A Non-Adaptive Stochastic Local Search Algorithm for the CHeSC 2011 Competition

Franco Mascia and Thomas Stützle

{fmascia,stuetzle}@ulb.ac.be
IRIDIA - Université Libre de Bruxelles

Abstract. In this work, we present our submission for the Cross-domain Heuristic Search Challenge 2011. We implemented a stochastic local search algorithm that consists of several algorithm schemata that have been offline-tuned on four sample problem domains. The schemata are based on all families of low-level heuristics available in the framework used in the competition with the exception of crossover heuristics. Our algorithm goes through an initial phase that filters dominated low-level heuristics, followed by an algorithm schemata selection implemented in a race. The winning schemata is run for the remaining computation time. Our algorithm ranked seventh in the competition results. In this paper, we present the results obtained after a more careful tuning, and a different combination of algorithm schemata included in the final algorithm design. This improved version would rank fourth in the competition.

1 Introduction

Recent years have seen progressive abstractions in the design of stochastic local search (SLS) algorithms. From the first heuristics designed to solve instances of specific hard combinatorial optimisation problems, the community of researchers moved towards the engineering of more generic algorithm schemata that could be applied across different problems. Among these SLS methods, also referred to in the literature as meta-heuristics, there are Tabu Search [7,8], Memetic Algorithms [18], Iterated Local Search [16], Iterated Greedy [19], and Variable Neighbourhood Search [17,9].

Hyper-heuristics were introduced in 2001 by Cowling et al. [4], and, as meta-heuristics, they aim at selecting, combining, or adapting low-level heuristics to solve optimisation problems. Hyper-heuristics are usually classified in two families: the first one is composed of algorithms that select the best low-level heuristics for the problem being optimised; the second family is composed of the algorithms that generate or adapt low-level heuristics for the problem at hand. For a recent survey on the subject, see [3].

The first Cross-domain Heuristic Search Challenge (CHeSC 2011) is a competition devised by Burke et al. [1], which aims at encouraging the design of generic heuristic algorithms that can be successfully applied across different problem domains. In this competition, the distinction between the problem domain and the

hyper-heuristic is very clear-cut; in fact, the contestants were asked to implement a hyper-heuristic using HyFlex [1], a Java framework that has some design decisions that emphasise the separation between problem domains and hyper-heuristics.

The framework makes available a series of low-level heuristics for different problem domains. These low-level heuristics are categorised in four different families: (i) mutation heuristics, which perturb the current solution without any guarantee of improving the solution quality; (ii) local search heuristics, which are hill-climbers that apply mutation heuristics accepting only non-deteriorating solutions; (iii) ruin-and-recreate heuristics, which remove solutions components from the current solution and reconstruct it with problem specific constructive heuristics; (iv) crossover heuristics, which combine different solutions to obtain a new solution. The primitives exposed by the framework allow the developers to know how many heuristics of each specific family are available, to set the intensity of mutation of mutation heuristics and the depth of search of local search heuristics, to apply a low-level heuristic to the current solution, to know the quality of the current solution, to store different solutions in memory, and to know how many CPU-seconds are available before the end of the allocated time.

Before the competition, four sample problem domains and ten instances for every domain had been made available to the contestants to test their implementations. The problem domains were: boolean satisfiability (MAX-SAT), one dimensional bin packing, permutation flow shop scheduling, and personnel scheduling. During the competition, three instances of the ten sample instances of each problem domain were selected, and other two instances for each problem domain were added. Moreover, two hidden problem domains were revealed with five instances each. The two hidden problem domains were the traveling salesman problem and the vehicle routing problem.

