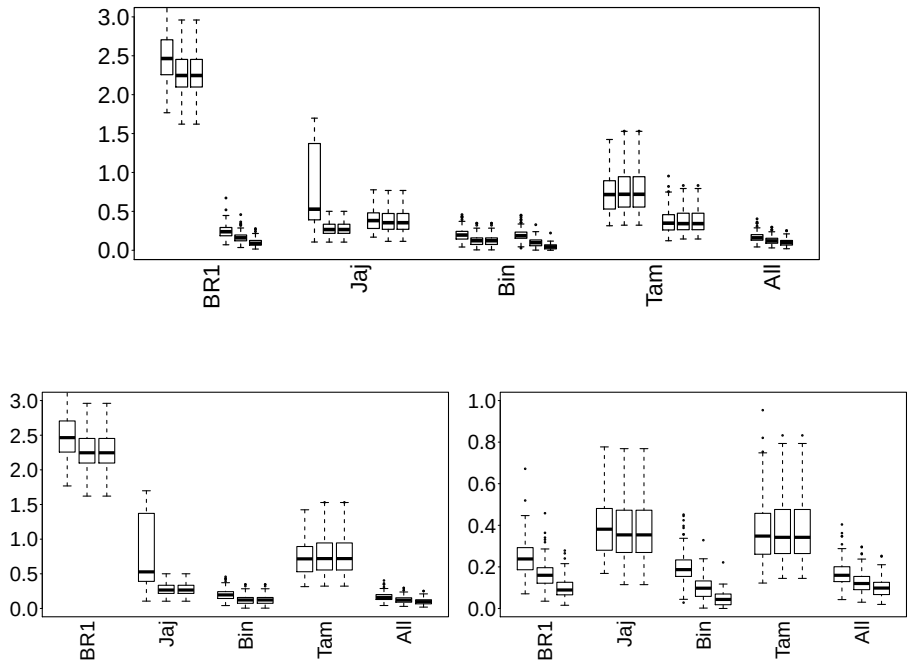

Figure 2: Overview of some of the SA algorithms for solution quality and anytime behaviour on the structured QAP scenario, for results obtained in 10, 30 and 100 seconds. The results are reported in terms of average percentage deviation from the best known solutions. The first plot is a comparison of four of the existing algorithms with the three runtimes in the default and the tuned settings, and the automatically designed ones with the tree runtimes; the second and third plot are respectively a direct comparison of the non-tuned algorithms (including the automatically designed ones as a reference point) and the tuned ones, inclusing the automatically designed ones.

seconds threshold is too short, and more time proves beneficial. However, once converged, the algorithm it is not able to escape the best solution found. Again, the tuning has a "regularization" effect and brings the algorithm to convergence in the ten seconds limit, but a longer runtime is of little help. For `BR1` and `Bin`, while the longer runtime does not have much impact in the nontuned setting, it is indeed beneficial after the tuning. Also the automatically generated algorithms profit by the additional runtime.

Looking at the composition of the algorithms, we observe that the factor that lets the algorithms benefit from the additional runtime is the presence of a temperature restart scheme. `Jaj` and `Tam` do not employ any, and while the tuning can bring their performance on par with (and sometimes better than) the other algorithms in the tuning time threshold, the lack of a temperature restart strategy prevents them from improving after that time.

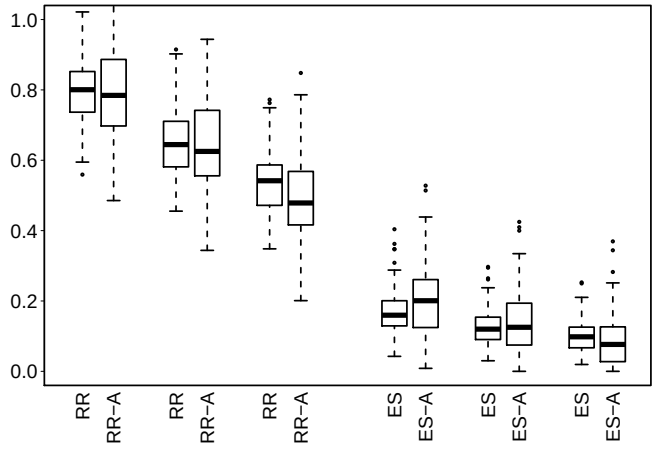

Figure 3: Overview of the automatically designed SA algorithms for solution quality and anytime behaviour on the two QAP scenarios, for results obtained in 10, 30 and 100 seconds. The results are reported in terms of average percentage deviation from the best known solutions.

In Figure 3 we compare the SA automatically generated for solution quality versus the algorithms generated for anytime behaviour. In the long run, the algorithms generated for anytime behaviour appear to have an edge over the ones designed for solution quality.

However, as the cost of the tuning for anytime behaviour is much higher than the (already high) one for solution quality, the user should carefully evaluate whether it is really necessary to bear such costs to reach such high quality solutions, or a quick sufficiently good solution is enough. The same considerations apply for the runtime needed. Finally, tuning over a longer runtime is likely to give further improved results, again at the cost of a much higher computational effort.

1.3. Move acceptance

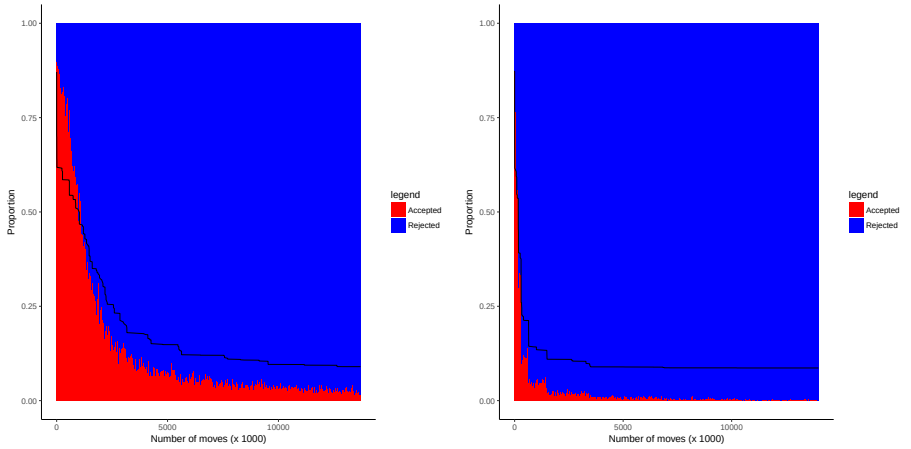

Figure 4: Ratio for 1000 moves of accepted (in red) versus rejected moves (in blue) for `Jaj` in its default settings and after the tuning of its numerical parameters on a size-100 structured QAP instance, observed over ten seconds. The black line indicates the progress of the solution quality.

In figures 4–6 we report the proportion of accepted versus rejected moves for three algorithms exhibiting different convergence behaviours, namely `Jaj`, `Bin`, and `Tam`, in both their default settings and after the tuning of their numerical parameters, on a structured QAP instance of size 100. In Figure 7 we show the same profile for one of the SA algorithms automatically designed for solution quality and for a SA algorithm automatically designed for anytime behaviour. Those convergence profiles show the proportion of accepted moves (in red) versus rejected moves (in blue) aggregated over 1000 moves evaluated. As a reference, the black line gives the progress of the solution quality during the search (we report the trend observed over the acceptance, rather than the exact value).

A clear indication of these plots is that the effect of the tuning is a careful balance between exploration and exploitation. In Figure 4, we observe that the tuning of `Jaj` enforces a stricter acceptance in the beginning of the search. For `Tam` (Figure 6), instead, a too strict acceptance policy is the cause of a premature convergence that results in a very poor performance using the default settings; the tuning instead allows the search to accept more worsening moves, which translates in a much improved quality of the final solution. This slow acceptance might however become a problem if the test settings differ from the tuning ones (e.g. a much shorter runtime), and this is one of the main reasons to aim for a good anytime behaviour (see Section 3 of the Supplementary Material). The effect of the tuning for `Bin` (Figure 5) is instead a sharper alternance between intensification and exploitation; in particular, we notice how the greater improvement of the solution quality comes with the first intensification phase.

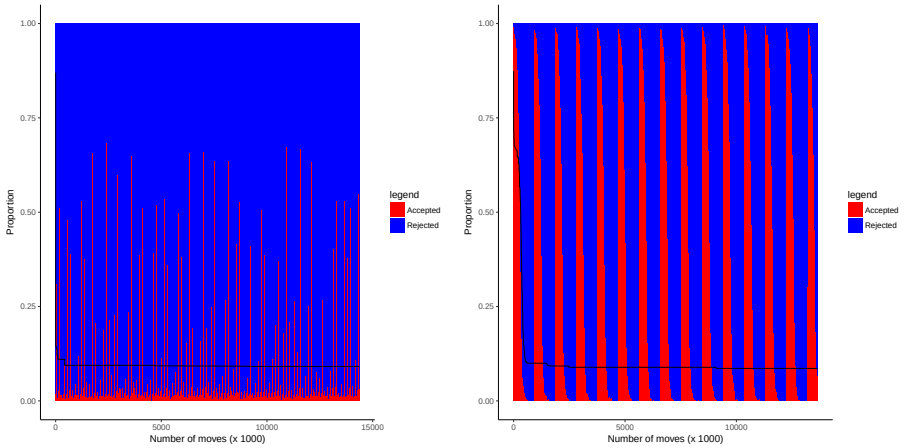

Figure 5: Ratio for 1000 moves of accepted (in red) versus rejected moves (in blue) for `Bin` in its default settings and after the tuning of its numerical parameters on a size-100 structured QAP instance, observed over ten seconds. The black line indicates the progress of the solution quality.

In Figure 7 we show instead the acceptance and convergence behaviour of one of the automatically generated SA algorithms. In this case, there is a strong. but not too strong, initial intensification, and a stronger intensification is observed for the anytime behaviour case (right plot). A small diversification is anyway ensured throughout the search in both cases.

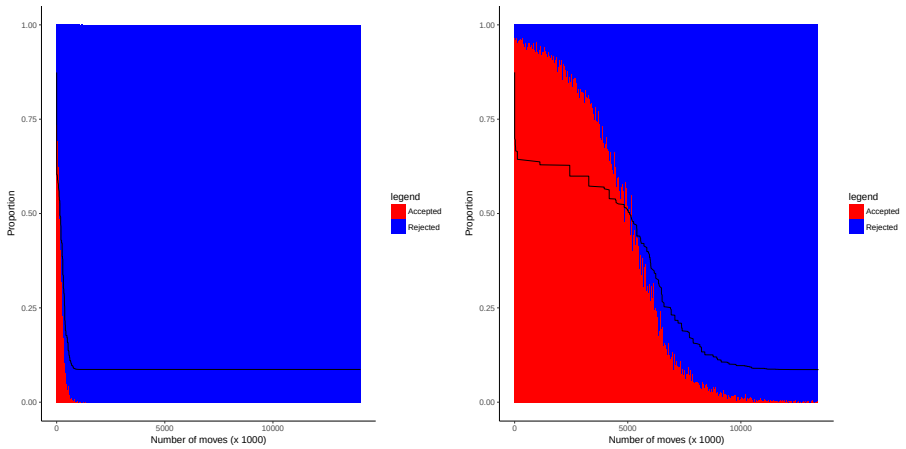

Figure 6: Ratio for 1000 moves of accepted (in red) versus rejected moves (in blue) for `Tam` in its default settings and after the tuning of its numerical parameters on a size-100 structured QAP instance, observed over ten seconds. The black line indicates the progress of the solution quality.

