Neighborhood composition strategies in Stochastic Local Search

Janniele A. S. Araujo[1,2], Haroldo G. Santos[2], Davi D. Baltar[1],
Túlio A. M. Toffolo[2,3], and Tony Wauters[3]

[1] Computer and Systems Department, Federal University of Ouro Preto, Brazil
janniele@decsi.ufop.br, davibaltarx@gmail.com
[2] Department of Computing, Federal University of Ouro Preto, Brazil
haroldo@iceb.ufop.br, tulio@toffolo.com.br
[3] Computer Science Department, CODeS, KU Leuven, Belgium
tony.wauters@cs.kuleuven.be

Abstract. Methods based on Stochastic Local Search (SLS) have been ranked as the best heuristics available for many hard combinatorial optimization problems. The design of SLS methods which use many neighborhoods poses difficult questions regarding the exploration of these neighborhoods: how much computational effort should be invested in each neighborhood? Should this effort remain fixed during the entire search or should it be dynamically updated as the search progresses? Additionally, is it possible to learn the best configurations during *runtime* without sacrificing too much the computational efficiency of the search method? In this paper we explore different tuning strategies to configure a state-of-the-art algorithm employing fourteen neighborhoods for the Multi-Mode Resource Constrained Multi-Project Scheduling Problem. An extensive set of computational experiments provide interesting insights for neighborhood selection and improved upper bounds for many hard instances from the literature.

Keywords: stochastic local search, project scheduling, multi-neighborhood search, learning, online and offline tuning

1 Introduction

Stochastic Local Search (SLS) methods obtained best results for many optimization problems. In school timetabling, Fonseca et al. [3] won the Third International Timetabling competition with a hybrid Simulated Annealing incorporating eight neighborhoods. In Project Scheduling, Asta et al. [1] won the MISTA Challenge [9] with a Monte Carlo based search method with nine neighborhoods. In both methods, neighborhoods are explored stochastically instead of the much popular deterministic best/first fit alternatives where local optimality is usually reached at every iteration. Instead, in these SLS algorithms, a random neighbor is generated in one of the neighborhoods and its acceptance is immediately decided. In this paper, we focus on SLS algorithms with these characteristics, instead of considering the broader definition of SLS [6].

Reasons for the good performance of these methods are subject of study. Some insights may come from theoretical models such as [7] or from experimental studies such as [5]. As both studies point out, more connected search spaces provide a landscape where it is easier to escape from attractive basins in the search landscape. This is a very favorable point of SLS, since the different neighborhoods and the flexibility used in their exploration contribute to an increased connectivity of the search space graph.

Given a set of neighborhoods $\{\mathcal{N}_1, \ldots, \mathcal{N}_k\}$ and a search method, we define probabilities $\mathbf{p} = (p_1, \ldots, p_k)$ of selecting each neighborhood at each iteration as *neighborhood composition*. As k increases, it gets harder to find the best configuration of $\mathbf{p}$. Some important considerations are: should $\mathbf{p}$ remain fixed during the entire search or should it be dynamically updated as the search progresses? Is it possible to learn the best configurations during runtime without sacrificing too much the computational efficiency of the search method? In this paper we try to answer these questions considering a state-of-the-art heuristic for the Multi-Mode Resource Constrained Multi-Project Scheduling Problem where fourteen neighborhoods are stochastically explored.

2 The Multi-Mode Resource Constrained Multi-Project Scheduling Problem

The Multi-Mode Resource Constrained Multi-Project Scheduling Problem (MMRCMPSP) is a generalization of the Project Scheduling Problem (PSP). In this problem jobs can be processed in different modes, with varying execution speeds and consuming different amounts of resources. Both renewable and non-renewable resources are present. Note that the non-renewable resources render the generation of an initial feasible solution NP-Hard, since selecting a valid set of modes corresponds to solving a multi-dimensional knapsack problem. The objective function considers the project delays.