The competition rules imposed a specific setting in which the submitted SLS algorithms had no possibility of recognising the problem or the instance being optimised, and no information about a solution could be extracted, except for the solution quality. Some limitations imposed by the rules are relatively unrealistic, and hardly encountered in real-world scenarios. For example, one way to enforce the separation between the hyper-heuristic and the problem domain is to randomly shuffle the low-level heuristics identifier. We decided to participate to the competition with an off-line tuned algorithm even though competition rules do not favour such techniques. Our algorithm is based on the rationale that with an appropriately chosen fixed algorithm schema, a problem can be solved competitively even when not adapting the heuristic selection mechanisms at runtime. Obviously, because heuristic identifiers are randomly shuffled in the framework, we cannot offline tune the sequence of the heuristics to be applied. As a work-around to this missing information, we applied a heuristic selection phase, in which we identify dominated heuristics that are eliminated from further consideration when running our algorithm schemata. The algorithm remains in the sense non-adaptive, in that there is no adaptation of the parameters

and no selection of the low level heuristics throughout the search. Moreover, experiments we performed after the submission showed a rather limited impact of this initial phase of filtering low level heuristics. To emphasise the aspect that the heuristics selection is not adapted during the main execution phase of the algorithm, we call it a Non-Adaptive SLS Algorithm (NA-SLS). In the end, our NA-SLS ranked seventh in the competition. This paper describes an improved version of our submission, which shares the same code but uses a more thorough off-line tuning, and adds some more analysis on the algorithm. The version presented in this paper would rank fourth among the algorithms that participated to the competition.

The rest of the paper is organised as follows. Section 2 presents the analysis on the low-level heuristics; Section 3 describes the algorithmic schemata we implemented as building blocks for NA-SLS; Section 4 presents the results of the tuning of these schemata on the four sample problem domains; Section 5 describes the design of NA-SLS; and Section 6 presents the results NA-SLS would have obtained on competition. Eventually, Section 7 draws the conclusion.

2 Analysis of the Low-Level Heuristics

Before actually running NA-SLS, the available low-level heuristics are tested to identify whether any of the heuristics are dominated. The criterion of dominance takes into account the computation time and the solution quality. This first phase of the analysis of the low-level heuristics is run for at most 7.5% of the total allocated runtime or for at most 25 runs of each heuristic. During each run a random solution is constructed and heuristics belonging to the local search, and to ruin-and-recreate families are applied to it. For each low-level heuristic A, the median run-time t_A and the median solution quality q_A are computed. A heuristic A is dominated by heuristic B if $t_B < t_A$ and $q_B < q_A$. The aim of this phase, is to enforce that only non-dominated heuristics will be available for the remaining runtime. Nevertheless, in order to avoid that a single heuristic that dominates all others is the only one available for the following phases, we make sure that the two heuristics having the lowest median solution quality are never discarded. Figure 1 shows an example of the analysis performed on an instance of the personnel scheduling problem. The plot on the left shows all local search heuristics with different values of the parameter that set the intensity of mutation. The plot on the right shows the non-dominated heuristics after the filtering.

3 Design and Implementation of Algorithmic Schemata

We implemented a series of algorithmic schemata that use all low-level heuristics (except crossovers) as basic building blocks. Among the large number of implemented schemata there are several algorithms that are well established in the literature. In the following we list the most relevant, with their variants and the parameters defined for the tuning:

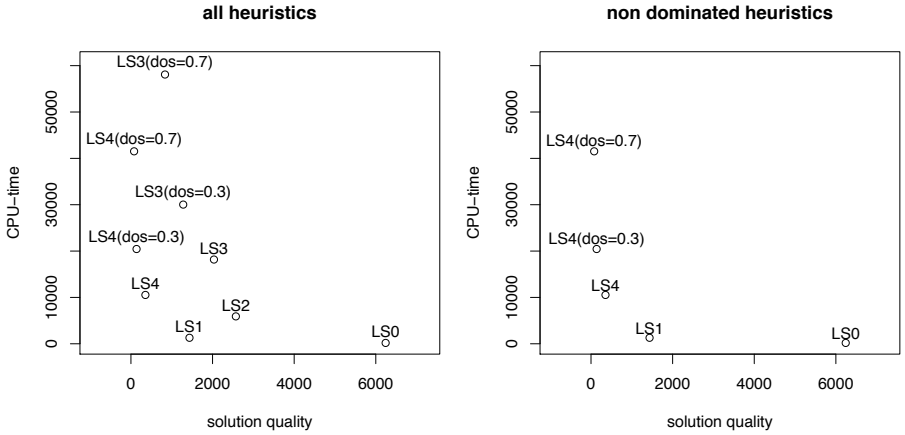

Fig. 1. Example for the analysis of the low-level heuristics on an instance of the personnel scheduling domain. LS4(dos=0.7) corresponds to local search heuristic number 4, with the depth of search parameter set to 0.7. Where not specified, the depth of search is set to the default 0.1. LS4(dos=0.7), LS3(dos=0.3), LS3, and LS2 are the dominated heuristics.