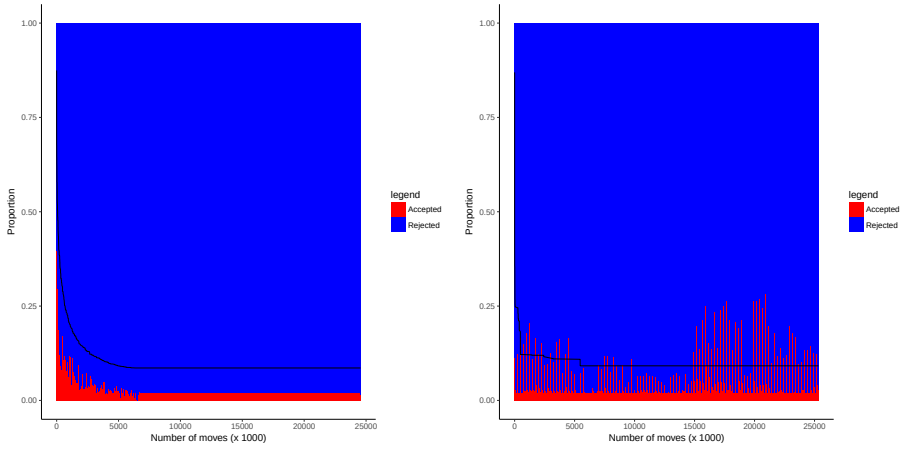

Figure 7: Ratio for 1000 moves of accepted (in red) versus rejected moves (in blue) for an automatically generated SA for solution quality and an automatically generated SA for anytime behaviour on a size-100 structured QAP instance, observed over ten seconds. The black line indicates the progress of the solution quality.

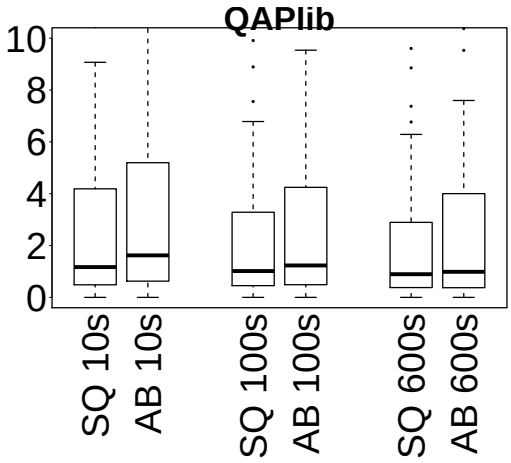

Figure 8: Average Relative Percentage Deviation (ARPD) from the optimal or best known solutions obtained by the automatically generated SA algorithms on the entire QAPlib for 10, 100 and 600 seconds of runtime.

1.4. Results on the QAPlib

Here we report the results obtained by the SA algorithms automatically generated on the random QAP instances scenario, both for solution quality and anytime behaviour (15 and 3 algorithm respectively), when ran on the QAPlib benchmark for different runtimes (10, 100, 600 seconds), divided by instance subgroup. The QAPlib is a benchmark containing instances of different characteristics, therefore a more diverse scenario with respect to the random and structured instances we have considered in the rest of the paper. As the algorithms are designed for uniformly random instances, they will not be able to exploit the particular structures of some instance classes, but they are more likely to outperform the algorithms designed for the structured instances. In Tables 2–4 and Figures 8, 9 we show our results, reporting for each instance the average percentage deviation from the optimal or best known solutions, the number of times an optimal or best known solution has been obtained, and the best deviation from the optimal or best known solutions, for the algorithms designed for solution quality (sq) and anytime behaviour (ab).

1.4.1. 10 seconds runtime

Instance	avg RPD sq	avg RPD ab	# solved sq	# solved ab	best RPD sq	best RPD ab
bur26a	0.1600171	0.2913082	2 /15	0 /3	0	0.001657739
bur26b	0.2388516	0.2951834	1 /15	0 /3	0	0.0004685881
bur26c	0.1591424	0.1837426	2 /15	0 /3	0	0.0004418814
bur26d	0.1724351	0.5016192	1 /15	0 /3	0	0.0002009827
bur26e	0.08798292	0.02997407	4 /15	0 /3	0	0.0001962175
bur26f	0.1364412	0.524399	2 /15	0 /3	0	0.000231092

bur26g	0.2340977	0.1835263	4 /15	0 /3	0	0.0001409485
bur26h	0.2574618	0.3386274	3 /15	0 /3	0	0.0001924307
chr12a	30.50112	65.42434	7 /15	1 /3	0	0
chr12b	16.60166	20.29015	9 /15	2 /3	0	0
chr12c	18.24071	23.90343	7 /15	1 /3	0	0
chr15a	22.72972	35.06467	5 /15	1 /3	0	0
chr15b	27.37922	38.28118	6 /15	1 /3	0	0
chr15c	32.83389	43.44837	6 /15	1 /3	0	0
chr18a	33.44987	22.97111	5 /15	1 /3	0	0
chr18b	4.632768	11.69057	8 /15	1 /3	0	0
chr20a	18.94769	10.88808	2 /15	1 /3	0	0
chr20b	17.36002	12.53264	2 /15	1 /3	0	0
chr20c	45.99915	67.83105	3 /15	1 /3	0	0
chr22a	7.24713	8.05718	3 /15	0 /3	0	0.04158545
chr22b	6.559036	7.426542	2 /15	0 /3	0	0.042299
chr25a	19.92624	24.6751	1 /15	0 /3	0	0.02423604
els19	12.21875	14.98959	1 /15	0 /3	0	0.04227079
esc16a	0.1960784	0	14 /15	3 /3	0	0
esc16b	0	0	15 /15	3 /3	0	0
esc16c	0	0	15 /15	3 /3	0	0
esc16d	0	0	15 /15	3 /3	0	0
esc16e	0	0	15 /15	3 /3	0	0
esc16f	0	0	15 /15	3 /3	0	0
esc16g	0	0	15 /15	3 /3	0	0
esc16h	0	0	15 /15	3 /3	0	0
esc16i	0	0	15 /15	3 /3	0	0
esc16j	0	0	15 /15	3 /3	0	0
esc32a	6.153846	5.641026	7 /15	1 /3	0	0
esc32b	5.079365	11.11111	10 /15	1 /3	0	0
esc32c	0	0	15 /15	3 /3	0	0
esc32d	0.6666667	2.333333	11 /15	2 /3	0	0
esc32e	0	0	15 /15	3 /3	0	0
esc32g	0	0	15 /15	3 /3	0	0
esc32h	1.065449	1.369863	8 /15	2 /3	0	0
esc64a	0	0	15 /15	3 /3	0	0
esc128	2.916667	0	9 /15	3 /3	0	0
had12	0.4923325	0.2421308	8 /15	1 /3	0	0
had14	0.0783162	0.7342144	13 /15	1 /3	0	0
had16	0.1971326	0.483871	6 /15	2 /3	0	0
had18	0.3682966	0.6470076	5 /15	1 /3	0	0
had20	0.6741789	0.2215159	5 /15	1 /3	0	0
kra30a	2.896138	4.836895	7 /15	0 /3	0	0.01338583
kra30b	1.883614	2.79297	7 /15	1 /3	0	0
kra32	2.866592	5.114619	7 /15	1 /3	0	0
lipa20a	0.9937551	1.592904	8 /15	1 /3	0	0

lipa20b	4.647658	4.237455	10 /15	2 /3	0	0
lipa30a	0.7178631	1.09273	8 /15	1 /3	0	0
lipa30b	4.06271	4.952694	11 /15	2 /3	0	0
lipa40a	1.098284	1.284095	0 /15	0 /3	0.005740162	0.005740162
lipa40b	3.242065	10.87839	12 /15	1 /3	0	0
lipa50a	0.41411	0.5620601	8 /15	1 /3	0	0
lipa50b	3.271384	10.55076	12 /15	1 /3	0	0
lipa60a	0.2918664	0.4660287	9 /15	1 /3	0	0
lipa60b	5.836468	5.858985	10 /15	2 /3	0	0
lipa70a	0.3313403	0.4039154	7 /15	1 /3	0	0
lipa70b	4.916859	6.370793	11 /15	1 /3	0	0
lipa80a	0.4397138	0.3628297	3 /15	1 /3	0	0
lipa80b	6.46078	6.751695	10 /15	1 /3	0	0
lipa90a	0.4462006	0.5053194	2 /15	0 /3	0	0.004966309
lipa90b	10.45023	13.19174	7 /15	1 /3	0	0
nug12	1.914648	1.730104	9 /15	2 /3	0	0
nug14	2.064431	2.761341	7 /15	1 /3	0	0
nug15	1.310145	1.565217	7 /15	1 /3	0	0
nug16a	1.995859	3.602484	6 /15	1 /3	0	0
nug16b	1.903226	2.795699	8 /15	1 /3	0	0
nug17	1.147036	1.847575	8 /15	1 /3	0	0
nug18	1.360967	2.38342	7 /15	1 /3	0	0
nug20	0.8871595	1.997406	7 /15	1 /3	0	0
nug21	1.53131	1.121138	7 /15	1 /3	0	0
nug22	1.171672	1.575825	4 /15	1 /3	0	0
nug24	1.043578	1.75841	8 /15	1 /3	0	0
nug25	0.8796296	0.8725071	8 /15	1 /3	0	0
nug27	1.413833	0.9552923	7 /15	2 /3	0	0
nug28	1.099497	1.638921	8 /15	1 /3	0	0
nug30	0.5617244	1.404311	5 /15	1 /3	0	0
rou12	2.242281	3.657598	8 /15	1 /3	0	0
rou15	2.415479	1.754703	7 /15	1 /3	0	0
rou20	1.131415	1.779592	7 /15	1 /3	0	0
scr12	2.409424	2.774063	10 /15	2 /3	0	0
scr15	4.253422	4.719072	7 /15	1 /3	0	0
scr20	4.350692	3.740798	7 /15	1 /3	0	0
ste36a	5.058437	9.43383	3 /15	0 /3	0	0.0247743
ste36b	12.70418	15.53957	4 /15	1 /3	0	0
ste36c	4.596085	5.840016	5 /15	0 /3	0	0.03973366
sko42	0.7867442	1.323889	2 /15	0 /3	0	0.004553504
sko49	0.6533824	1.049061	2 /15	0 /3	0	0.007953476
sko56	0.7421595	1.437499	1 /15	0 /3	0	0.009228626
sko64	0.6818151	1.418615	1 /15	0 /3	0	0.009278733
sko72	0.7564598	1.130967	0 /15	0 /3	0.0001811157	0.00908597
sko81	0.617889	1.026396	0 /15	0 /3	0.0001098925	0.006747401

sko90	0.6835506	1.166179	0 /15	0 /3	0.0006405041	0.004050756
sko100a	0.6334127	0.8078841	0 /15	0 /3	0.0004868357	0.002276286
sko100b	0.6360387	1.036671	0 /15	0 /3	3.898889e-05	0.006303204
sko100c	0.8661229	1.283178	0 /15	0 /3	1.352613e-05	0.01078032
sko100d	0.7215953	0.9471216	0 /15	0 /3	0.0006418142	0.005134514
sko100e	0.6877193	1.336462	1 /15	0 /3	0	0.01067382
sko100f	0.7334246	0.812779	0 /15	0 /3	5.367831e-05	0.003824579
tai10a	2.580601	4.029288	8 /15	1 /3	0	0
tai12a	3.512524	4.509482	9 /15	1 /3	0	0
tai15a	1.325764	1.620756	7 /15	1 /3	0	0
tai17a	1.607118	3.21261	7 /15	1 /3	0	0
tai20a	1.828296	1.827197	3 /15	1 /3	0	0
tai25a	1.195442	1.942961	1 /15	1 /3	0	0
tai30a	1.435646	1.200564	2 /15	1 /3	0	0
tai35a	1.17521	1.419735	1 /15	0 /3	0	0.00376548
tai40a	1.056161	1.146302	0 /15	0 /3	0.003777191	0.006593043
tai50a	1.168933	1.462826	0 /15	0 /3	0.006996442	0.01112984
tai60a	1.017502	1.097267	0 /15	0 /3	0.007549859	0.008605097
tai80a	1.16212	1.126424	0 /15	0 /3	0.008452659	0.009826075
tai100a	1.121996	1.182306	0 /15	0 /3	0.008782629	0.009902688
tai10b	4.576609	7.085558	8 /15	1 /3	0	0
tai12b	5.835181	5.762965	8 /15	1 /3	0	0
tai15b	0.4384698	0.5468152	3 /15	0 /3	0	0.004048564
tai20b	9.068211	11.93523	3 /15	0 /3	0	0.01787806
tai25b	6.512946	7.879769	5 /15	0 /3	0	0.01736179
tai30b	7.64588	9.709667	2 /15	0 /3	0	0.03758459
tai35b	4.68597	6.990417	2 /15	0 /3	0	0.02191215
tai40b	7.128245	5.275997	2 /15	0 /3	0	0.02276873
tai50b	4.12182	3.532311	0 /15	0 /3	0.0005867358	0.01966091
tai60b	3.577894	5.855074	1 /15	0 /3	0	0.0179717
tai80b	3.472607	3.834669	0 /15	0 /3	0.0001543373	0.01599609
tai100b	3.380434	3.226662	0 /15	0 /3	0.001073137	0.02709011
tai150b	2.082348	2.663557	0 /15	0 /3	0.005737577	0.02283725
tai64c	0.2029964	0.008405498	5 /15	2 /3	0	0
tai256c	0.4572899	0.4745353	0 /15	0 /3	0.002471978	0.004268298
tho30	1.457155	1.981734	3 /15	0 /3	0	0.005255576
tho40	1.083559	1.759273	0 /15	0 /3	0.0001081009	0.01170816
tho150	0.9878053	1.700913	0 /15	0 /3	0.001870313	0.007448547
wil50	0.4782585	0.6077242	1 /15	0 /3	0	0.004588659
wil100	0.401214	0.6438664	0 /15	0 /3	3.662494e-05	0.00287872

Table 2: Results obtained on the QAPlib for 10 seconds of runtime.