An instance of the MMRCMPSP[9] can be defined by:

P projects with release dates h_p, $p \in P$;
J jobs, connected by a set of precedence relations B;
K nonrenewable resources with capacities o_k, $k \in K$;
R renewable resources with capacities q_r, $r \in R$;

Jobs must be scheduled respecting release dates and precedence constraints. Each job has to be assigned to a processing mode, which defines its resource consumption and duration. A job j executed in mode m has duration d_{jm} and consumes u_{kjm} units of non-renewable resource k, $k \in K$, and v_{rjm} units of renewable resource r, $r \in R$. The objective functions minimize, hierarchically: (i) the total project delay (TPD), which considers project completion times and (ii) the total makespan (TMS), which considers the completion of the last project.

3 Neighborhoods

A solution is represented by an ordered pair $(\pi, \mathcal{M})$, where π indicates an allocation sequence and $\mathcal{M}$ is a feasible set of modes, i.e. an allocation of modes which respects the availability of nonrenewable resources. A complete solution is decoded with a Serial SGS algorithm [4] from $(\pi, \mathcal{M})$, allocating each job in the first timeslot where precedence constraints are respected and sufficient renewable resources are available. All neighborhoods operate in the search space of feasible values for $(\pi, \mathcal{M})$, i.e. movements violating precedence constraints or nonrenewable resource usage constraints are discarded. Neighborhoods operate either on π or $\mathcal{M}$. Most neighborhoods were proposed by [1]. Some of these neighborhoods were conceived aiming at the minimization of the TPD. They *compact* jobs in π, reorganizing them so that all jobs of a given project are contiguous in the allocation sequence.

Fourteen neighborhoods are considered:

`SPE` **squeeze project on extreme**: all jobs of a project p are compacted around a reference job, while jobs of other projects are moved either before or after project p; Figure 1 shows a sample neighbor s' of s in the `SPE` neighborhood, with position 4 of π being used as a reference job, thus all jobs pertaining to the same project of job 14 are squeezed immediately at its side;

s'=SPE(s,4)

s	2	3	5	8	14	10	7	9	12
s'	3	8	2	5	14	10	9	7	12

Fig. 1. Example of a neighbor s' generated in the `SPE` neighborhood of s

`SCP` **swap and compact two projects**: relative positions of projects are swapped and all jobs of both projects are sequentially allocated, starting with jobs from the project which was previously starting at a latter timeslot;

`OP` **offset project**: all jobs of a given project are shifted by the same number of positions;

`CPP` **compact project on percentage**: a percentage of jobs of a project p are justified to the left, starting from the end of π; Figure 2 shows a sample neighbor s' of s in the `CPP` neighborhood considering project 4 (jobs shown in shaded cells) and compaction percentage 0.5; note that the project has 5 jobs, so $\lfloor 5 \times 0.5 \rfloor = 2$ and thus the last two jobs are contiguously allocated in π;

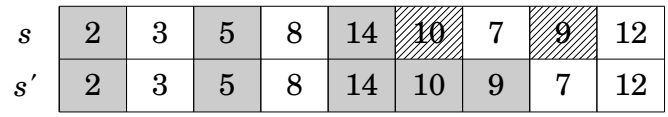

Fig. 2. Example of a neighbor s' generated in the CPP neighborhood of s

ISJ **invert sequence of jobs**: a subsequence of jobs in π is inverted;
OJ **offset job**: shifts the position of one job in the sequence π;
STJ **swap two jobs**: swaps the position of two jobs in the sequence;
CSP **compact subsequent projects**: compress a contiguous list of projects in a sequence;
SSJW **successive swap of a job in a window**: all possible swap positions for a job in a window are explored in a first-fit fashion;
SIJW **successive insertions of a job in a window**: similar to SSJW, but instead of swapping jobs, a job is re-inserted in another position within the explored window;
C1M **change one mode**: changes the mode of one job;
C2M **change two modes**: changes the mode of two jobs;
C3M **change three modes**: changes the mode of three jobs; to reduce the size of this neighborhood only triples of consecutive jobs in the precedence graph are considered;
C4M **change four modes**: changes the mode of four jobs that are consecutive in the precedence graph (similarly to C3M).

