Evaluating balancing on social networks through Correlation Clustering problems - *Additional data*

1. Additional experimental data

1.1. Sequential ILS vs. sequential GRASP

We compared both SeqGRASP and SeqILS, in their best configurations.

When applying the set of completely random instances (iii) as input, we observe the superiority of SeqILS over SeqGRASP. As Table 1 shows, for 36 random instances with $200 \leq n \leq 600$, SeqILS running time was strictly better in 30 of these instances, while SeqGRASP found the target solution value (column Target I(P)) earlier for 6 instances. Still, the average time to target of ILS was smaller than GRASP's: on average, ILS is 5 times faster than GRASP.

Additionally, when applying both metaheuristics to solve the random instances with predefined community structure in (iv), SeqILS was, on average, superior to SeqGRASP in execution time. As shown in Table 2, an analysis of the gap in average time to target (column Gap time) between SeqILS and SeqGRASP indicates the superiority of SeqILS in 28 of 44 instances. On average, SeqILS is more than 2 times faster than SeqGRASP to find the specified target solution values (column Target I(P)).

1.2. Sequential vs. Parallel GRASP

Since the Slashdot instances in (i) have shown the limits of the sequential version of GRASP, we have used them to assess the performance of the parallel GRASP algorithm. Three different solution methods were applied:

- sequential GRASP metaheuristic ($SeqGRASP$, using only one processor core);
- independent parallel algorithm with sequential local search ($ParGRASP/SeqVND$);
- parallel GRASP with parallel local search ($ParGRASP/ParVND$).

We conducted several experiments to find the optimal number of processes to be used in the parallel algorithms. This setting is closely related to the hardware configuration of the computer cluster used in the experiments. As previously explained in the article, since each machine has 2 quad-core CPUs (8 processor cores), it can host 8 processes running in parallel. In $ParGRASP/ParVND$, we chose to group each GRASP master process together with its corresponding VND search slaves, in order to maximize the performance of message exchange between related processes. As seen in Figure 2, the parallel procedures with the best efficiency were:

- independent parallel algorithm with sequential local search ($ParGRASP/SeqVND$), using 8 cores;