1.4.2. 100 seconds runtime

Instance	avg RPD sq	avg RPD ab	# solved sq	# solved ab	best RPD sq	best RPD ab
bur26a	0.1550552	0.2862468	2 /15	0 /3	0	0.001505896
bur26b	0.2209864	0.2902941	2 /15	0 /3	0	0.0003219088
bur26c	0.1392092	0.1768447	2 /15	0 /3	0	0.0002349453
bur26d	0.1717111	0.5001276	2 /15	0 /3	0	0.0001598964
bur26e	0.08028644	0.02868699	4 /15	0 /3	0	0.0001576052
bur26f	0.1334129	0.5190844	3 /15	0 /3	0	7.165438e-05
bur26g	0.177315	0.1788873	4 /15	0 /3	0	0.0001409485
bur26h	0.2537343	0.3369557	4 /15	0 /3	0	0.0001422804
chr12a	28.65159	54.78085	8 /15	1 /3	0	0
chr12b	16.60166	20.29015	9 /15	2 /3	0	0
chr12c	14.7042	23.90343	7 /15	1 /3	0	0
chr15a	21.90245	35.06467	6 /15	1 /3	0	0
chr15b	27.2257	38.28118	6 /15	1 /3	0	0
chr15c	32.34848	43.40629	6 /15	1 /3	0	0
chr18a	30.62053	21.78771	5 /15	1 /3	0	0
chr18b	4.380704	11.69057	8 /15	1 /3	0	0
chr20a	18.23601	10.88808	5 /15	1 /3	0	0
chr20b	15.20743	12.53264	4 /15	1 /3	0	0
chr20c	42.58521	67.83105	3 /15	1 /3	0	0
chr22a	6.434914	6.779294	4 /15	0 /3	0	0.005523067
chr22b	5.695835	7.017544	2 /15	0 /3	0	0.03002906
chr25a	17.67475	22.56762	3 /15	1 /3	0	0
els19	12.12866	14.98356	1 /15	0 /3	0	0.04208999
esc16a	0.1960784	0	14 /15	3 /3	0	0
esc16b	0	0	15 /15	3 /3	0	0
esc16c	0	0	15 /15	3 /3	0	0
esc16d	0	0	15 /15	3 /3	0	0
esc16e	0	0	15 /15	3 /3	0	0
esc16f	0	0	15 /15	3 /3	0	0
esc16g	0	0	15 /15	3 /3	0	0
esc16h	0	0	15 /15	3 /3	0	0
esc16i	0	0	15 /15	3 /3	0	0
esc16j	0	0	15 /15	3 /3	0	0
esc32a	5.948718	5.641026	7 /15	1 /3	0	0
esc32b	4.920635	11.11111	10 /15	1 /3	0	0
esc32c	0	0	15 /15	3 /3	0	0
esc32d	0.6666667	2.333333	11 /15	2 /3	0	0
esc32e	0	0	15 /15	3 /3	0	0
esc32g	0	0	15 /15	3 /3	0	0
esc32h	1.065449	1.369863	8 /15	2 /3	0	0
esc64a	0	0	15 /15	3 /3	0	0
esc128	2.291667	0	10 /15	3 /3	0	0
had12	0.4923325	0.2421308	8 /15	1 /3	0	0
had14	0.05384239	0.7342144	14 /15	1 /3	0	0

had16	0.1756272	0.483871	6 /15	2 /3	0	0
had18	0.2811995	0.6470076	6 /15	1 /3	0	0
had20	0.6684003	0.09631128	6 /15	2 /3	0	0
kra30a	2.512186	4.390701	7 /15	1 /3	0	0
kra30b	1.694013	2.282506	7 /15	1 /3	0	0
kra32	2.753852	4.340474	7 /15	1 /3	0	0
lipa20a	0.8579962	1.592904	8 /15	1 /3	0	0
lipa20b	4.647658	4.237455	10 /15	2 /3	0	0
lipa30a	0.7082511	1.052259	8 /15	1 /3	0	0
lipa30b	1.976367	4.952694	13 /15	2 /3	0	0
lipa40a	0.9985756	1.233072	0 /15	0 /3	0.005740162	0.005740162
lipa40b	1.065982	10.42859	14 /15	1 /3	0	0
lipa50a	0.3460401	0.537366	9 /15	1 /3	0	0
lipa50b	1.066264	0	14 /15	3 /3	0	0
lipa60a	0.269793	0.2145162	9 /15	2 /3	0	0
lipa60b	3.473213	0	12 /15	3 /3	0	0
lipa70a	0.2606894	0.365625	8 /15	1 /3	0	0
lipa70b	1.233446	0	14 /15	3 /3	0	0
lipa80a	0.2350494	0.1640369	8 /15	2 /3	0	0
lipa80b	2.561015	0	13 /15	3 /3	0	0
lipa90a	0.2905101	0.4772204	6 /15	0 /3	0	0.004578099
lipa90b	2.593604	0	13 /15	3 /3	0	0
nug12	1.914648	1.730104	9 /15	2 /3	0	0
nug14	1.959237	2.761341	7 /15	1 /3	0	0
nug15	1.275362	1.507246	8 /15	1 /3	0	0
nug16a	1.648033	2.691511	6 /15	1 /3	0	0
nug16b	1.623656	2.795699	10 /15	1 /3	0	0
nug17	1.016166	1.847575	8 /15	1 /3	0	0
nug18	1.126079	2.38342	8 /15	1 /3	0	0
nug20	0.8041505	1.945525	9 /15	1 /3	0	0
nug21	1.482089	1.121138	7 /15	1 /3	0	0
nug22	1.108639	1.575825	4 /15	1 /3	0	0
nug24	0.9594801	1.75841	8 /15	1 /3	0	0
nug25	0.8475783	0.730057	8 /15	1 /3	0	0
nug27	1.375621	0.9552923	7 /15	2 /3	0	0
nug28	0.8878565	1.638921	9 /15	1 /3	0	0
nug30	0.5399521	1.262791	8 /15	1 /3	0	0
rou12	2.242281	2.864769	8 /15	1 /3	0	0
rou15	2.305676	0.6931858	7 /15	1 /3	0	0
rou20	1.084994	1.231665	7 /15	1 /3	0	0
scr12	2.406028	2.774063	10 /15	2 /3	0	0
scr15	4.066484	4.719072	7 /15	1 /3	0	0
scr20	4.022176	3.700203	8 /15	1 /3	0	0
ste36a	4.21863	7.425292	4 /15	0 /3	0	0.0247743
ste36b	9.907478	12.54521	5 /15	1 /3	0	0

ste36c	3.961442	4.252466	5 /15	0 /3	0	0.02815668
sko42	0.6206257	1.218484	3 /15	0 /3	0	0.002023779
sko49	0.4772086	0.5416346	2 /15	0 /3	0	0.003164286
sko56	0.606342	1.274982	4 /15	0 /3	0	0.009112543
sko64	0.4629744	0.8880091	2 /15	0 /3	0	0.005690956
sko72	0.5707156	0.8995412	0 /15	0 /3	0.0001811157	0.008029461
sko81	0.5030147	0.8740119	1 /15	0 /3	0	0.003208862
sko90	0.5945724	0.7051315	0 /15	0 /3	6.924369e-05	0.00225042
sko100a	0.5382385	0.6254304	0 /15	0 /3	0.0001578927	0.0002894699
sko100b	0.4736717	0.7689475	1 /15	0 /3	0	0.001715511
sko100c	0.7048915	0.9508866	0 /15	0 /3	1.352613e-05	0.005491607
sko100d	0.5329732	0.7314008	0 /15	0 /3	0.0001337113	0.001283628
sko100e	0.5661415	0.8434462	2 /15	0 /3	0	0.00177003
sko100f	0.5885826	0.7747569	2 /15	0 /3	0	0.002952307
tai10a	2.580601	4.029288	8 /15	1 /3	0	0
tai12a	2.99842	4.509482	10 /15	1 /3	0	0
tai15a	1.263736	1.620756	8 /15	1 /3	0	0
tai17a	1.529663	2.860849	7 /15	1 /3	0	0
tai20a	1.701024	1.827197	5 /15	1 /3	0	0
tai25a	0.9845312	1.885733	4 /15	1 /3	0	0
tai30a	1.057399	0.6986971	4 /15	1 /3	0	0
tai35a	0.7992947	0.9826306	4 /15	1 /3	0	0
tai40a	0.7290677	0.8281067	0 /15	0 /3	0.003170063	0.003347805
tai50a	0.9267711	0.6344866	0 /15	0 /3	0.004267437	0.002995467
tai60a	0.8352491	0.6907891	0 /15	0 /3	0.005183486	0.006052766
tai80a	0.8902721	0.8702798	0 /15	0 /3	0.007287848	0.008381395
tai100a	0.8786334	0.8283273	0 /15	0 /3	0.006881664	0.007368923
tai10b	4.576609	7.085558	8 /15	1 /3	0	0
tai12b	5.835181	5.762965	8 /15	1 /3	0	0
tai15b	0.3677806	0.5086319	3 /15	0 /3	0	0.003095319
tai20b	8.891616	11.76788	4 /15	0 /3	0	0.01285766
tai25b	6.352692	7.68787	5 /15	0 /3	0	0.01160482
tai30b	7.558016	9.536209	4 /15	0 /3	0	0.03374976
tai35b	4.567697	5.780727	3 /15	0 /3	0	0.01626727
tai40b	6.784171	5.141299	4 /15	0 /3	0	0.02276873
tai50b	3.943679	3.285393	1 /15	0 /3	0	0.01620332
tai60b	3.416735	5.771464	2 /15	0 /3	0	0.01629278
tai80b	3.292134	3.800812	0 /15	0 /3	1.299952e-05	0.01498373
tai100b	3.2773	2.993834	0 /15	0 /3	0.0007184206	0.02053136
tai150b	1.966695	2.608316	0 /15	0 /3	0.003486343	0.02149772
tai64c	0.2013153	0.008405498	6 /15	2 /3	0	0
tai256c	0.4419218	0.4434937	0 /15	0 /3	0.002471978	0.004268298
tho30	1.140309	1.275655	5 /15	1 /3	0	0
tho40	0.862479	1.009496	0 /15	0 /3	0.0001081009	0.005587986
tho150	0.7639087	1.419817	0 /15	0 /3	0.0001989328	0.006362163

wil50	0.43046	0.4943734	3 /15	0 /3	0	0.001843658
wil100	0.3392934	0.5503507	0 /15	0 /3	2.197496e-05	0.001897172

Table 3: Results obtained on the QAPlib for 100 seconds of runtime.