4 Offline neighborhood composition

In this section we evaluate a metric to define **p** *a priori*, i.e. based on a statistical analysis of the neighborhoods efficiency. The performance of all neighborhoods in a previously generated pool of solutions is considered. We call a neighborhood *efficient* when its stochastic exploration produces good results, i.e. if it improves the solution cost or generates sideways moves[4] with minimal computational effort.

In a preliminary set of experiments we observed that the efficiency of different neighborhoods varied considerably depending on the *stage* of the search. Neighborhoods which were quite efficient in the beginning of the search did not present the same good properties as the search advanced.

We evaluate the efficiency of neighborhoods in two different stages. In the first stage, low quality (initial) solutions are considered, generated by quick constructive algorithms. In the second stage, incumbent solutions obtained after 10,000 iterations without improvement of a state-of-the-art LAHC solver [8] are

[4] Moves which do not change the objective function value but modify the solution.

selected. Table 1 describes the minimum, maximum and average solution quality[5] for solutions of both phases. Each value consider all instances and 10 solutions per instance (300 solutions in total), taking into account the best known solutions in literature to compute the quality.

Table 1. Characteristics of the solution pool used for *offline* neighborhood analysis

	Solution quality		
	Minimum	Average	Maximum
0	<0.0057	0.3566	0.6666
10000	0.6175	0.8193	1.000

The efficiency of each neighborhood k was computed considering a random sampling of neighbors (10,000) for each solution in the pool. It was observed that only analyzing the improvement of neighbors was not enough, so we consider the number of neighbors corresponding to improved solutions $\tilde{s}(k)$, the number of neighbors corresponding to sideways moves $\underline{s}(k)$ with factor $\vartheta \in [0, 1]$, and the total CPU time $\tilde{c}(k)$ spent validating and evaluating neighbors. Thus, the efficiency $\tilde{e}_k$ of a neighborhood k is given by:

$$\tilde{e}_k = \begin{cases} \dfrac{\tilde{s}(k) + \vartheta\ \underline{s}(k)}{\tilde{c}(k)} & \text{if } \tilde{c}(k) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

To ease comparison, we normalize the efficiency values. Let $\max(\tilde{e})$ be the maximum $\tilde{e}_k$ for all $k \in N$. Equation (2) shows how the normalized efficiency e_k is obtained for a neighborhood $k \in N$:

$$e_k = \frac{\tilde{e}_k}{\max(\tilde{e})} \tag{2}$$

Table 2 presents an efficiency comparison between all neighborhoods in the two different stages. To illustrate the impact of the sideways moves on the final solution quality, the first columns are computed with $\vartheta = 0$ and the last ones are computed with $\vartheta = 0.1$. Note that neighborhoods which operate on entire projects, like `OP`, `CPP`, `SCTP`, are usually well ranked in the first stage. Their efficiency dramatically decreases in the second stage, except for the `CSP`, that has a very low efficiency in the first stage, increasing in the second one due mainly to sideways moves. Neighborhoods which change one mode, swap, inverts or shifts jobs are the most significant for the second stage, corresponding to a stage of small adjustments in π and $\mathcal{M}$.

[5] the quality $q_i = \frac{b_i}{c_i}$ of a solution for instance i with cost c_i, considering the best known solution's cost b_i

Table 2. Normalized neighborhoods efficiency computed *offline*, considering the two stages solution pool

$\vartheta = 0.0$				$\vartheta = 0.1$			
First stage		Second stage		First stage		Second stage	
C1M	1.0000	C1M	1.0000	C1M	1.0000	OJ	1.0000
OP	0.7407	C2M	0.2990	OJ	0.6969	CSP	0.7323
C2M	0.7288	OJ	0.2609	OP	0.6661	STJ	0.6651
OJ	0.5143	STJ	0.2076	C2M	0.6427	C1M	0.5709
CPP	0.4729	ISJ	0.2037	STJ	0.4650	ISJ	0.4345
C3M	0.4716	OP	0.1812	C3M	0.4091	OP	0.3474
STJ	0.3677	C3M	0.0950	CPP	0.4053	CPP	0.1959
SCTP	0.3281	SPE	0.0540	ISJ	0.2824	SCTP	0.1921
C4M	0.3058	SCTP	0.0502	SCTP	0.2753	SPE	0.1789
SPE	0.2725	C4M	0.0167	C4M	0.2578	C2M	0.0865
ISJ	0.2304	CPP	0.0023	CSP	0.2540	C3M	0.0371
SSJW	0.0024	SSJW	0.0019	SPE	0.2327	C4M	0.0029
SIJW	0.0005	SSIW	0.0003	SSJW	0.0016	SSJW	0.0001
CSP	0.0000	CSP	0.0000	SIJW	0.0000	SIJW	0.0000