- Randomized Iterative Improvement (RII) [10] with probabilistic mutations. The parameters defined for the tuning are two continuos parameters representing the probability of selecting a mutation or a ruin-and-recreate heuristic, respectively, and a continuous parameter representing the probability of accepting worsening solutions.
- Probabilistic Iterative Improvement (PII) [10]; this algorithm is the same as the previous, but it uses a Metropolis condition for computing the probability of accepting worsening solutions and has continuous parameter for the temperature.
- Variable Neighbourhood Descent (VND) [9]; for this algorithm we implemented two versions. The first one uses local search heuristics, and the second one uses ruin-and-recreate heuristics. In both cases, the heuristics are applied in the order of decreasing median computation time, which is computed by running the low-level heuristics several times from an initial solution. The parameters defined for this schema are a categorical parameter that allows to select among the two variants, and a continuos parameter representing the probability of accepting worsening solutions as in RII.
- Iterated Local Search (ILS) [16]; few variations of this algorithm have been implemented. We implemented the ILS described in [2], an ILS that uses the VND variants described above as subsidiary local search, and a Hierarchical ILS [11]. Hierarchical ILS uses an ILS as subsidiary local search and applies a strong perturbation in the outer ILS and a small perturbation in the inner ILS. We defined a categorical parameter that allows to select among the

variants, a continuous parameter for the probability of accepting a worsening solution, and an integer parameter for determining the perturbation size, which corresponds to the number of times a random mutation heuristic is applied to the current solution. For the Hierarchical ILS we also used the same parameters for the inner ILS, and a further continuos parameter for the time allocated to the inner ILS as a fraction of the total run time.
- Simulated Annealing (SA) [12,20]; this algorithm estimates the initial temperature by measuring the average solution quality improvement after applying a fixed number of steps of first improvement. The initial acceptance probability and the number of steps for this initial phase are a continuos and an integer parameter respectively. The algorithm continues with a cooling schedule that is defined by a constant multiplicative factor (continuos parameter) applied regularly after a specified number of steps (integer parameter) until a final temperature is reached (continuous parameter). When the final temperature is reached, the algorithm restarts from the initial estimated temperature.
- Iterated Greedy (IG) [19]; we implemented IG by means of repeated applications of a random ruin-and-recreate heuristic with a probability of accepting worsening solutions, which is specified by a continuous parameter. We also implemented few variations of this algorithmic schema, which differ for a probabilistic acceptance criterion, and different selection strategies of low-level heuristics.

For all schemata we defined two continuos parameters that set the intensity of mutation for the mutation heuristics and the depth of search for local search heuristics. The parameters range from 0 to 1, with 1 representing the maximum intensity of mutation and the maximum depth of search respectively. If the application of a specific local search heuristic takes more than 10 CPU-seconds, the depth of search parameter for that specific heuristic is fixed to the default value defined in the framework, which is 0.1. For RII, PII, and the ILS variants, we implemented also a fixed restart policy. We defined a categorical parameter that could select the restart, and a continuous parameter representing the fraction of the allocated runtime between restarts.

Besides the algorithms that are well-established in the literature, we implemented a further algorithm we called tuneable SLS algorithm (TSA), which is a juxtaposition of blocks of low-level heuristics. TSA has been designed with the aim to see how good could perform an algorithm with less rationale in the design and a larger parameter space for automatic tuning. Listing 1.1 shows the pseudo-code of TSA with the parameters defined for the tuning. The algorithm starts from an initial random solution s, and until the allocated time has not expired, it goes through a series of phases characterised by the application of heuristics belonging to a specific family. In the first phase, the algorithm applies sequentially n_{rr} randomly selected ruin-and-recreate heuristics which, if defined in the framework, exploit constructive heuristics for the problem at hand. Non improving solutions are accepted with a probability $p_{a_{rr}}$. In the following phase, the solution that is constructed by the ruin-and-recreate heuristics is (possibly)