				ILP		Target	SeqGRASP		SeqILS			T(GRASP) /
n	**e**	**d**	**d-**	**BestSol**	**Time**	**I(P)**	**AvgTime**	**CI**	**AvgTime**	**CI**	**Gap time**	**T(ILS)**
200	3980	0.1	0.2	**796**	812.28	792	0.01	0.00	0.01	0.00	0.00	0.89
			0.5	**1990**	703.44	1212	1.94	0.79	1.25	0.57	-0.69	1.55
			0.8	**3184**	1,289.60	406	3.50	1.53	0.61	0.14	-2.89	5.71
	7960	0.2	0.2	**1592**	994.13	1592	0.01	0.00	0.01	0.00	0.00	0.98
			0.5	**3980**	1,001.72	2852	5.03	1.64	2.44	0.93	-2.60	2.07
			0.8	**6368**	2,988.93	1098	14.96	5.01	1.31	0.21	-13.65	11.42
	19900	0.5	0.2	**3980**	2,386.59	3980	0.01	0.00	0.01	0.00	0.00	1.10
			0.5	**9606**	1,687.30	8144	4.37	2.54	2.86	1.38	-1.50	1.53
			0.8	**3824**	2,395.06	3350	11.82	4.46	2.19	0.53	-9.63	5.39
	31840	0.8	0.2	6368	1,449.56	6368	0.01	0.00	0.01	0.00	0.00	1.11
			0.5	**15920**	2,272.33	13604	10.55	4.04	11.28	9.39	0.73	0.94
			0.8	**6236**	2,108.50	5698	38.76	12.92	2.28	0.46	-36.48	17.00
400	15960	0.1	0.2	**3192**	7,200.00	3192	0.01	0.00	0.01	0.00	0.00	0.99
			0.5	**7980**	7,200.00	5750	13.16	5.91	6.37	3.97	-6.79	2.07
			0.8	**12768**	4,675.04	2210	26.67	9.02	3.01	0.66	-23.65	8.85
	31920	0.2	0.2	**6384**	4,711.29	6384	0.02	0.00	0.02	0.00	0.00	1.11
			0.5	**15960**	4,819.54	12764	32.10	13.72	10.44	5.45	-21.66	3.08
			0.8	**25536**	7,200.00	5190	29.84	10.09	3.20	0.54	-26.65	9.34
	79800	0.5	0.2	**15960**	3,921.77	15960	0.03	0.00	0.03	0.00	0.00	1.04
			0.5	**39900**	7,200.00	34744	33.24	21.13	14.33	8.52	-18.91	2.32
			0.8	**63840**	4,970.73	14490	62.23	22.03	6.32	1.41	-55.91	9.84
	127680	0.8	0.2	**25536**	7,200.00	25536	0.04	0.00	0.04	0.00	0.00	1.03
			0.5	**63840**	7,200.00	57356	77.12	61.04	15.89	8.82	-61.24	4.86
			0.8	**102144**	7,200.00	23926	109.30	34.33	8.18	1.47	-101.12	13.37
600	35940	0.1	0.2	-	7,200.00	7188	0.03	0.00	0.03	0.00	0.00	0.94
			0.5	-	7,200.00	13814	34.30	10.27	10.56	5.04	-23.74	3.25
			0.8	-	7,200.00	5496	60.01	21.99	6.27	1.06	-53.74	9.57
	71880	0.2	0.2	-	7,200.00	14376	0.04	0.00	0.04	0.00	0.00	0.98
			0.5	-	7,200.00	30024	80.22	59.71	21.91	11.28	-58.31	3.66
			0.8	-	7,200.00	12364	119.48	49.82	9.78	2.60	-109.70	12.22
	179700	0.5	0.2	-	7,200.00	35940	0.07	0.01	0.07	0.00	0.00	1.04
			0.5	-	7,200.00	80512	249.22	196.25	58.82	39.03	-190.40	4.24
			0.8	-	7,200.00	33552	93.98	36.65	12.46	3.63	-81.52	7.54
	287520	0.8	0.2	-	7,200.00	57504	0.10	0.01	0.10	0.01	0.00	1.03
			0.5	-	7,200.00	131892	171.05	174.45	24.91	9.77	-146.14	6.87
			0.8	-	7,200.00	54886	209.53	72.35	17.89	4.12	-191.64	11.71
Average				**-**	**-**	**-**	**41.47**	**-**	**7.08**	**-**	**-34.38**	**4.74**

Table 1: ILP, sequential GRASP (SeqGRASP) and sequential ILS (SeqILS) results for random instances in (iii). Instances solved to optimality by ILP formulation are marked with bold values in column (ILP - BestSol); in the other case this column exhibits the value of the best integer solution found in the time limit. Target I(P) is the target imbalance / solution value used as stopping criterion for each algorithm. AvgTime is the average execution time spent (in seconds) by each algorithm to reach the specified target solution value, after 25 independent executions. CI is the 95% confidence interval of the execution time, for each algorithm. Gap Time is the gap between SeqILS and SeqGRASP execution times. T(GRASP)/T(ILS)= [AvgTime(SeqGRASP)/AvgTime(SeqILS)].