5 Online neighborhood composition

In this section we consider the *online* neighborhood composition. In this approach, all neighborhoods start with the same probability of being chosen and this probability is dynamically updated considering results obtained during the search. While this approach introduces a learning overhead, it can more easily adapt itself if the neighborhoods' efficiencies change considerably during the search or vary in different instances.

Whenever a sample of z neighbors is explored, the normalized efficiency e_k (Equations (1) and (2)) of each neighborhood $k \in N$ is computed and their selection probabilities are updated according to the results obtained exploring this last batch of neighbors. The probability p_k of selecting a neighborhood $k \in N$ is given by Equation (3). Note that a constant β is included when calculating the probability. This constant prevents assigning probability zero to a neighborhood, and thus all neighborhoods have a minimal chance of being selected in any stage of the search. All experiments in this paper employs $\beta = 0.01$.

$$p_k = \frac{e_k + \beta}{\sum_{k' \in N} (e_{k'} + \beta)} \tag{3}$$

The definition of z is a crucial point: while larger values of z provide a more accurate evaluation of each neighborhood, smaller values offer a more reactive method.

Figure 3 shows how the probabilities evolve over time considering a small instance, A-10. The first graph assumes $\vartheta = 0.0$ (sideways moves are not rewarded),

while the second considers $\vartheta = 0.1$ (sideways moves are partially rewarded). Note that, as one would expect, most improving moves for small instances are executed in the beginning of the search. Therefore, once a local optima is reached, the probabilities remain unchanged when sideways moves are not considered. When sideways moves are considered, however, the probabilities vary during the entire search. It is noteworthy that more complex and expensive neighborhoods such as `CSP`, `SSJW` and `SIJW` were given lower probabilities when $\vartheta = 0.1$.

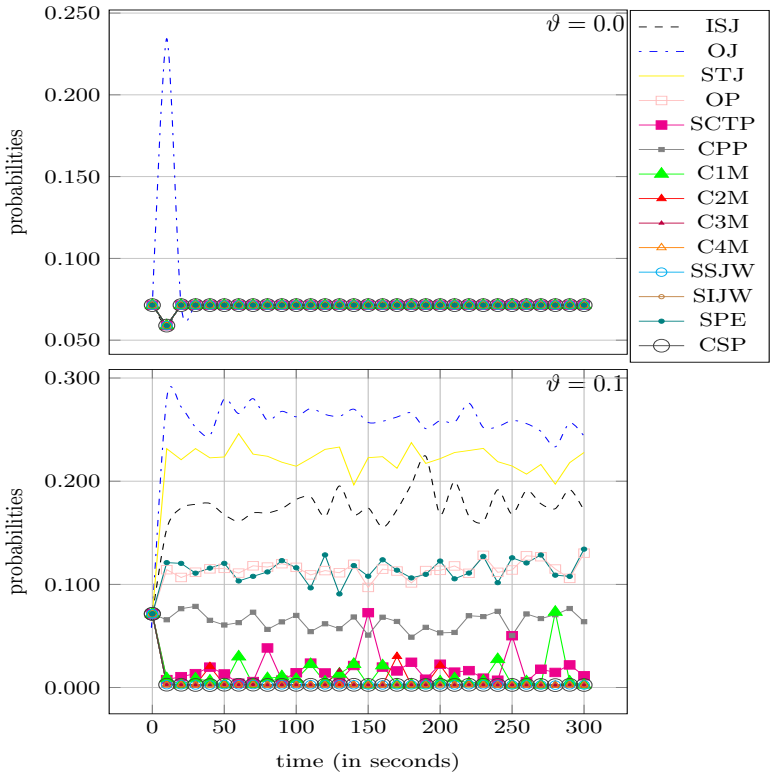

Fig. 3. Evolution of the neighborhood selection probabilities over time considering instances A-10 and *online* tuning ($z = 1,000$ and $\beta = 0.01$)