1.4.3. 600 seconds runtime

Instance	avg RPD sq	avg RPD ab	# solved sq	# solved ab	best RPD sq	best RPD ab
bur26a	0.1524287	0.2417812	2 /15	0 /3	0	0.0008154172
bur26b	0.2175866	0.2847413	2 /15	0 /3	0	0.0001553229
bur26c	0.1333691	0.173276	2 /15	0 /3	0	0.000127884
bur26d	0.1694535	0.4954781	2 /15	0 /3	0	2.04123e-05
bur26e	0.07600195	0.02581581	4 /15	0 /3	0	0.0001576052
bur26f	0.1326408	0.5181413	3 /15	0 /3	0	4.33628e-05
bur26g	0.1738905	0.1788873	4 /15	0 /3	0	0.0001409485
bur26h	0.2487503	0.3368618	4 /15	0 /3	0	0.000139463
chr12a	27.51256	54.78085	8 /15	1 /3	0	0
chr12b	16.60166	20.29015	9 /15	2 /3	0	0
chr12c	14.41257	23.90343	7 /15	1 /3	0	0
chr15a	21.31501	35.06467	7 /15	1 /3	0	0
chr15b	25.38006	38.28118	6 /15	1 /3	0	0
chr15c	31.21072	43.40629	6 /15	1 /3	0	0
chr18a	26.9454	21.78771	5 /15	1 /3	0	0
chr18b	4.380704	11.69057	8 /15	1 /3	0	0
chr20a	17.40268	10.37105	5 /15	1 /3	0	0
chr20b	14.84189	12.09748	4 /15	1 /3	0	0
chr20c	42.34007	67.83105	3 /15	1 /3	0	0
chr22a	6.064544	5.847953	4 /15	0 /3	0	0.005523067
chr22b	5.065117	6.296416	2 /15	0 /3	0	0.008395221
chr25a	15.96417	22.56762	4 /15	1 /3	0	0
els19	11.98951	14.98356	1 /15	0 /3	0	0.04208999
esc16a	0.1960784	0	14 /15	3 /3	0	0
esc16b	0	0	15 /15	3 /3	0	0
esc16c	0	0	15 /15	3 /3	0	0
esc16d	0	0	15 /15	3 /3	0	0
esc16e	0	0	15 /15	3 /3	0	0
esc16f	0	0	15 /15	3 /3	0	0
esc16g	0	0	15 /15	3 /3	0	0
esc16h	0	0	15 /15	3 /3	0	0
esc16i	0	0	15 /15	3 /3	0	0
esc16j	0	0	15 /15	3 /3	0	0
esc32a	5.74359	5.641026	8 /15	1 /3	0	0

esc32b	4.920635	11.11111	10 /15	1 /3	0	0
esc32c	0	0	15 /15	3 /3	0	0
esc32d	0.6666667	2.333333	11 /15	2 /3	0	0
esc32e	0	0	15 /15	3 /3	0	0
esc32g	0	0	15 /15	3 /3	0	0
esc32h	1.065449	1.369863	8 /15	2 /3	0	0
esc64a	0	0	15 /15	3 /3	0	0
esc128	1.875	0	10 /15	3 /3	0	0
had12	0.4358354	0.2421308	8 /15	1 /3	0	0
had14	0.05384239	0.7342144	14 /15	1 /3	0	0
had16	0.1684588	0.4480287	7 /15	2 /3	0	0
had18	0.2811995	0.6470076	6 /15	1 /3	0	0
had20	0.6395069	0.09631128	7 /15	2 /3	0	0
kra30a	2.382452	4.390701	7 /15	1 /3	0	0
kra30b	1.637862	2.136659	7 /15	1 /3	0	0
kra32	2.68696	4.340474	7 /15	1 /3	0	0
lipa20a	0.6824147	1.592904	9 /15	1 /3	0	0
lipa20b	4.618851	4.237455	10 /15	2 /3	0	0
lipa30a	0.6288258	0.9864926	9 /15	1 /3	0	0
lipa30b	1.967474	4.952694	13 /15	2 /3	0	0
lipa40a	0.9921976	1.205434	0 /15	0 /3	0.005740162	0.005740162
lipa40b	0	5.342429	15 /15	2 /3	0	0
lipa50a	0.2694882	0.2582148	10 /15	2 /3	0	0
lipa50b	1.062684	0	14 /15	3 /3	0	0
lipa60a	0.1727944	0	11 /15	3 /3	0	0
lipa60b	2.313638	0	13 /15	3 /3	0	0
lipa70a	0.1837157	0.3548251	10 /15	1 /3	0	0
lipa70b	1.203453	0	14 /15	3 /3	0	0
lipa80a	0.1006076	0	12 /15	3 /3	0	0
lipa80b	0	0	15 /15	3 /3	0	0
lipa90a	0.2175452	0.1440997	8 /15	2 /3	0	0
lipa90b	2.586771	0	13 /15	3 /3	0	0
nug12	1.730104	1.730104	9 /15	2 /3	0	0
nug14	1.867193	2.695595	7 /15	1 /3	0	0
nug15	1.275362	1.507246	8 /15	1 /3	0	0
nug16a	1.407867	2.691511	7 /15	1 /3	0	0
nug16b	1.516129	1.88172	10 /15	1 /3	0	0
nug17	1.016166	0.9237875	8 /15	1 /3	0	0
nug18	1.126079	2.176166	8 /15	1 /3	0	0
nug20	0.8041505	1.037613	9 /15	2 /3	0	0
nug21	1.082855	1.121138	9 /15	1 /3	0	0
nug22	1.045606	1.483129	5 /15	1 /3	0	0
nug24	0.9174312	1.509939	9 /15	1 /3	0	0
nug25	0.8226496	0.3917379	8 /15	1 /3	0	0
nug27	1.304292	0.8024455	7 /15	2 /3	0	0

nug28	0.8646277	1.638921	9 /15	1 /3	0	0
nug30	0.5399521	1.230133	8 /15	1 /3	0	0
rou12	1.575694	2.864769	8 /15	1 /3	0	0
rou15	2.231558	0.6931858	9 /15	1 /3	0	0
rou20	0.8133362	0.8867179	7 /15	1 /3	0	0
scr12	2.055821	2.774063	10 /15	2 /3	0	0
scr15	3.917351	4.612176	7 /15	1 /3	0	0
scr20	4.015269	3.700203	8 /15	1 /3	0	0
sko42	0.4932962	0.7251876	3 /15	0 /3	0	0.00101189
sko49	0.4447105	0.4333077	3 /15	0 /3	0	0.00068417
sko56	0.5618434	1.033142	4 /15	0 /3	0	0.004875501
sko64	0.4484034	0.4879926	3 /15	0 /3	0	0.002598045
sko72	0.5260404	0.7979152	1 /15	0 /3	0	0.004980681
sko81	0.4448449	0.7611889	1 /15	0 /3	0	0.003208862
sko90	0.4985545	0.6670475	2 /15	0 /3	0	0.001765714
sko100a	0.4870989	0.4868357	0 /15	0 /3	0.0001578927	0.0002631544
sko100b	0.4208201	0.7238937	1 /15	0 /3	0	0.001442589
sko100c	0.6203983	0.5847795	1 /15	0 /3	0	0.00132556
sko100d	0.4849263	0.6409228	0 /15	0 /3	0.0001337113	0.0006551853
sko100e	0.5050844	0.7862331	2 /15	0 /3	0	6.70466e-05
sko100f	0.5192481	0.7237625	2 /15	0 /3	0	0.002724174
ste36a	4.110855	6.242564	6 /15	0 /3	0	0.0247743
ste36b	9.602994	11.38027	5 /15	1 /3	0	0
ste36c	3.150458	3.974313	6 /15	0 /3	0	0.02815668
tai10a	2.580601	4.029288	8 /15	1 /3	0	0
tai12a	2.665051	3.139408	10 /15	1 /3	0	0
tai15a	1.255218	1.444392	8 /15	1 /3	0	0
tai17a	1.435264	1.968774	7 /15	1 /3	0	0
tai20a	1.470495	1.827197	5 /15	1 /3	0	0
tai25a	0.8939256	1.885733	5 /15	1 /3	0	0
tai30a	0.9405625	0.6986971	4 /15	1 /3	0	0
tai35a	0.6004729	0.8238088	5 /15	1 /3	0	0
tai40a	0.5665595	0.7347971	0 /15	0 /3	0.0007428242	0.003170063
tai50a	0.7720127	0.5231774	0 /15	0 /3	0.00404957	0.002995467
tai60a	0.6608195	0.5340578	1 /15	0 /3	0	0.004950345
tai80a	0.7868229	0.6665341	0 /15	0 /3	0.006330605	0.00579961
tai100a	0.6683562	0.7062926	0 /15	0 /3	0.004689902	0.00636315
tai10b	3.557393	7.085558	8 /15	1 /3	0	0
tai12b	5.835181	5.762965	8 /15	1 /3	0	0
tai15b	0.3499952	0.4676546	3 /15	0 /3	0	0.001866
tai20b	8.853299	11.76788	4 /15	0 /3	0	0.01285766
tai25b	6.286803	7.595594	5 /15	0 /3	0	0.008836559
tai30b	7.373084	9.530083	4 /15	0 /3	0	0.03356596
tai35b	4.502475	5.654832	3 /15	0 /3	0	0.01249043
tai40b	6.766405	5.141299	4 /15	0 /3	0	0.02276873

tai50b	3.909013	3.245417	1 /15	0 /3	0	0.015539
tai60b	3.312533	5.761051	3 /15	0 /3	0	0.01598039
tai80b	3.206649	3.800812	0 /15	0 /3	1.299952e-05	0.01498373
tai100b	3.100179	2.969815	0 /15	0 /3	0.0002411779	0.02026677
tai150b	1.86853	2.360226	0 /15	0 /3	0.00175396	0.0155608
tai64c	0.2013153	0.008405498	6 /15	2 /3	0	0
tai256c	0.4187034	0.4196998	0 /15	0 /3	0.002471978	0.003876558
tho30	1.088109	1.258759	5 /15	1 /3	0	0
tho40	0.7353634	0.7957891	1 /15	0 /3	0	0.002993564
tho150	0.6994698	1.061721	0 /15	0 /3	0.000154425	0.002404653
wil50	0.4072435	0.312739	4 /15	0 /3	0	0.001843658
wil100	0.3144861	0.4424293	1 /15	0 /3	0	0.001259898

Table 4: Results obtained on the QAPlib for 60 seconds of runtime.

1.4.4. Discussion

Overall, we can consider the algorithms to perform quite well across the entire QAPlib benchmark. The algorithms designed for solution quality obtain in general better results than the algorithms designed for anytime behaviour. This can be explained with the fact that the combinations/configurations that yield a good anytime behaviour are probably very sensitive to instance characteristics, therefore robust when scaling the instance size, but less so when applied to different instance classes; this might be anyway a consequence of the lower number of algorithms considered (three versus fifteen).

The average percentage deviation from the optimal or best known solutions is often low. The only exception is for the `chr` and `els` instances, probably due to their very particular structure; still, several of our algorithms obtain the optimal solutions for those instances too. On many instances our algorithms are able to find the optimal or best known solutions, and when this does not happen the best algorithm finds solutions very close to the bound or optimum. From the split view of Figure 9 we notice how the instance classes for which we obtain the best results are the ones closer to random instances, while worse average results are obtained on instance classes with specific structures.

Another interesting observation is that the optimal or best solutions are usually obtained very quickly, in few seconds or even fractions of a second. Notably, for the largest QAPlib instance, `tai256c`, a solution that is worse with respect to the best known one by 0.0025% can be found by one of our algorithms in 370555 moves (0.67 seconds in our computational environment). While a longer runtime is in general beneficial to the algorithms (see Figure 8), in particular for certain instance classes, this observation suggests that a selection (or switching) strategy with early termination might indeed be more a efficient policy.