Instance								SeqGRASP		SeqILS			T(GRASP)
c	**n**	$\|E\|$	**k**	p_{in}	p_-	p_+	**Target I(P)**	**Avg Time**	**CI(Time)**	**Avg Time**	**CI(Time)**	**Gap Time**	**/ T(ILS)**
4	16	64	16	0.2	0	0	0	0.73	0.26	0.63	0.33	-0.10	1.16
				0.7	0	0	0	0.03	0.01	0.08	0.04	0.05	0.38
4	32	128	20	0.7	0.6	0.6	80.8669	0.67	0.20	0.82	0.34	0.15	0.82
			24	0.7	0.2	0.2	62.8762	1.58	0.56	0.47	0.33	-1.11	3.34
4	64	256	24	0.7	0.2	0.2	99.8465	3.53	1.15	1.15	0.49	-2.38	3.07
4	64	256	32	0.8	0	0	0	1.12	0.41	0.52	0.36	-0.59	2.13
						0.2	86.2275	2.13	0.72	0.85	0.36	-1.28	2.50
						0.4	173.2103	0.69	0.30	0.69	0.30	0.00	1.00
						0.6	166.9959	0.01	0.00	0.18	0.08	0.17	0.08
						0.8	81.5059	0.01	0.00	0.09	0.03	0.08	0.06
						1	0	0.00	0.00	0.05	0.00	0.05	0.07
					0.2	0	49.6108	7.11	3.09	1.27	0.58	-5.84	5.61
						0.2	121.3758	4.06	1.61	0.71	0.28	-3.35	5.72
						0.4	191.5338	1.76	0.58	1.13	0.53	-0.62	1.55
						0.6	224.8849	0.35	0.15	1.42	0.73	1.06	0.25
						0.8	190.6345	1.09	0.38	1.09	0.39	-0.01	1.01
						1	142.315	0.11	0.04	0.59	0.27	0.48	0.19
					0.4	0	84.1122	0.87	0.28	1.61	1.25	0.74	0.54
						0.2	145.6284	7.23	2.69	2.42	1.02	-4.81	2.99
						0.4	202.3622	3.04	1.45	3.13	1.50	0.09	0.97
						0.6	253.4543	0.09	0.02	3.01	1.26	2.92	0.03
						0.8	251.4013	0.16	0.06	3.08	1.20	2.92	0.05
						1	212.095	1.19	0.41	2.23	1.07	1.05	0.53
					0.6	0	76.3863	0.15	0.03	0.59	0.35	0.44	0.25
						0.2	153.0779	6.59	1.99	4.60	1.68	-1.99	1.43
						0.4	193.5687	8.84	5.34	2.64	1.82	-6.21	3.36
						0.6	235.7613	0.36	0.10	6.47	2.93	6.11	0.06
						0.8	235.2052	10.06	4.57	6.34	2.76	-3.73	1.59
						1	270.1083	5.55	2.92	3.98	1.77	-1.57	1.40
					0.8	0	55.4669	0.08	0.01	0.27	0.02	0.19	0.30
						0.2	97.2707	10.30	4.82	2.03	0.82	-8.27	5.06
						0.4	140.8	13.23	9.35	3.22	1.45	-10.00	4.10
						0.6	187.4619	18.12	6.95	3.82	1.64	-14.30	4.74
						0.8	259.2399	24.09	9.48	6.03	3.00	-18.07	4.00
						1	269.8675	7.46	2.68	3.34	2.53	-4.12	2.24
					1	0	0	0.01	0.00	0.16	0.01	0.15	0.09
						0.2	27.2516	15.83	5.27	2.47	1.13	-13.36	6.41
						0.4	67.7761	6.41	1.96	1.18	0.52	-5.23	5.44
						0.6	117.656	15.69	8.45	3.14	1.29	-12.55	5.00
						0.8	174.2849	13.34	8.10	4.31	1.93	-9.03	3.10
						1	232.5436	10.16	5.03	4.43	2.36	-5.73	2.29
4	96	384	24	0.7	0.2	0.2	160.0418	11.08	5.61	1.06	0.61	-10.02	10.43
4	128	512	24	0.7	0.2	0.2	221.71	16.62	7.20	8.32	4.03	-8.30	2.00
25	30	750	20	0.6	0.3	0.3	414.3783	33.16	12.16	3.96	0.74	-29.20	8.37
Average							-	**6.02**	-	**2.26**	-	**-3.75**	**2.40**

Table 2: Sequential GRASP (SeqGRASP) and sequential ILS (SeqILS) CC results for random instances with predefined community structure in (iv). Target I(P) is the target imbalance / solution value used as stopping criterion for each algorithm. Avg Time is the average execution time (in seconds) spent by each algorithm to reach the specified target solution value, after 25 independent executions. CI(Time) is the 95% confidence interval of the execution time, for each algorithm. Gap Time is the gap between SeqILS and SeqGRASP average execution times. T(GRASP)/T(ILS)= [AvgTime(SeqGRASP)/AvgTime(SeqILS)].

- parallel GRASP with parallel local search ($ParGRASP/ParVND$), using 8 cores, where 2 cores run GRASP master processes and 6 cores run VND search slave processes, 3 for each master.