Figure 4 presents similar graphs to those of Figure 3, but considering a larger instance, B-9. For this instance, neighborhoods `C1M` and `C2M`, which change job execution modes, were highly rewarded when $\vartheta = 0.0$. This stresses the importance of the modes selection for this particular instance. Additionally, the simple neighborhoods `OJ` and `OP`, which shifts the position of one job and one project, respectively, were also assigned high probabilities. Note that if $\vartheta = 0.0$ and the current solution is not improved by any neighborhood within z iterations,

all neighborhoods are assigned the same probability. Figure 4 shows that this situation is recurring.

When sideways moves are rewarded ($\vartheta = 0.1$), the neighborhoods `ISJ`, `OJ` and `STJ`, which change the sequence of jobs, were given high selection probabilities during the entire search.

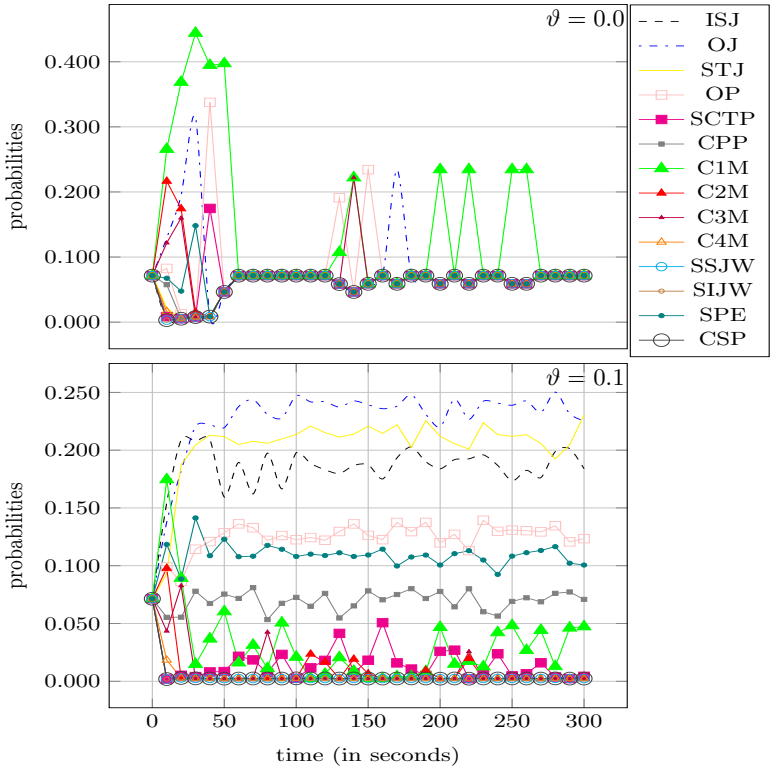

Fig. 4. Evolution of the neighborhood selection probabilities over time considering instances B-9 and *online* tuning ($z = 1,000$ and $\beta = 0.01$)

Figure 5 presents how probabilities evolve for a medium size instance, X-10. We can see that neighborhoods `C1M`, `OJ`, `OP` remain the most significant ones. Other neighborhoods appear with high probabilities at the beginning, like `C2M`. When $\vartheta = 0.1$ the neighborhoods with higher potential to generate sideways moves are assigned higher probabilities.