While our results seem to compare unfavourably against those reported for a recent state-of-the-art algorithm in [1], Breakout Local Search (BLS), we have

to note that BLS is a more complex hybrid stochastic local search that has been overtuned for the QAPlib (and in particular a critical parameter is set to different values for different instance classes) and that is run for a much longer runtime; the time required for finding the best solution for the large instances by BLS is often several minutes. Therefore, a direct comparison, performed in the same way as in the main paper for our set of algorithms, is neither really possible nor meaningful in this case.

We have instead observed how simple SA algorithms, designed over uniformly random instances, can be applied to a more diverse benchmark and still obtain satisfactory results in many cases.

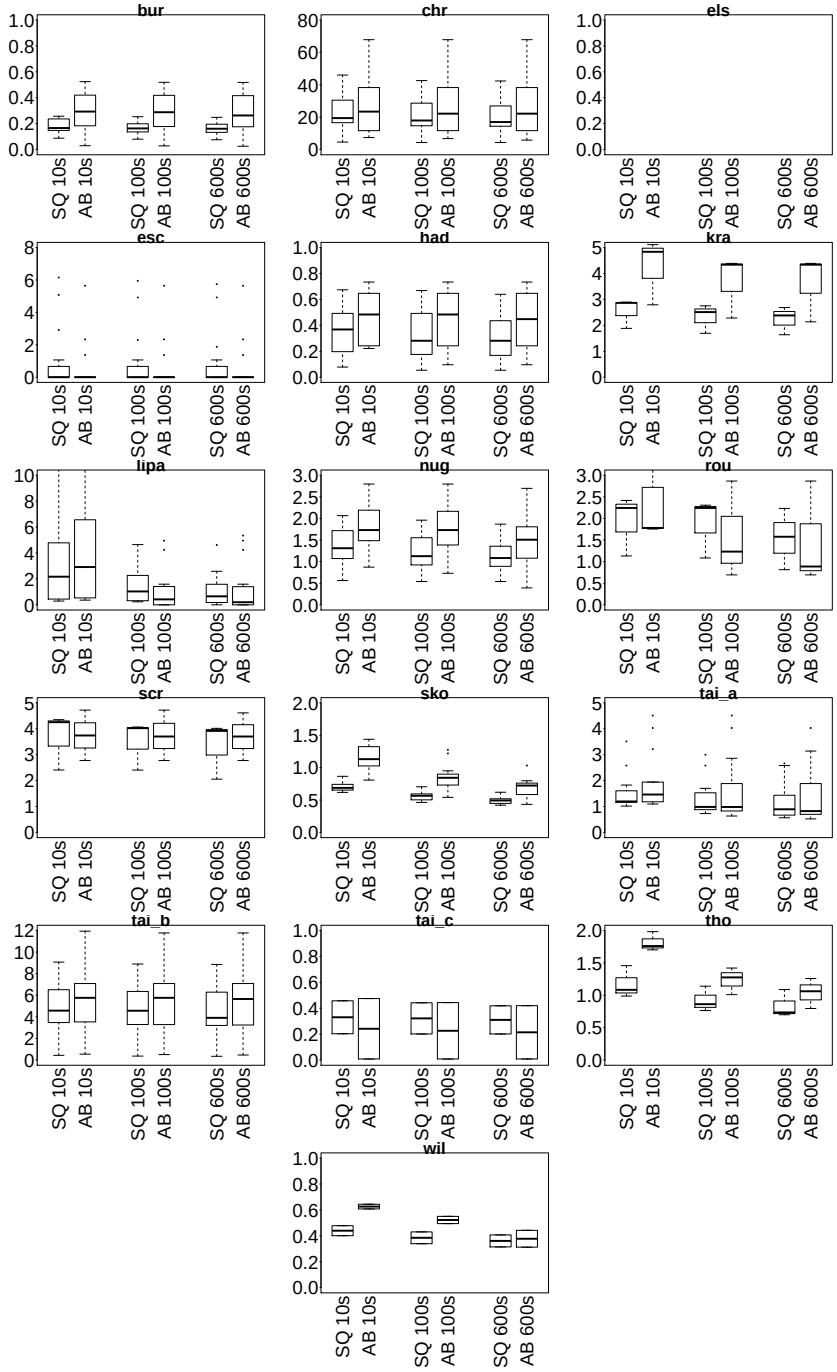

Figure 9: Average Relative Percentage Deviation (ARPD) from the optimal or best known solutions obtained by the automatically generated SA algorithms on the several instance classes of the QAPlib for 10, 100 and 600 seconds of runtime.

Algorithm	Temperature components	Exploration	Neighbourhood
`OP-IR`	**IT9**($k = 1$, $df = 5$) **AC1** **CS5**(a,b based on the termination) **TL1**($k = 1$) **TR1** **SC3**($n = 1$)	**NE1**	Exchange
`OP-SR`	**IT9**($k = 1$, $df = 5$) **AC1** **CS5**(a,b based on the termination) **TL1**($k = 1$) **TR1** **SC3**($n = 1$)	**NE1**	Insert
`OP-IO`	**IT9**($k = 1$, $df = 5$) **AC1** **CS5**(a,b based on the termination) **TL1**($k = 1$) **TR1** **SC3**($n = 1$)	**NE2**	Exchange
`OP-SO`	**IT9**($k = 1$, $df = 5$) **AC1** **CS5**(a,b based on the termination) **TL1**($k = 1$) **TR1** **SC3**($n = 1$)	**NE2**	Insert
`OS-E`	**IT6**($l = 10^4$, $k = 1$, $p = 0.2$) **AC5**($r = 0.75$) **CS2**($\alpha = 0.75$) **TL2**($k = 50$) **TR1** **SC4**($n = 15$)	**NE1**	Exchange
`OS-I`	**IT6**($l = 10^4$, $k = 1$, $p = 0.2$) **AC5**($r = 0.75$) **CS2**($\alpha = 0.75$) **TL2**($k = 50$) **TR1** **SC4**($n = 15$)	**NE1**	Insert
`CH1`	**IT1**($k = 1$) **AC3**($\phi = 001\%5$) **CS2**($\alpha = 0.999$) **TL1**($k = 1$) **TR1** **SC1**	**NE1**	Exchange
`CH5`	**IT1**($k = 1$) **AC3**($\phi = 005\%5$) **CS2**($\alpha = 0.999$) **TL1**($k = 1$) **TR1** **SC1**	**NE1**	Exchange

Table 5: Default settings for the eight algorithms of Section 6 of the main paper.

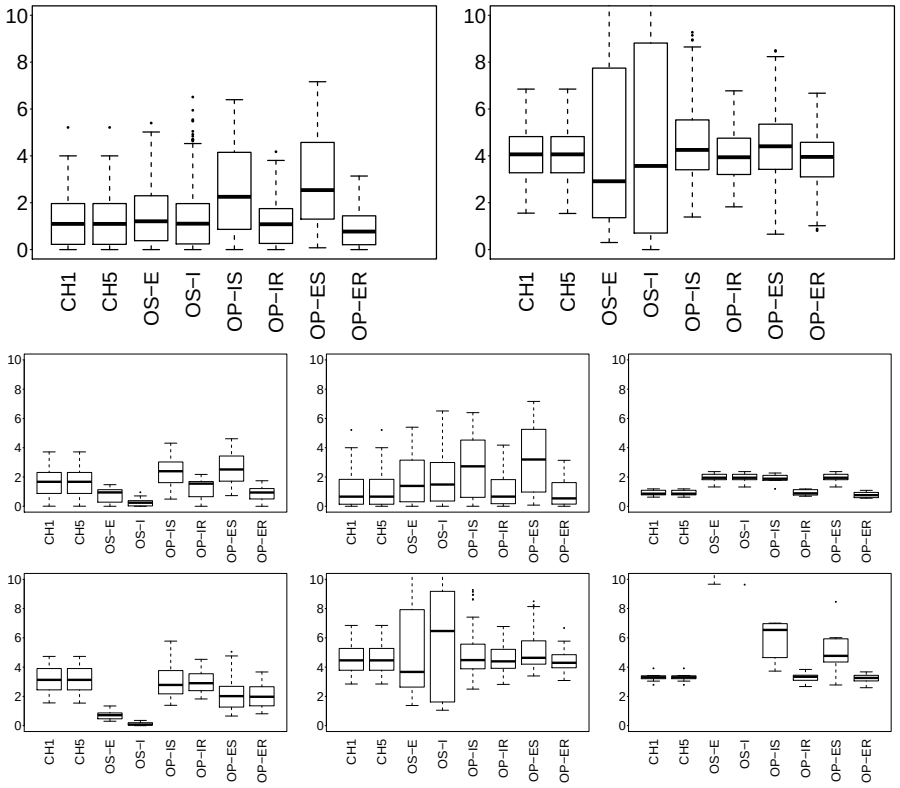

Figure 10: Average Relative Percentage Deviation (ARPD) from the best known solutions obtained by the algorithms on the whole Taillard benchmark under the MS and TCT objectives (top row, left and right plot respectively), and divided into the small, medium-size and large instances for MS (middle row) and TCT (bottom row).

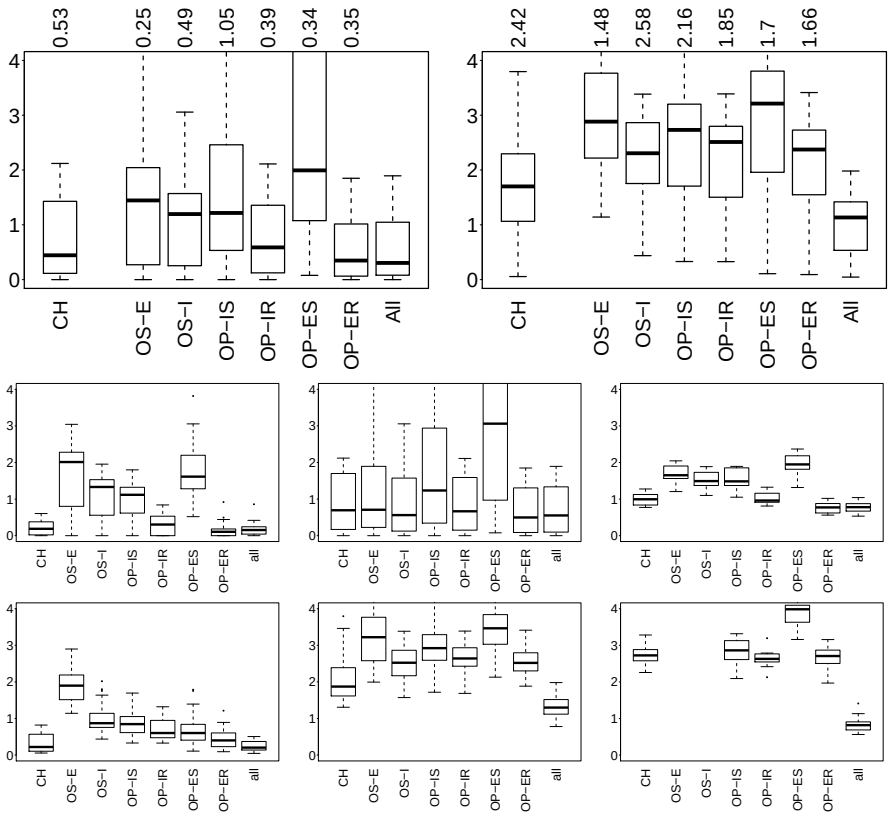

Figure 11: Average Relative Percentage Deviation (ARPD) from the best known solutions obtained by the algorithms after the tuning on the whole Taillard benchmark under the MS and TCT objectives (top row, left and right plot respectively), and divided into the small, medium-size and large instances for MS (middle row) and TCT (bottom row).