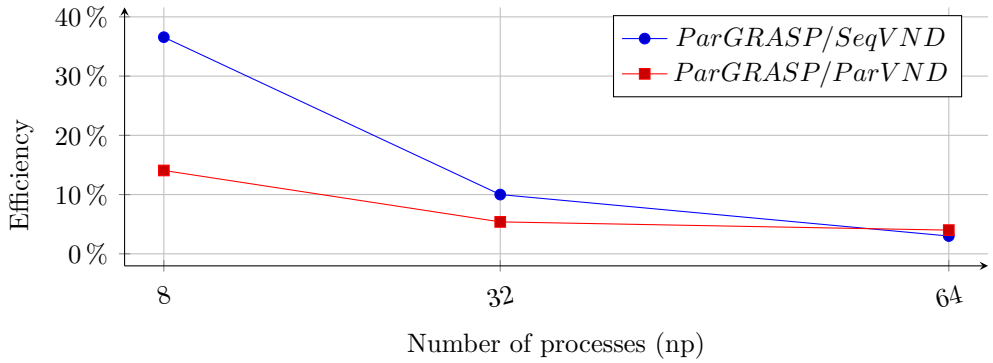

Figure 1: Average efficiency of parallel GRASP with sequential local search ($ParGRASP/SeqVND$) and parallel GRASP with parallel local search ($ParGRASP/ParVND$) when solving Slashdot-based signed graphs, after 25 independent executions.

The parameters of the procedures used in the parallel approaches of GRASP are the same used for testing the sequential algorithms, except for the number of GRASP iterations. In $ParGRASP/SeqVND(8)$, the number of iterations is reduced from 400 to 50. This is due to the fact this parallel algorithm executes 8 GRASP procedures at the same time, which provides enough variability to make the 50-iteration parallel GRASP results quality-equivalent to the 400-iteration sequential GRASP. In $ParGRASP/ParVND(8)$, 2 GRASP master processes run 200 iterations each. There are also 6 VND search slaves, 3 for each master.

Note that, in order to compare the sequential and parallel procedures, the following additional stopping criterion was applied: all procedures stop whenever the specified target solution value [Target I(P)] is found.

1.3. Sequential vs. Parallel GRASP

In this section, the Parallel GRASP algorithm results [$ParGRASP/SeqVND(8)$ and $ParGRASP/ParVND(8)$] are compared to the sequential version ($SeqGRASP$).

As seen in Table 3, the algorithm with the best efficiency was $ParGRASP/SeqVND$ (8 cores from one machine with 2 quad-core CPUs), with an average speed-up of 2.93 (minimum of 0.97 and maximum of 4.11) and efficiency of 37% (minimum of 12% and maximum of 51%). Remark that lower speed-up and efficiency values are related to smaller instances, whose solution takes just a few seconds. Also, we did not obtain linear speed-ups because of the random nature of the heuristics. The sequential algorithm reached the target solution values proportionately faster than their parallel counterparts.

The execution times of $ParGRASP/ParVND(8)$ were in average higher than the execution times of $ParGRASP/SeqVND(8)$. Therefore $ParGRASP/ParVND$, using the

Instance	SeqGRASP			ParGRASP/SeqVND(8)			ParGRASP/ParVND(8)		
	Target I(P)	Avg Time	CI(Time)	Avg Time	CI(Time)	Speedup	Avg Time	CI(Time)	Speedup
200	45	1.82	0.04	1.88	0.30	0.97	1.35	0.17	1.36
300	54	3.54	0.16	1.70	0.33	2.09	2.41	0.17	1.47
400	57	4.88	0.19	1.53	0.22	3.19	3.41	0.19	1.43
600	109	8.78	0.31	3.17	0.39	2.77	5.51	0.35	1.59
800	240	18.54	0.95	4.96	0.34	3.74	11.71	1.12	1.58
1000	600	32.80	1.40	7.98	0.75	4.11	23.62	1.99	1.39
2000	2187	169.50	19.02	41.83	6.84	4.05	263.74	26.29	0.64
4000	6203	751.70	94.42	218.89	68.64	3.43	2,049.40	199.34	0.37
8000	16083	2,891.84	599.95	1,152.27	534.20	2.51	3,924.46	482.53	0.74
10000	20586	4,589.55	660.33	1,918.06	958.57	2.39	6,654.05	448.19	0.69
Average						**2.93**			**1.13**