Listing 1.1. Tuneable SLS algorithm pseudo-code.

```
1  procedure TSA(n_rr, p_a_rr, n_ls, p_a_ls, p_a, p_m, p_restart, dos, iom, iom_rr)
2      s' ← s ← random initial solution
3      set depth of search of local search heuristics to dos
4      set intensity of mutation of mutation heuristics to iom
5      set intensity of mutation of ruin−and−recreate heuristics to iom_rr
6      while time has not expired:
7          for n_rr times:
8              apply randomly selected heuristic of type ruin−and−recreate
9              accept non−improving solution with probability p_a_rr
10         for n_ls times:
11             apply randomly selected heuristic of type local search
12             accept non−improving solution with probability p_a_ls
13         if f(s) < f(s') or rand(0, 1) < p_a:
14             s' ← s
15         else
16             s ← s'
17         with probability p_m:
18             apply randomly selected heuristic of type mutation
19         with probability p_restart:
20             s ← random initial solution
```

improved by the application of n_{ls} randomly selected local search heuristics. Non-improving solutions in this phase are accepted with probability $p_{a_{ls}}$. After the first two phases, an improving solution is always accepted and stored in s', while worsening solutions are accepted with probability p_a. Before going again through the series of ruin-and-recreate heuristics of the first phase, the current solution is perturbed by the application of a random mutation heuristic with probability p_m. Finally, with probability $p_{restart}$ the algorithm restarts from a new random solution.

4 Off-line Tuning of the Algorithmic Schemata

After the implementation of the algorithmic schemata, we tuned them on the four problem domains and the ten instances that were available before the competition, and selected a small subset that would be part of the NA-SLS algorithm.

For the off-line tuning, we used irace [14], a software package that implements an iterated racing procedure for the off-line tuning of categorical, integer, and continuous parameters. Moreover, irace allows to specify conditional parameters and therefore tune the parameters regarding a specific algorithm schemata, only when the specific algorithm schemata has been selected by a categorical parameter. These characteristics and a number of successful applications [5,6,13,14,15], made it the natural choice for tuning the parameters of our SLS algorithm.

The off-line tuning is divided in two phases. In the first phase, we tuned the parameters of each algorithmic schema on the single problem domains. In the

Problem domain	**Algorithmic Schema**
MAX-SAT	TSA, parameter setting MAX-SAT
Bin packing	TSA, parameter setting bin packing
Personnel scheduling	TSA, parameter setting personnel scheduling
Permutation flow shop	Iterated Greedy with probabilistic acceptance criterion

Table 1. Selected algorithmic schemata for the problem domains.

Problem domain	**Parameters**									
	n_{rr}	$p_{a_{rr}}$	n_{ls}	$p_{a_{ls}}$	p_a	p_m	$p_{restart}$	dos	iom	iom_{rr}
MAX-SAT	1	0.4966	50	0.9265	0.2341	0.0959	0.0008	0.6	0.1063	0.2
Bin packing	4	0.057	4	0.43	0.0001	0.29	0.014	0.46	0.11	0.69
Personnel Scheduling	11	0.54	15	0.72	0.54	0.2	0.67	0.85	0.43	0.21

Table 2. Parameters of the tuned TSA schemata on three problem domains.

second phase, for each problem domain, we selected the best tuned schemata. The off-line tuning of both phases was performed with irace on cluster nodes with 16GB of RAM and 2 AMD Opteron 6128 CPUs, each with eight cores running at 2GHz. During the two tuning phases, the number of different configurations tested amounted to 136,276 for a total of 3 years of CPU-time.

Table 1 shows the best performing algorithms for the single domains. Surprisingly, three differently tuned TSA were selected for three out of four problem domains, namely MAX-SAT, bin packing and personnel scheduling. The parameter settings for the TSAs are shown in Table 2. For the permutation flow shop scheduling, the best schema is an IG with a probabilistic acceptance criterion. The tuned parameters lead to an IG that executes at each iteration one or two random ruin-and-recreate heuristics. If no such heuristics are available for the problem domain, the algorithm executes one or two mutation heuristics. After the perturbation, one local search heuristic is executed. Worsening solutions are accepted with probability $\exp\{-\Delta f/0.87\}$.