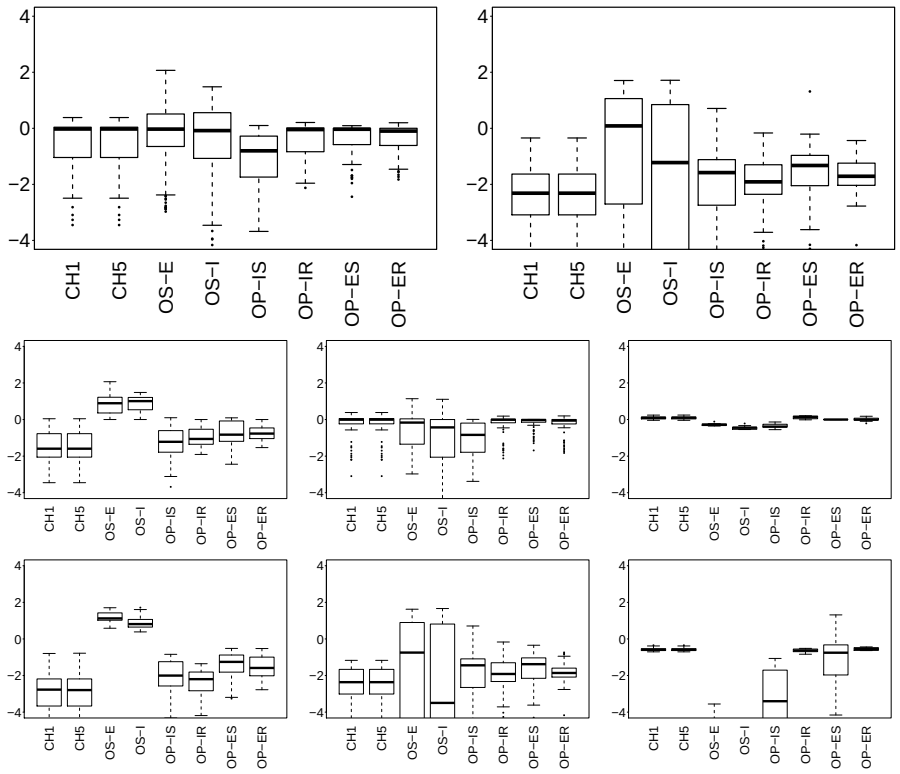

Figure 12: Improvement in terms od ARPD of the algorithms after the tuning on the whole Taillard benchmark under the MS and TCT objectives (top row, left and right plot respectively), and divided into the small, medium-size and large instances for MS (middle row) and TCT (bottom row).

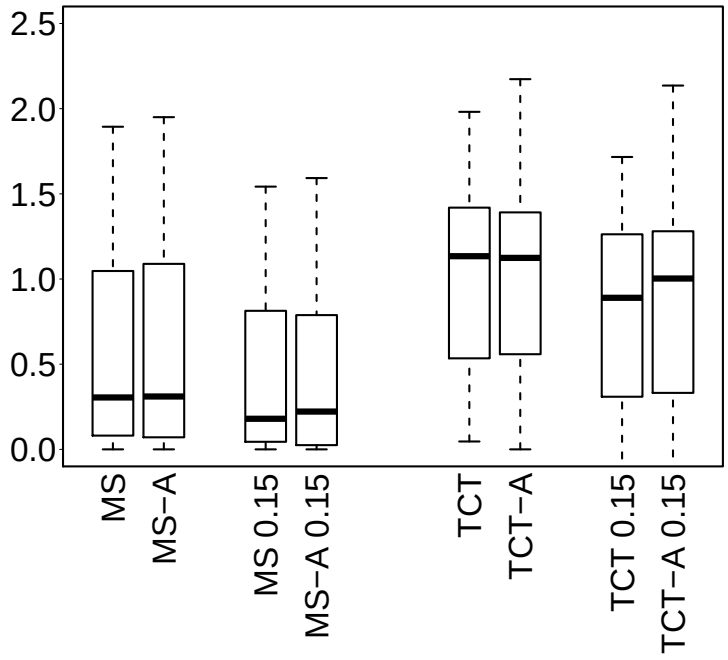

Figure 13: Results in terms od ARPD of the algorithms automatically designed for solution quality and anytime behaviour on the whole Taillard benchmark under the MS and TCT objectives. We compare the results obtained with the default runtime, and with a runtime ten times longer ($\rho = 0.15$).

2.1. Results for longer runtimes

In Figure 13 we report the results obtained using a ten-times increment in runtime ($\rho = 0.15$), compared to the runtime used in the other PFSP experiments in this work ($\rho = 0.015$). We compare both the algorithms automatically generated for solution quality, and for anytime behaviour. In both cases, for both the objectives we observe an improvement, made possible by the additional runtime. However, this improvement is not as substantial as for the QAP scenarios, partly due to the fact that for small instances good results were already consistently obtained, and partly because of limitations of the SA structure and of our set of moves. In fact, we can consider the state of the art for (these two objectives of) the PFSP the more complex, automatically generated hybrid SLS algorithms of Pagnozzi and Stützle [2].

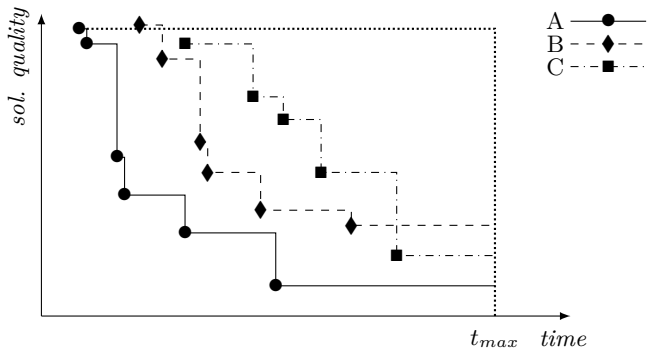

Figure 14: Example of convergence behaviour of three algorithms A, B, C, with the hypervolume defined by the convergence at a reference time t_{max} of A, the algorithm exhibiting the best anytime behaviour among the three. Algorithm C converges to a better final solution than B, but in a longer time.

3. Convergence of SA algorithms and anytime behaviour

In this section, we consider improving the convergence behaviour of SA algorithms by an automatic design approach. In particular, we automatically configure SA algorithms trying to maximize their anytime behaviour [3], that is, we require an algorithm to discover as good solutions as early as possible during the search [4]. In a nutshell, an optimization algorithm is said to have a good anytime behaviour when it rapidly improves the incumbent solution and continues improving it when given more time. In practice, having a good anytime behaviour is a desirable property for a heuristic optimization algorithm.

The tuning for anytime behaviour is modeled as a bi-objective optimization problem, in which the development of the best solution found so far over computation time is considered a non-dominated set of points. Every point in this frontier corresponds to a set of pairs $(t_i, f(s_i))$, $i \in [1, \ldots, n_i]$, each point corresponding to the time t_i where the ith time the incumbent solution has been improved and $f(s_i)$ is the corresponding solution quality; n_i is the number of times the best solution found so far has been improved. The goal of tuning for anytime behaviour is illustrated in Figure 14. Three algorithms (or three runs of the same stochastic algorithm) are run for a given amount of time t_{max}. Algorithm B converges relatively early to a solution that it cannot improve; algorithm C has a slower convergence, but it returns a better solution than B. Algorithm A instead has a better anytime behaviour than B and C: it discover a better final solution, and the whole search process shows how its convergence is quicker than the convergence of B and C. We measure the anytime behaviour in terms of the *hypervolume*, which in Figure 14 is represented by the area enclosed by the perimeter defined by the set of pairs $(t_i, f(s_i))$ of Algorithm A and by the boundaries (the dotted lines) determined by the maximum runtime and the initial solution quality: a better anytime behaviour defines a greater hypervolume. The goal of tuning for anytime behaviour hence is to maximize the hypervol-

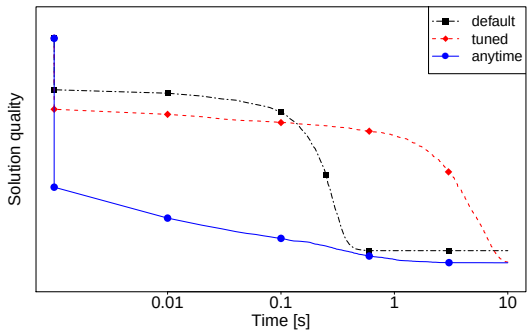

Figure 15: Example of convergence behaviour of `Tam` on a QAP random instance of size 100 and a runtime of ten seconds, in its default parameter settings (black), after the tuning (red), and after tuning for anytime behaviour (blue). Time is given in log scale.

ume. Following the procedure proposed in [3], we evaluate the configurations with respect to the hypervolume obtained by the set of non-dominated points defined by the solutions traversed during the search. Except for the evaluation of the algorithm's results, the tuning settings in this section are the same as for the tuning for solution quality of Sections 5 and 6 of the main paper.

When configuring, we sometimes observe that after tuning an algorithm improves its final solution quality but worsens its anytime behaviour. This happens because the tuning process may exploit the given execution environment, including the available running time. An example is given in Figure 15. In this plot we show the convergence behaviour of `Tam` on random QAP instances of size 100 and a runtime of ten seconds. The dashdotted black line shows the convergence of the algorithm in its original parameter settings, while the dashed red line represents the algorithm after the tuning of Section 5 of the main paper. We can see how the final solution quality of the tuned algorithm is better than the one of the algorithm using the default parameter values, but for a large part of the runtime, the default settings discover better solutions (consider that the x-axis is given in logscale). In other words, if we had stopped the algorithms earlier we would have obtained worse results with the tuned settings, than with the default ones. This sensitivity of the tuning with respect to the computational environment is an issue, because it might hinder the performance when the production environment is different from the tuning one. This may happen if the algorithms are used with a different runtime or (even when given the same wall-clock time) on a slower machine. The blue solid line shows instead the convergence of an algorithm when tuned for optimizing the anytime behaviour. When compared to the other two settings, it finds quicker good quality solutions, and then continues refining the incumbent solution.

As for the tuning with respect to the final solution quality, this approach can be used to fully exploit the choice of algorithmic components to automatically design SA algorithms that exhibit a good anytime behaviour. Here, we show

QAP Random		**QAP Structured**	
Instance	46.67%	Instance	14.59%
Temperature Restart	3.39%	**CS2** α	9.73%
TL4 k	3.38%	**TR1** t	5.11%
Cooling Scheme	3.05%	Temperature Restart	4.05%
IT6 k	2.50%	**IT3** k	4.05%
Temperature Length	2.18%	**TL6** m	3.84%
IT6 p_0	2.09%	**AC5** r	3.6%
Temperature Restart	2.08%	Acceptance Criterion	2.99%
CS11 a	2.05%	**CS3** a	2.83%
Acceptance Criterion	2.02%	**TL6** c	2.77%
PFSP-MS		**PFSP-TCT**	
Instance	52.64%	Instance	66.29%
IT5 k	10.17%	LAHC κ	5.09%
IT4 k	2.64%	**IT4** k	4.35%
Acceptance Criterion	2.32%	Acceptance Criterion	2.59%
Exploration Criterion	2.23%	Exploration Criterion α	2.34%
CS3 a	2.11%	Cooling Scheme	1.80%
TR15 k	1.73%	Temperature Length	1.13%
AC3 ϕ_{BM}	1.62%	Temperature Restart	0.85%
Temperature Restart	1.34%	**TL7** m	0.62%
TL1 k	1.34%	Initial Temperature	0.59%

Table 6: The ten most important components and numerical parameters on the four scenarios, in the tuning for anytime behaviour.

results obtained from three tunings per scenario under the same conditions of the previous experiments, with a budget of 60000 experiments and a termination condition of ten seconds for the QAP and $\rho = 0.015$ for the PFSP. To observe the scaling behaviour, a relevant scenario suited for anytime behaviour algorithms, we also show the convergence on large instances, namely of size 500 for the QAP from [5], and 800×60 for the PFSP from [6].

3.1. QAP

In Figures 17–19 we show the convergence of the automatically designed SA algorithms for QAP of Section 5 of the main paper, compared with SA algorithms automatically designed for anytime behaviour. The final solution quality is compared in Figure 16. On the random instances the convergence behaviour is very similar, with a final solution quality that is not statistically significantly different between the two sets of results (the p-value of a pairwise Wilcoxon test is 0.8896). That is, the SA algorithms automatically designed for solution quality for this scenario already exhibit a very good anytime behaviour, even for the larger instances.