Table 3: CC results obtained on Slashdot signed graphs by the use of the following approaches: Sequential GRASP ($SeqGRASP$), Parallel GRASP with sequential VND ($ParGRASP/SeqVND(np)$) and Parallel GRASP with parallel VND ($ParGRASP/ParVND(np)$), where np is the number of processes in parallel. Target I(P) is the target imbalance value used as stopping criterion for each algorithm. Avg Time is the average execution time spent (in seconds) by each algorithm to reach the specified target solution value, after 25 independent executions. CI(Time) is the 95% confidence interval of the execution time, for each algorithm.

same number of processes, presented a speedup of x1.13 and an average efficiency of only 14%.

We also ran both parallel algorithms to solve the random networks in (iii) and the results shown in Figure 2 confirm the superiority of $ParGRASP/SeqVND(8)$, with average efficiency of 40% (minimum of 28% and maximum of 56%), while $ParGRASP/ParVND(8)$ presented a maximum efficiency of only 12%. This degradation in speedup and efficiency is somewhat expected, since the $ParallelLocalSearch$ algorithm causes an overhead in the number of messages exchanged between processes.[1] Therefore, even in random networks, where higher edge density and complex structure increase the time spent by local search, the additional computational resources required to run parallel local search with message-passing are not worth the available acceleration and efficiency brought by this procedure.

The obtained computational results indicate that our independent parallel GRASP metaheuristic is an efficient approach for the heuristic solution of the CC problem, with $ParGRASP/SeqVND$ with 8 processes being the fastest configuration for networks of up to $10,000$ vertices.

1.4. Parallel ILS vs. parallel GRASP

The last question that comes to mind is knowing which parallel metaheuristic presents the best efficiency and possibly the best solution values. First, when comparing the

[1]Please note that the loss of efficiency when using parallel local search is not caused by the lack of work for the VND search slaves. For example, when solving Slashdot instance of size $n = 10000$, local search performs more than 600 million neighborhood evaluations, and when solving random instances with $n = 600$, this number rises to 700 million.