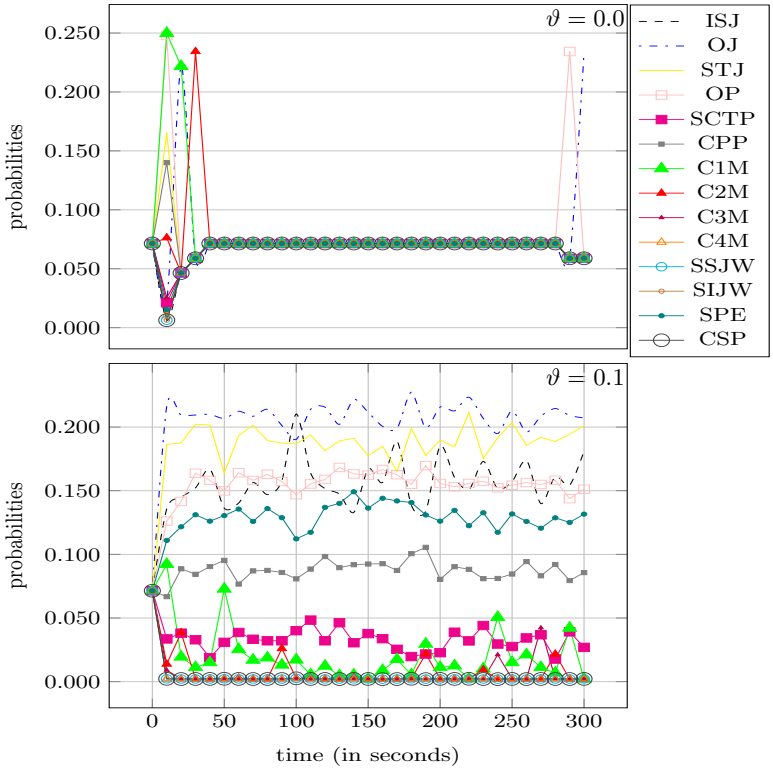

Fig. 5. Evolution of the neighborhood selection probabilities over time considering instances X-10 and *online* tuning ($z = 1,000$ and $\beta = 0.01$)

6 Comparing composition approaches

We evaluated the performance of *offline* and *online* tuning strategies in a metaheuristic framework consisting of the Late Acceptance Hill Climbing (LAHC) algorithm. At each iteration a random neighborhood is chosen considering the probabilities $\mathbf{p}$ in a roulette scheme [2]. The LAHC metaheuristic suits well our purposes since it has only one parameter, the LAHC list size. Most of our tuning effort could then focus in the neighborhood composition. *Offline* and *online* tuning strategies were considered over a single value for this parameter[6], so that both strategies receives the same tuning effort. The overall quality was evaluated over the complete set of instances of the MISTA 2013 challenge [9].

Each neighborhood composition strategy was evaluated on the whole instance set using five independent executions on each instance. The quality q_i of a solution obtained for an instance i is calculated as $q_i = \frac{b_i}{a_i}$, where b_i is the best known solution for i and a_i is the average solution costs. Therefore, the quality

[6] LAHC list size value was fixed to 1,000

$\omega \in [0, 1]$ of solutions for an instance set I is computed as the average value of q_i for all $i \in I$. The standard deviation σ is also included.

Table 3 presents the results obtained with *offline* tuning. Best average results are shown in bold. The first column presents results with uniform probabilities $(1/k)$ for selecting all neighborhood during the entire search. The second column shows the results when probabilities are defined using a single stage, and the third column presents results obtained with the two stages tuning approach. As it can be seen, the efficiency metric (Equation (1)) used to tune the probabilities of selecting each neighborhood produced good results: better average results were produced, in addition to an smaller standard deviation. The best results were obtained in the two stages approach. This indicates that it would be probably beneficial to update the computational effort invested in each neighborhood in different phases of the search process. A natural extension of our proposal would be a more granular approach, i.e. the definition of more than two search phases to more properly adjust the probabilities of selecting each neighborhood as the search progresses.

Table 3. Quality of results for the *offline* tuning strategy

	Offline tuning		
	Uniform	Single stage	Two stages
$\omega_{\vartheta=0}$	0.935	0.949	**0.955**
$\sigma_{\vartheta=0}$	0.041	0.039	0.039
$\omega_{\vartheta=0.1}$	0.935	0.952	0.953
$\sigma_{\vartheta=0.1}$	0.041	0.038	0.037

After evaluating the *offline* approach, we evaluated the *online* approach. Table 4 shows the average quality and standard deviation obtained with the *online* approach using the metric $\tilde{e}$ and different values of z (interval in which probabilities are updated). The best results were obtained with $z = 1000$ and $\vartheta = 0$.