All algorithms in Table 1 were included in the algorithm we submitted for the competition. In the implementation described in this paper we decided to extend the pool of available algorithms by looking at the second ranking algorithm schemata for the four problem domains.

- For MAX-SAT, the second ranking algorithm was an ILS with VND as subsidiary local search (ILS+VND); the probability of accepting a worsening solution was 0.99 and the perturbation size amounted to the application of 2 random mutation heuristics. The intensity of mutation was 0.3 and the depth of search was 0.41.
- For personnel scheduling, it was also an ILS+VND; the tuned probability of accepting a worsening solution was 0.23 and the perturbation size amounted to the application of 6 random mutation heuristics. The intensity of mutation was 0.08 and the depth of search was 0.25.

– For FlowShop again an ILS+VND ranked second, with a low probability of accepting worsening solutions, i.e. 0.39, and large perturbations amounting to 11 applications of random mutation heuristics. The intensity of mutation was 0.59 and the depth of search was 0.022.
– For bin packing an ILS with acceptance probability equal to 0.037 ranked second; for this schema the amount of perturbation selected was of 4 random mutation heuristics. The intensity of mutation was 0.033 and the depth of search was 0.31.

We decided to include in the pool of available schemata three of the four first ranking algorithms, i.e., TSA for bin packing, TSA for personnel scheduling and the Iterated Greedy for permutation flow shop scheduling. We did not include TSA for MAX-SAT since the performances were pretty close to the second ranking and less tailored ILS+VND. In three domains out of four, ILS+VND ranked second, therefore we added two different ILS+VND schemata, namely the ILS+VND tuned for MAX-SAT and the ILS+VND tuned for personnel scheduling. The rationale behind this choice is to add some well-known meta-heuristics that even if tuned on specific problem domains could be effective on the two hidden problem domains.

5 Final SLS Algorithm

NA-SLS is composed of the following three phases.

Phase 1: Analysis of the low-level heuristics.
This phase, which lasts at most 7.5% of the total runtime, is devoted to the analysis of the low-level heuristics available for the problem being optimised. As described in Section 2, only non dominated low-level heuristics will be available for the six selected algorithm schemata in the following phases.

Phase 2: Algorithm selection.
In this phase, a fraction of the remaining time is allocated for selecting the best performing schema for the problem at hand among the schemata with fixed parameters described in Section 4. The selection is as in a race, where at each step the worst candidate schema is eliminated. Starting from the best solution found during the first phase, the algorithmic schemata are run in an interleaved manner each for a fraction of CPU-seconds that amounts to 2.5% of the time remaining after the first phase. After each race, the worst performing schema is discarded, and a new race is performed on the remaining algorithms. The phase terminates when only one schema remains. In order to keep this phase as short as possible, if this phase lasts 25% of the total computation time, the phase is automatically terminated and the best of the remaining algorithms is kept as the winner of the race. This is a very simplistic algorithm selection, which could be seen as a workaround to the competition rules. By using a more sophisticated schema as in [21], we probably would make a faster and better selection.

Phase 3: Run.
The best-performing algorithm is executed for the remaining allocated time.

6 Experimental Results

The competition organisers made available an updated version of the HyFlex framework with all instances and all domains used during the competition, i.e., the four sample ones and the two hidden ones, travelling salesman problem and vehicle routing problem. In order to facilitate the comparison with the algorithms submitted to the competition, the organisers published also all detailed results obtained by all algorithms on all instances, and the random seed used to select the five random instances for each problem domain.

During the competition, algorithms have been run 31 times for 600 seconds on each domain and each instance. The median values have been selected and the algorithms have been ranked. The ranking system is mutuated from the Formula-1 point system. The first eight best heuristics receive 10, 8, 6, 5, 4, 3, 2, and 1 point respectively. In case of ties, the points of the concerned positions are summed and equally divided by the algorithms having the same median solution quality.