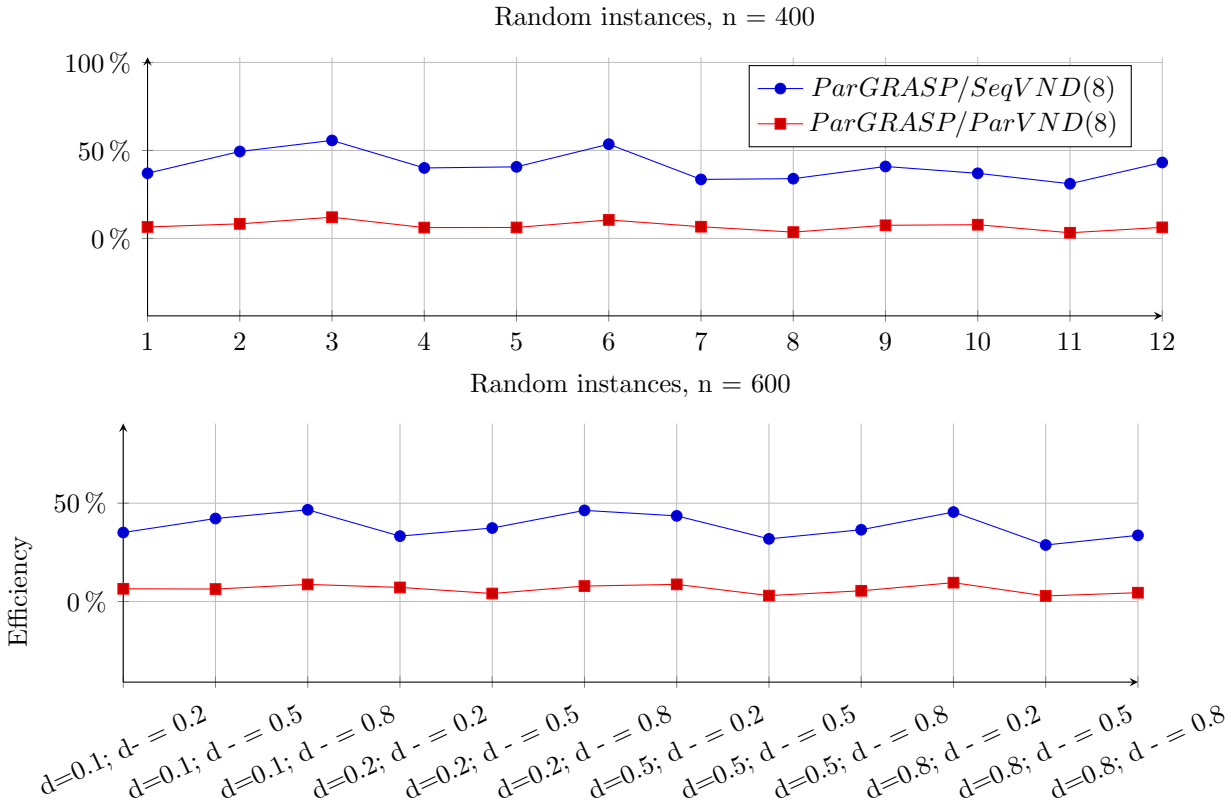

Figure 2: Average efficiency of parallel GRASP with sequential local search ($ParGRASP/SeqVND(8)$) and parallel GRASP with parallel local search ($ParGRASP/ParVND(8)$) when solving random instances in (iii), after 25 independent executions.

results over Slashdot instances in (i), $ParILS/SeqVND$ is up to 4 times faster than $ParGRASP/SeqVND$ when the stopping criterion is a specific target solution (Table 4).

To compare the behavior of $ParGRASP/SeqVND$ and $ParILS/SeqVND$ algorithms, we present a Time-to-Target Plot (TTT-plot) [1] when solving Slashdot $n = 8000$ instance (Figure 3). The improved behavior of $ParILS/SeqVND$ is evident. For example, the probability of $ParILS/SeqVND$ finding the target solution of 16087 in 400 seconds is almost equal to 100% while the same probability lies below 90% for $ParGRASP/SeqVND$.

Finally, for the random instances in (iii), as seen in Table 5, $ParILS/SeqVND$ is always faster than $ParGRASP/SeqVND$ when finding the specified target solution values, being on average 3 times faster.

In summary, it is clear that the parallel multistart ILS metaheuristic is an improvement over the parallel GRASP approach to solve the CC Problem, either outperforming, in processing time, the GRASP metaheuristic proposed earlier, or improving the solution quality.

n	Target I(P)	ParGRASP/SeqVND(8)		ParILS/SeqVND(10)			T(GRASP) / T(ILS)
		Avg Time	CI	Avg Time	CI	Gap Time	
2000	2187	10.83	3.74	4.18	2.08	-6.64	2.59
4000	6203	64.03	37.36	20.33	3.55	-43.70	3.15
8000	16087	347.27	222.29	88.33	23.61	-258.94	3.93
10000	20589	985.85	786.81	246.33	199.92	-739.52	4.00
Average	**-**	**351.99**	**-**	**89.79**	**-**	**-262.20**	**3.42**

Table 4: CC results obtained on Slashdot signed graphs by the use of the following metaheuristic approaches: Parallel GRASP with Sequential VND ($ParGRASP/SeqVND(np)$) and Parallel ILS with Sequential VND ($ParILS/SeqVND(np)$), where np is the number of processes in parallel. Target I(P) is the target imbalance value used as stopping criterion for each algorithm. Avg Time is the average execution time (in seconds) spent by each algorithm to reach the specified target solution value, after 25 independent executions. CI is the 95% confidence interval of the execution time, for each algorithm. T(GRASP)/T(ILS)= [AvgTime(ParGRASP/SeqVND)/AvgTime(ParILS/SeqVND)].