Table 4. Quality of results for *online* tuning $\beta = 0.01$

	Online tuning			
	z=1000	z=10000	z=50000	z=100000
$\omega_{\vartheta=0}$	**0.9468**	0.9428	0.9431	0.9462
$\sigma_{\vartheta=0}$	0.0438	0.0415	0.0444	0.0437
$\omega_{\vartheta=0.1}$	0.9339	0.9358	0.9309	0.9330
$\sigma_{\vartheta=0.1}$	0.0420	0.0422	0.0411	0.0428

7 Composition strategies results

This section presents the results (Table 5) obtained by the LAHC metaheuristic considering both neighborhood composition strategies, and compared them with the best results from literature. An Intel ® Core i7-4790 processor with 3.6 GHz and 16 GB of RAM running SUSE Leap Linux was used during the experiments. All algorithms were coded in `ANSI C 99` and compiled with GCC 4.8.3 using flags *-Ofast* and *-flto*. The runtime limit was set to 300 seconds (the same used in the MISTA Challenge [9]).

Table 5. Comparative results of the LAHC metaheuristic

	LAHC				MISTA [9]		Asta et al. [1]			
	Best		Average		Best		Best		Average	
	TPD	TMS	TPD	TMS	TPD	TMS	TPD	TMS	TPD	TMS
A-1	**1**	**23**	**1**	**23**	**1**	**23**	**1**	**23**	**1**	**23**
A-2	**2**	**41**	**2**	**41**	**2**	**41**	**2**	**41**	**2**	**41**
A-3	**0**	**50**	**0**	**50**	**0**	**50**	**0**	**50**	**0**	**50**
A-4	**65**	**42**	66.2	43.0	**65**	**42**	**65**	**42**	**65**	**42**
A-5	153	104	163.0	108.6	153	105	**150**	**103**	**155**	**105**
A-6	**133**	**91**	147.4	96.0	147	96	133	99	**141**	**808**
A-7	597	203	625.0	202.6	596	196	**590**	**190**	**605**	**201**
A-8	**270**	**153**	**285.4**	**152.8**	302	155	272	148	292	153
A-9	**194**	**126**	**204.2**	**126.0**	223	119	197	122	208	128
A-10	845	308	**877.2**	**310.2**	969	314	**836**	**303**	880	313
B-1	352	128	365.4	130.6	349	127	**294**	**118**	**352**	**128**
B-2	431	162	**448.2**	**167.4**	434	160	**431**	**158**	452	167
B-3	530	208	**550.6**	**208.6**	545	210	**526**	**200**	554	210
B-4	1267	281	1313.6	284.4	1274	289	**1252**	**275**	**1299**	**283**
B-5	816	253	835.4	254.4	820	254	**807**	**245**	**832**	**255**
B-6	**877**	**222**	**917.6**	**226.0**	911	226	905	225	950	232
B-7	793	229	854.2	238.0	792	228	**782**	**225**	**802**	**232**
B-8	**2865**	**520**	**3065.2**	**530.0**	2974	541	2974	541	3323	545
B-9	**4059**	**743**	**4133.4**	**745.0**	4192	746	4062	738	4247	754
B-10	**3030**	**445**	**3094.0**	**442.4**	3050	448	3050	448	3290	455
X-1	390	143	427.6	145.6	392	142	**386**	**137**	**405**	**143**
X-2	351	163	366.6	164.6	349	163	**345**	**158**	**356**	**164**
X-3	**307**	**183**	**325.6**	**190.6**	324	192	310	187	329	193
X-4	909	206	**958.6**	**207.2**	915	208	**907**	**201**	960	209
X-5	1749	371	1786.4	374.0	1768	374	**1727**	**362**	**1785**	**373**
X-6	**686**	**226**	**708.4**	**230.2**	700	234	690	226	730	238
X-7	845	226	887.4	232.8	861	236	**831**	**220**	**866**	**233**
X-8	**1183**	**281**	**1235.2**	**282.4**	1218	286	1201	279	1256	288
X-9	**3084**	**626**	**3189.0**	**638.2**	3268	643	3155	632	3272	648
X-10	1580	381	1617.0	382.4	1600	381	**1573**	**383**	**1613**	**383**

Table 5 shows the results obtained with the LAHC implementation and the best results obtained at the MISTA Challenge (column MISTA), in addition to the improved post competition results obtained by Asta et al. [7] [1]. Average values were computed from five independent executions of the best *online* approach and five of the best *offline* approach.