On the structured instances, instead, the results are more variable. In two cases the anytime behaviour of the SA algorithms was good, though noticeably

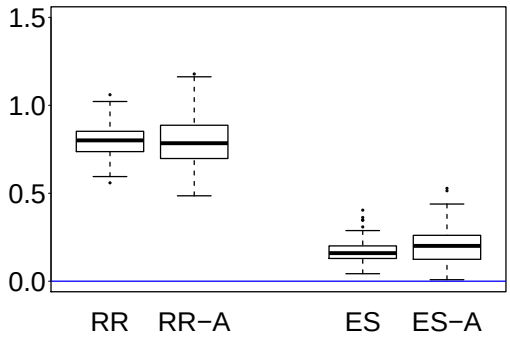

Figure 16: Comparison of the final solution quality in terms of ARPD obtained by automatically-designed SA algorithms of Section 5 of the main paper and the SA algorithms automatically designed for anytime behaviour for the QAP problem (random instances on the left, structured instances on the right).

different, on the instances of size 60 and 100, but much worse on the instance of size 500, for which the convergence was much slower and failed to return good solutions in the given ten seconds of runtime. In the third case, instead, the anytime behaviour was very good, and on the size 500 instances the SA of Section 5 of the main paper converged to significantly better solutions than the SA designed for anytime behaviour, continuing discovering better solutions during the whole runtime.

3.2. PFSP

In Figure 20 we show the quality, in terms of ARPD, of the final solutions obtained by SA algorithms automatically designed for anytime behaviour for the PFSP, compared with the three automatically designed SA algorithms of Section 6 of the main paper obtained with the same random seeds. The results are divided into the 12 subclasses of different size that compose the Taillard benchmark; the upper plot shows the results on the Makespan objective, while the bottom plot contains the results for the Total Completion Time objective. The results are clearly similar, in particular for the MS objective, while for the TCT objective in most subclasses the SAs specifically designed for solution quality obtain slightly better results. The convergence behaviour is reported in Figures 21 and 22 for the MS and TCT objectives respectively.

On both objectives the anytime behaviour of the SA algorithms of Section 6 of the main paper is again good, sometimes even slightly better than the SAs designed for anytime behaviour (which, in turn, have a more regular convergence). On the larges instances the search takes more time to discover a good solution, but from that point the behaviour is similar to the SAs tuned for anytime behaviour.

It is very easy to observe that a good anytime behaviour corresponds to stricter conditions for accepting worsening moves. An analysis of the selected

components and numerical parameters shows that the outcome of the automatic design for anytime behaviour is a set of stricter conditions for the acceptance of worsening moves, with respect to the automatic design for final solution quality: a much lower initial temperature, and a tighter bound for the Bounded Metropolis acceptance criterion. For the TCT objective, the same effect is obtained by a much shorter tenure of the LAHC acceptance criterion.

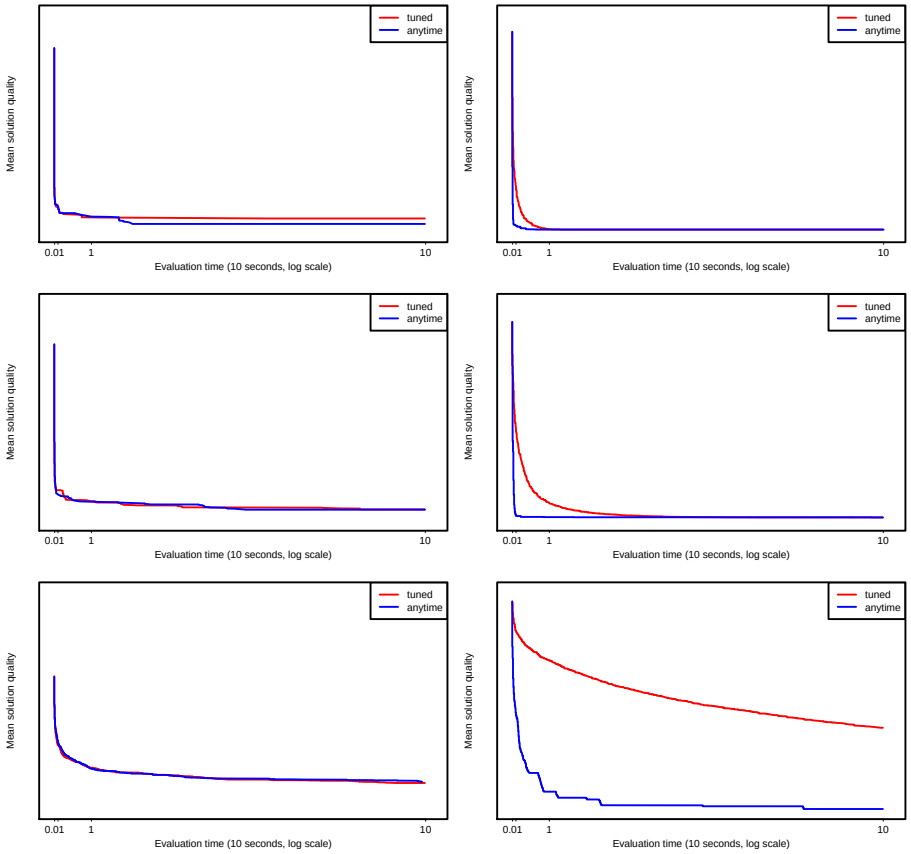

Figure 17: Convergence behaviour of automatically designed SAs for final solution quality and anytime behaviour on random and structured instances (left and right column) respectively on instances `taillardrr.60.0.dat`, `taillardrr.100.1.dat` and `taillardrr.500.0.dat` (from top to bottom), and `EuclideanStructured.1020000.n60.sp72.00.dat`, `EuclideanStructured.1000.n100.sp72.00.dat` and `EuclideanStructured.1256751137.n500.K60.m10.A20.00.B4.00.sp10.00.dat` (from top to bottom).

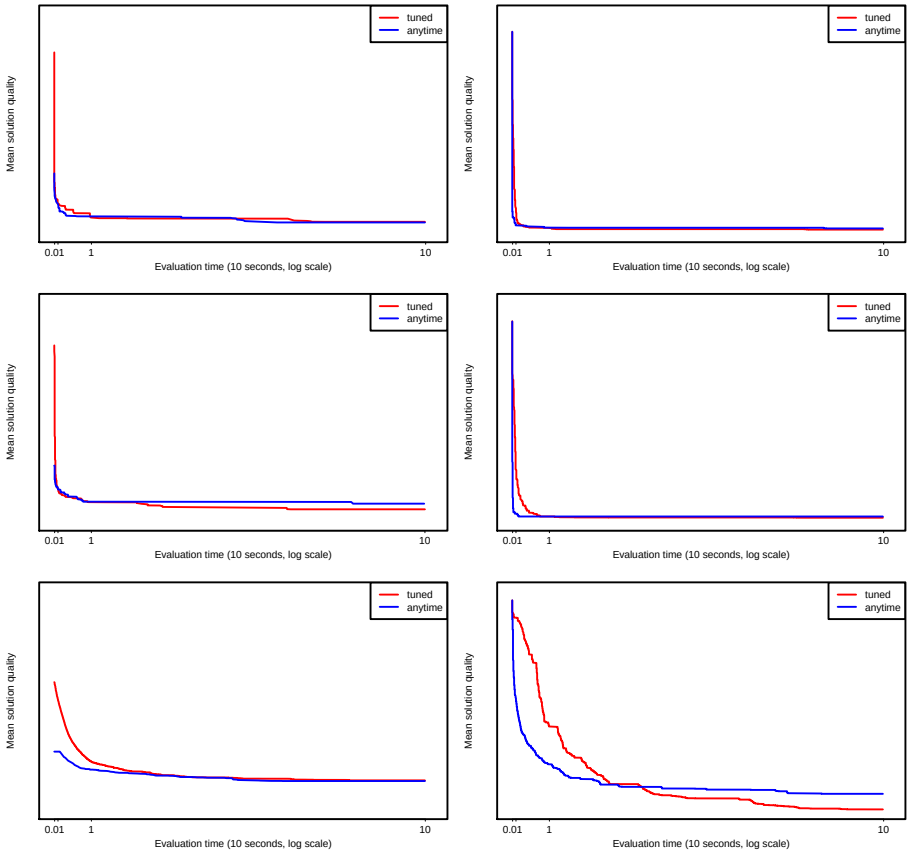

Figure 18: Convergence behaviour of automatically designed SAs for final solution quality and anytime behaviour on random and structured instances (left and right column) respectively on instances `taillardrr.60.0.dat`, `taillardrr.100.1.dat` and `taillardrr.500.0.dat` (from top to bottom), and `EuclideanStructured.1020000.n60.sp72.00.dat`, `EuclideanStructured.1000.n100.sp72.00.dat` and `EuclideanStructured.1256751137.n500.K60.m10.A20.00.B4.00.sp10.00.dat` (from top to bottom).

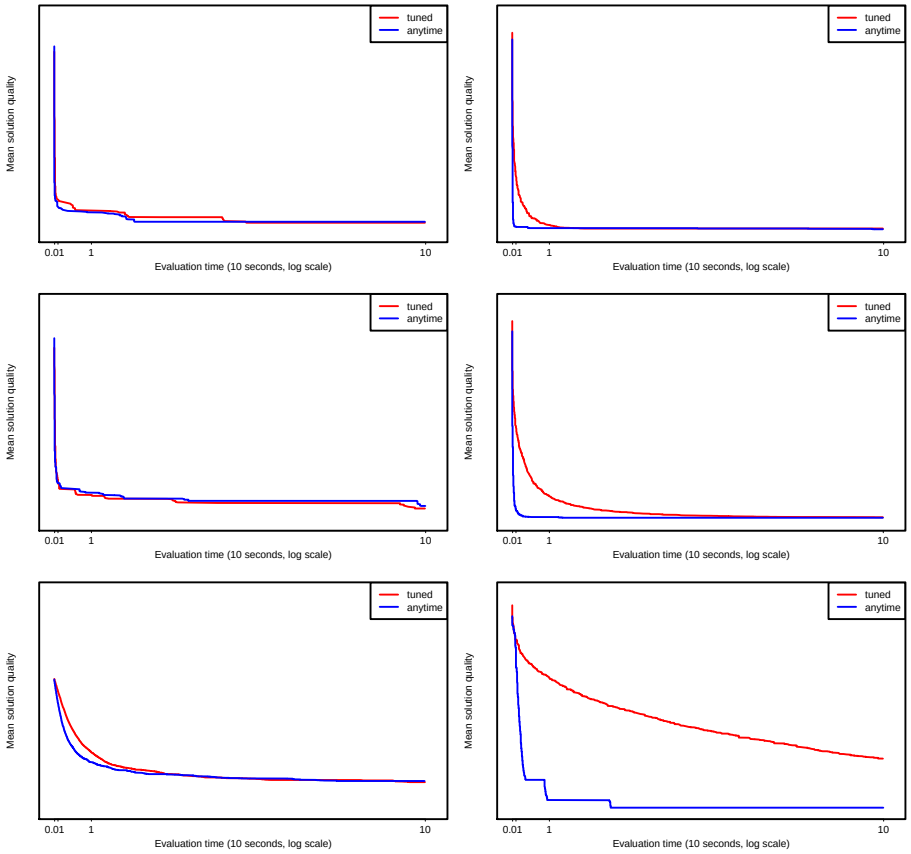

Figure 19: Convergence behaviour of automatically designed SAs for final solution quality and anytime behaviour on random and structured instances (left and right column) respectively on instances `taillardrr.60.0.dat`, `taillardrr.100.1.dat` and `taillardrr.500.0.dat` (from top to bottom), and `EuclideanStructured.1020000.n60.sp72.00.dat`, `EuclideanStructured.1000.n100.sp72.00.dat` and `EuclideanStructured.1256751137.n500.K60.m10.A20.00.B4.00.sp10.00.dat` (from top to bottom).