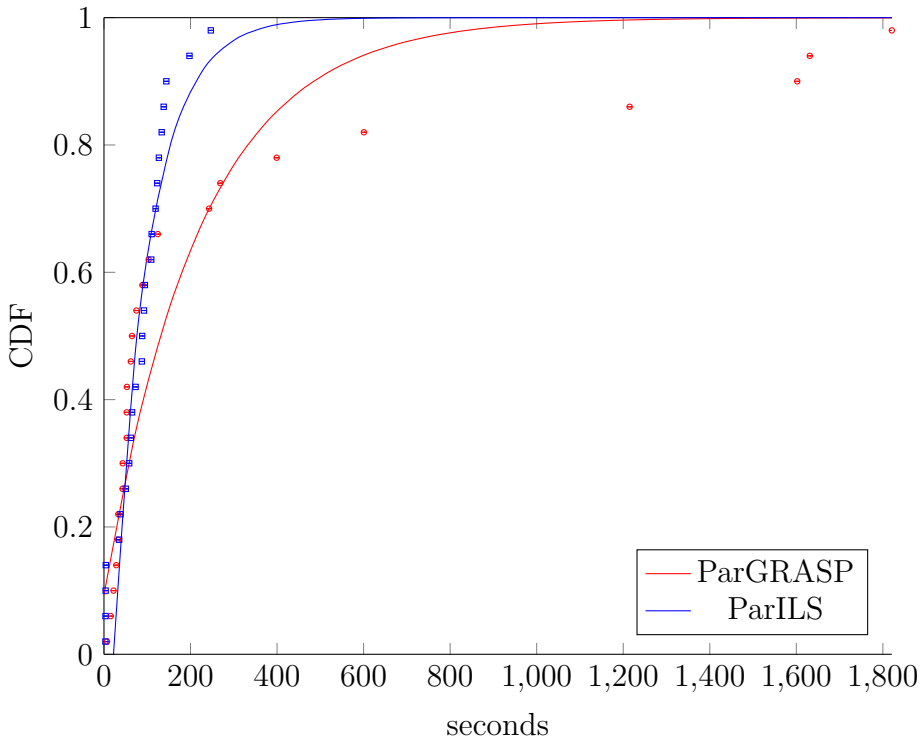

Figure 3: Time-to-Target Plot (TTT-plot)[1] for ParGRASP and ParILS algorithms when solving Slashdot $n = 8000$ instance to obtain the target solution value $I(P) = 16087$. TTT-plots show, on the y-axis, the probability that an algorithm will find a solution at least as good as a given target solution value within a specific running time, displayed on the x-axis.