Many new best known solution were obtained, in addition to improved average solution costs for many instances. considering Asta's results. Note that in [1], the best known results were obtained by running 2,500 independent executions in a computer cluster.

All new best solutions were sent to the official website[8] of MISTA Challenge.

8 Conclusions and Future Works

In this paper we evaluated different strategies to define the best neighborhood composition in stochastic local search methods. These strategies were evaluated in a metaheuristic with fourteen neighborhoods. The first important observation is how important the definition of appropriate neighborhood selection probabilities is: the trivial implementation with uniform neighborhood selection probabilities performed worse than versions with more elaborated strategies to define these values.

Our results provided some valuable insights on the role of each neighborhood. While some neighborhoods are quite useful in the beginning of the search, they may prove themselves ineffective when improved solutions are being processed. Other neighborhoods remain effective during the entire search. One future work would be to understand what makes these neighborhoods special and try to create new neighborhoods inspired in a way that they preserve these good characteristics or discard some neighborhoods that are not significant. Concerning the update of neighborhood selection probabilities during the search, for cases where strong dual bounds can be obtained during the search a *gap based* approach could be used to change neighborhood selection probabilities based on the current maximum relative distance to the optimal solution.

Online tuning approaches can be very useful to simplify the tuning process and to define the best neighborhood composition strategies during *runtime*. While this approach significantly improves the uniform probabilities strategies, it performs slight worse (1% in our tests) than the best configuration obtained with *offline* tuning.

Future work include applying additional intensification and diversification strategies. Furthermore, the presented neighborhood composition may be employed in others metaheuristics.

Acknowledgments: The authors thank CNPq and FAPEMIG for supporting this research.

[7] To the best of our knowledge the post competition results of Asta et al. [1] were not submitted to the MISTA Challenge official website.

[8] https://gent.cs.kuleuven.be/mista2013challenge/

References

1. S. Asta, D. Karapetyan, H. Kheiri, E. Ozcan, and A.J. Parkes. Combining monte-carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. In G. Kendall, G. Vanden Berghe, and B. McCollum, editors, *Proceedings of the 6th Multidisciplinary International Scheduling Conference*, volume (in review, pages 836–839, 2013.
2. J.E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proc. of the 2nd Intl Conf on GA*, pages 14–21. Lawrence Erlbaum Associates, Inc. Mahwah, NJ, USA, 1987.
3. G.H.G da Fonseca, H. G. Santos, T.A.M. Toffolo, S.S. Brito, and M.J.F. Souza. GOAL solver: a hybrid local search based solver for high school timetabling. *Annals of Operations Research*, 2014.
4. E. L. Demeulemeester and W. S. Herroelen. *Project Scheduling: A Research Handbook*. Kluwer Academic Publishers, 2002.
5. P. Hansen and N. Mladenović. First vs. best improvement: An empirical study. *Discrete Applied Mathematics*, 154(5):802–817, 2006.
6. H. H. Hoos and T. Stützle. Stochastic local search: Foundations and applications. pages 149–201. Morgan Kaufmann, 2005.
7. G. Ochoa, S. Verel, F. Daolio, and M. Tomassini. Local optima networks: A new model of combinatorial fitness landscapes. *Recent Advances in the Theory and Application of Fitness Landscapes*, 6:233–262, 2014.
8. J.A. Soares, H. G. Santos, Davi Baltar, and T.A.M. Toffolo. LAHC applied to the multi-mode resource-constrained multi-project scheduling problem. In *Proceedings of 7th Multidisciplinary International Conference on Scheduling*, pages 905–908, 2015.
9. T. Wauters, J. Kinable, P. Smet, W. Vancroonenburg, G. Vanden Berghe, and J. Verstichel. The multi-mode resource-constrained multi-project scheduling problem: The MISTA 2013 challenge. *Journal of Scheduling*, 2014.