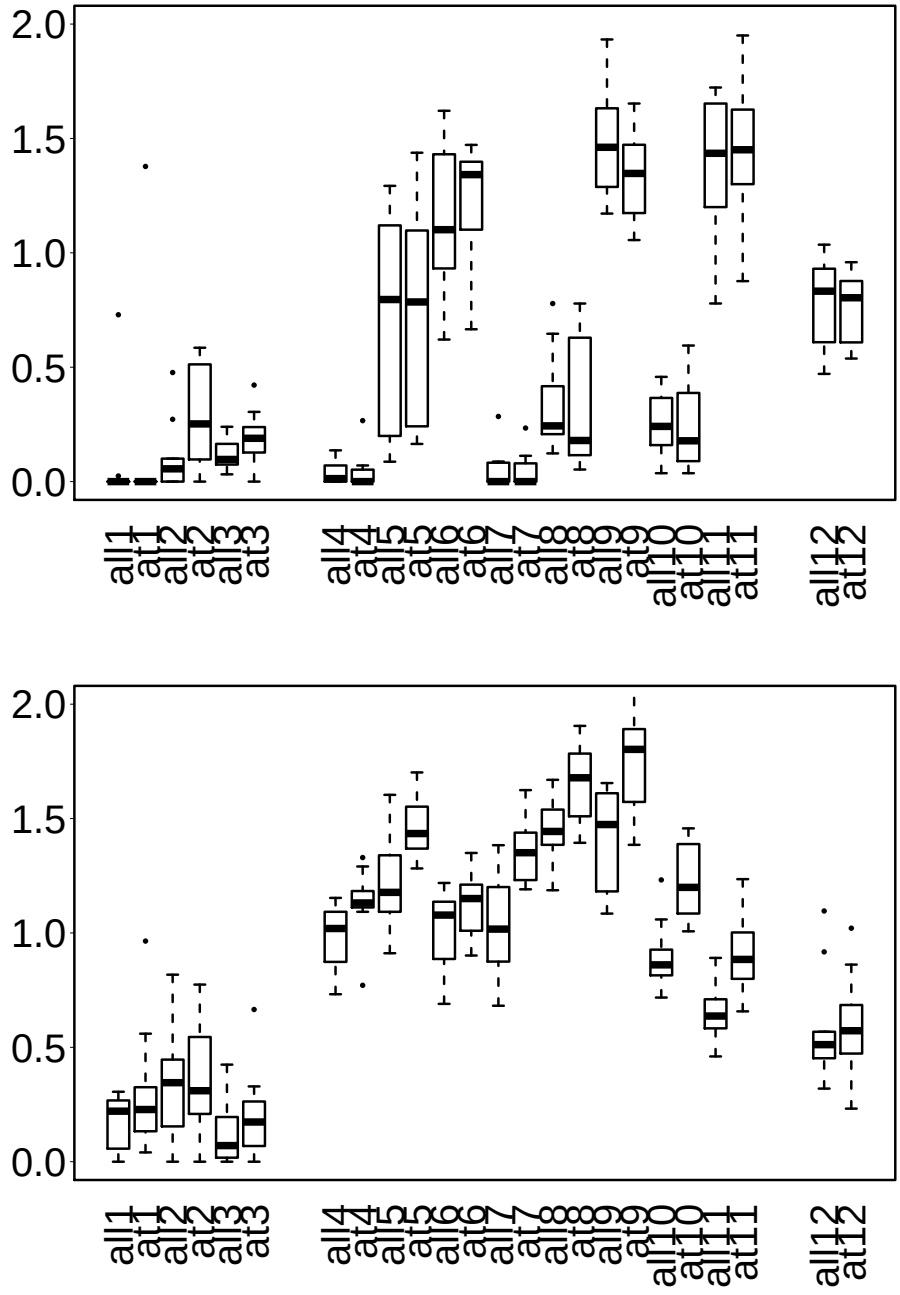

Figure 20: Comparison of the final solution quality in terms of ARPD obtained by automatically-designed SA algorithms of Section 6 of the main paper and the SA algorithms automatically designed for anytime behaviour for the PFSP problem (MS objective on top, TCT objective on bottom). The results are divided per instance subclass.

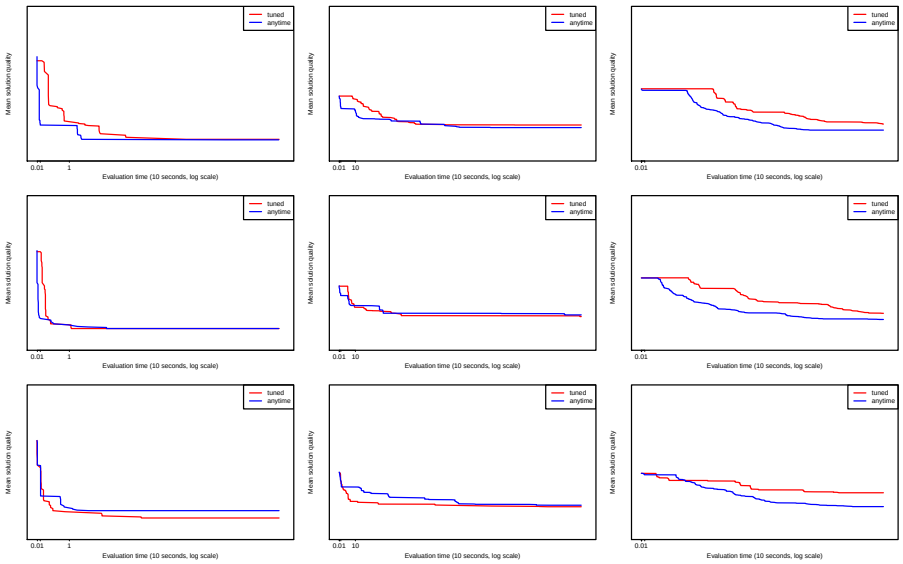

Figure 21: Convergence behaviour of automatically designed SAs for final solution quality and anytime behaviour on the PFSP with MS objective. From top to bottom we show three different tunings. Left to right, we show the convergence on `Ta041`, `Ta120` and `VFR800_60_1`.

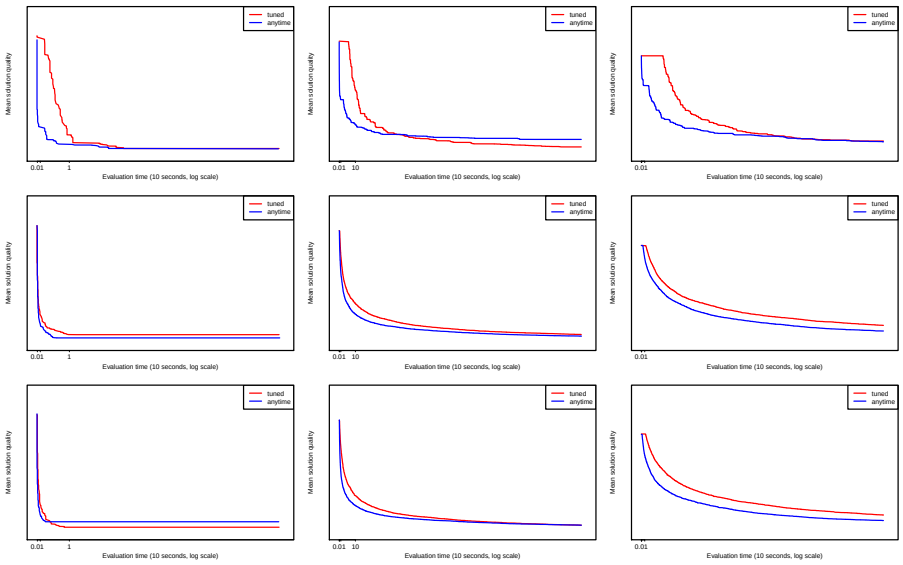

Figure 22: Convergence behaviour of automatically designed SAs for final solution quality and anytime behaviour on the PFSP with TCT objective. From top to bottom we show three different tunings. Left to right, we show the convergence on `Ta041`, `Ta120` and `VFR800_60_1`.

4. Composition of automatically generated SA algorithms

We analyze the components selected by the best configuration found in the 15 executions of the automatic design process, starting with the algorithms for random QAP instances. We observe a majority of occurrences of the Metropolis condition (in ten cases, with two additional occurrences of the Bounded Metropolis criterion), with the Geometric acceptance chosen twice and the RTR acceptance once. We observe six different cooling schemes, namely **CS2** appearing 7 times, **CS11** 3 times, **CS5** twice and **CS6** once. In this scenario we have also 8 occurrences of the sequential exploration scheme **NE2**, all combined with Metropolis-based acceptance criteria and initial temperature schemes based on a preliminary random walk. In six cases the cooling scheme of choice is **CS2** with a very steep cooling behaviour (α values mostly between 0.12 and 0.45) and **TL5** as temperature length. In the remaining two cases, we observe the temperature band cooling scheme **CS11**, that is, a non-monotonically decreasing scheme. The remaining algorithms feature the **NE3** exploration scheme, that performs a limited local search in the neighbourhood and selects for comparison the best solution encountered; the other components in this case differ more widely between algorithms, suggesting that when using **NE3**, it is this component the most important one, as it already does a preliminary screening of the search space and evaluates only the most promising candidate solutions.

On the structured QAP instances, the Geometric cooling scheme **CS2** is selected in 12 out of 15 cases, with the α coefficients ranging around 0.8. In all cases the exploration criterion selected is the sequential one **NE2**, corroborating the observation of Sect. 5 of the main paper). In eight cases the traditional Metropolis condition is the preferred choice for the acceptance condition; in three cases instead Threshold acceptance is selected. In the remaining four cases the choice is LAHC.

Now we analyse the composition of the fifteen automatically generated SA algorithms for the MS objective. The first observation is that, contrarily to the QAP case, and aligning with the findings of the tuned algorithms, the random exploration is chosen for all the algorithms. Eight of them use a Q8-7 cooling, three its original version of Lundy and Mees, while three more employ a Geometric Cooling; the remaining one is not considered, as the acceptance chosen is LAHC. Thirteen algorithms use a Metropolis-based acceptance criterion. The importance analysis shows 20% importance for the exploration criterion, and 14.5% for the instance; next, there are the numerical parameters for the Lundy-Mees and Q8-7, and Geometric cooling schemes. The acceptance criterion features a 4% of importance, still among the most important components/parameters, but not as much as in the other cases.

Out of the fifteen automatically generated SAs for the TCT objective, twelve feature the Late Acceptance criterion; the κ parameter in eleven cases is around 150 (ranging from 99 to 185), with one outlier case in 5616. The other three cases instead use a Bounded Metropolis criterion, set to immediately discard solutions whose cost is 0.53 to 0.68% higher than the incumbent; these three algorithms are all paired with a Geometric cooling scheme. In summary, the best

algorithms for the TCT objective are quite conservative in accepting worsening moves. Again, all the fifteen SAs employ a random exploration.

References

[1] U. Benlic, J.-K. Hao, Breakout local search for the quadratic assignment problem, Applied Mathematics and Computation 219 (9) (2013) 4800–4815.

[2] F. Pagnozzi, T. Stützle, Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems, Tech. Rep. TR/IRIDIA/2018-005, IRIDIA, Université Libre de Bruxelles, Belgium (Apr. 2018).
URL http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2018-005.pdf

[3] M. López-Ibáñez, T. Stützle, Automatically improving the anytime behaviour of optimisation algorithms, European Journal of Operational Research 235 (3) (2014) 569–582. doi:10.1016/j.ejor.2013.10.043.

[4] S. Zilberstein, Using anytime algorithms in intelligent systems, AI Magazine 17 (3) (1996) 73–83.

[5] M. S. Hussin, T. Stützle, Tabu search vs. simulated annealing for solving large quadratic assignment instances, Computers & Operations Research 43 (2014) 286–291.

[6] E. Vallada, R. Ruiz, J. M. Framiñán, New hard benchmark for flowshop scheduling problems minimising makespan, European Journal of Operational Research 240 (3) (2015) 666–677. doi:10.1016/j.ejor.2014.07.033.