				Target	ParGRASP/SeqVND(8)		ParILS/SeqVND(10)			
n	**e**	**d**	**d-**	**I(P)**	**AvgTime**	**CI**	**AvgTime**	**CI**	**Gap time**	**Speedup**
100	990	0.1	0.2	198	0.00	0.00	0.00	0.00	0.00	1.16
			0.5	228	0.07	0.02	0.04	0.01	-0.02	1.55
			0.8	56	0.10	0.03	0.05	0.01	-0.06	2.18
	1980	0.2	0.2	396	0.00	0.00	0.00	0.00	0.00	1.21
			0.5	582	0.28	0.10	0.15	0.08	-0.14	1.91
			0.8	208	0.24	0.09	0.09	0.02	-0.15	2.71
	4950	0.5	0.2	990	0.00	0.00	0.00	0.00	0.00	1.08
			0.5	1838	0.22	0.07	0.13	0.04	-0.09	1.72
			0.8	734	0.76	0.19	0.18	0.04	-0.58	4.28
	7920	0.8	0.2	1584	0.00	0.00	0.00	0.00	0.00	1.17
			0.5	3124	0.42	0.16	0.40	0.15	-0.02	1.05
			0.8	1308	0.42	0.15	0.16	0.03	-0.26	2.62
200	3980	0.1	0.2	792	0.01	0.00	0.00	0.00	0.00	1.15
			0.5	1206	0.55	0.19	0.33	0.11	-0.22	1.67
			0.8	402	1.11	0.35	0.31	0.06	-0.80	3.53
	7960	0.2	0.2	1592	0.01	0.00	0.00	0.00	0.00	1.16
			0.5	2852	0.80	0.39	0.24	0.08	-0.56	3.29
			0.8	1098	1.08	0.31	0.35	0.08	-0.74	3.12
	19900	0.5	0.2	3980	0.01	0.00	0.01	0.00	0.00	1.14
			0.5	8140	1.43	0.61	0.65	0.30	-0.77	2.18
			0.8	3354	2.75	1.10	0.66	0.12	-2.09	4.17
	31840	0.8	0.2	6368	0.01	0.00	0.01	0.00	0.00	1.22
			0.5	13586	1.81	0.97	0.62	0.22	-1.18	2.89
			0.8	5696	4.70	1.46	0.93	0.18	-3.77	5.05
300	8970	0.1	0.2	1794	0.01	0.00	0.01	0.00	0.00	1.10
			0.5	3040	1.36	0.56	0.50	0.14	-0.86	2.72
			0.8	1122	1.71	0.67	0.60	0.11	-1.12	2.87
	17940	0.2	0.2	3588	0.01	0.00	0.01	0.00	0.00	1.15
			0.5	6862	1.53	0.58	0.67	0.30	-0.86	2.29
			0.8	2758	3.58	1.20	0.86	0.15	-2.72	4.18
	44850	0.5	0.2	8970	0.02	0.00	0.01	0.00	0.00	1.22
			0.5	19080	7.14	4.31	1.40	0.68	-5.73	5.08
			0.8	7928	6.88	1.96	1.27	0.16	-5.61	5.42
	71760	0.8	0.2	14352	0.02	0.00	0.02	0.00	0.00	1.16
			0.5	31698	7.84	4.15	2.33	1.05	-5.51	3.36
			0.8	13212	10.39	3.58	1.98	0.36	-8.41	5.25
400	15960	0.1	0.2	3192	0.01	0.00	0.01	0.00	0.00	1.10
			0.5	5746	3.19	1.50	0.91	0.33	-2.28	3.49
			0.8	2204	3.83	1.51	1.29	0.25	-2.55	2.98
	31920	0.2	0.2	6384	0.02	0.00	0.01	0.00	0.00	1.16
			0.5	12778	2.95	1.09	0.84	0.26	-2.11	3.52
			0.8	5180	11.15	3.50	1.59	0.37	-9.55	6.99
	79800	0.5	0.2	15960	0.03	0.00	0.03	0.00	0.00	1.13
			0.5	34764	9.59	5.49	3.00	0.94	-6.59	3.20
			0.8	14476	14.59	4.16	2.34	0.44	-12.25	6.23
	127680	0.8	0.2	25536	0.05	0.00	0.04	0.00	-0.01	1.16
			0.5	57342	22.78	11.94	5.58	2.52	-17.20	4.08
			0.8	23922	22.73	6.05	4.16	0.80	-18.58	5.47
600	35940	0.1	0.2	7188	0.03	0.00	0.02	0.00	0.00	1.13
			0.5	13802	6.72	3.94	1.28	0.37	-5.45	5.26
			0.8	5484	13.29	4.12	2.74	0.52	-10.55	4.85
	71880	0.2	0.2	14376	0.04	0.00	0.03	0.00	0.00	1.13
			0.5	30046	17.30	7.93	3.20	1.11	-14.10	5.40
			0.8	12362	15.20	5.64	3.76	0.75	-11.44	4.04
	179700	0.5	0.2	35940	0.08	0.00	0.06	0.01	-0.02	1.26
			0.5	80522	54.04	34.43	6.72	3.16	-47.33	8.05
			0.8	33524	34.81	9.99	7.68	1.74	-27.12	4.53
	287520	0.8	0.2	57504	0.11	0.01	0.09	0.01	-0.02	1.17
			0.5	132006	116.92	88.85	24.61	15.92	-92.30	4.75
			0.8	54872	58.45	15.50	11.43	2.45	-47.02	5.11
Average				**-**	**-**	**-**	**-**	**-**	**-6.15**	**2.94**

Table 5: Parallel GRASP ($ParGRASP/SeqVND(8)$) and parallel ILS ($ParILS/SeqVND(10)$) results for random instances in (iii). Target I(P) is the target imbalance value used as stopping criterion for each algorithm. Avg Time is the average execution time (in seconds) spent by each algorithm to reach the specified target solution value, after 25 independent executions. CI is the 95% confidence interval of the execution time, for each algorithm. Gap time is the gap between parallel ILS and parallel GRASP execution times. Speedup= [AvgTime(ParGRASP)/AvgTime(ParILS)].

References

[1] R. M. Aiex, M. G. Resende, C. C. Ribeiro, Ttt plots: a perl program to create time-to-target plots, Optimization Letters 1 (4) (2007) 355–366.