We run NA-SLS on one node of the cluster for 1176 CPU-seconds which correspond to 600 CPU-seconds on the competition reference machine. The speedup between the machines has been computed with a benchmark tool supplied by the organisers of the competition. We verified that the amount of seconds computed by the benchmark tool was correct by running on the same node of the cluster our original submission and verifying (with the same random seed used in the competition) that the results obtained in 1176 CPU-seconds corresponded to the results obtained in the competition.

Table 3 and Table 4 show the results on the different problem domains. On three out of four of the sample domains we would score high achieving the first, the third, and the fourth position. For personnel scheduling, we already knew before the final competition that our results would not be very competitive; this is confirmed by the thirteenth position achieved by our algorithm with only 2.5 points. On the two hidden domains, we scored relatively good on the travelling salesman problem and for us surprisingly bad on the vehicle routing problem, where we did not score any point. Future work will be devoted to understand the reasons behind the poor performances on the vehicle routing problem. Overall, with a more careful tuning, and a different combination of schemata, our algorithm would have ranked fourth at the competition.

In order to better understand the results, after the competition, we run an analysis of the impact of the heuristics selection phase. We ran an identical copy of NA-SLS where we allowed all low-level heuristics to be used. The results showed that the impact was more limited than expected. In fact, by allowing dominated heuristics, the final algorithm would still score fourth in the final ranking with 108.2 points, which is more than 107.7 points achieved by the version with the heuristic selection phase. For what concerns the single domains,

MAX-SAT

Rank	Algorithm	Score
1	**NA-SLS**	44.2
2	AdapHH	29.28
3	VNS-TW	28.08
4	HAHA	27.28
5	KSATS-HH	19.5
6	ML	12.30
7	AVEG-Nep	12.1
8	PHUNTER	9.3
9	ISEA	4.10
10	MCHH-S	3.75
11	XCJ	3.60
12	GISS	0.75
12	SA-ILS	0.75
14	ACO-HH	0
14	GenHive	0
14	SelfSearch	0
14	Ant-Q	0
14	EPH	0
14	DynILS	0
14	HAEA	0

Bin packing

Rank	Algorithm	Score
1	AdapHH	45
2	ISEA	30
3	**NA-SLS**	23
4	ACO-HH	19
5	GenHive	14
6	XCJ	12
6	DynILS	12
6	ML	12
9	KSATS-HH	11
10	EPH	9
11	PHUNTER	3
11	VNS-TW	3
13	HAEA	2
14	AVEG-Nep	0
14	GISS	0
14	SA-ILS	0
14	HAHA	0
14	SelfSearch	0
14	Ant-Q	0
14	MCHH-S	0

Flow shop

Rank	Algorithm	Score
1	ML	38
2	AdapHH	36
3	VNS-TW	32
4	**NA-SLS**	26
5	EPH	21
6	HAEA	10
7	PHUNTER	9
7	ACO-HH	9
9	GenHive	7
10	ISEA	3.5
10	HAHA	3.5
12	AVEG-Nep	0
12	GISS	0
12	SA-ILS	0
12	SelfSearch	0
12	Ant-Q	0
12	XCJ	0
12	DynILS	0
12	KSATS-HH	0
12	MCHH-S	0

Table 3. Comparison with CHESC 2011 contestants on MAX-SAT, bin packing, and Flow Shop.

Personnel scheduling

Rank	Algorithm	Score
1	VNS-TW	39.5
2	ML	31
3	HAHA	25
4	SA-ILS	18.5
5	ISEA	14.5
6	PHUNTER	12.5
7	GISS	10
7	EPH	10
9	AdapHH	9
9	KSATS-HH	9
11	GenHive	6.5
12	SelfSearch	5
13	**NA-SLS**	2.5
14	HAEA	2
15	AVEG-Nep	0
15	ACO-HH	0
15	Ant-Q	0
15	XCJ	0
15	DynILS	0
15	MCHH-S	0

TSP

Rank	Algorithm	Score
1	AdapHH	41.25
2	EPH	36.25
3	PHUNTER	26.25
4	VNS-TW	17.25
5	DynILS	13
5	ML	13
7	**NA-SLS**	12
8	ISEA	11
8	HAEA	11
10	ACO-HH	8
11	GenHive	3
11	SelfSearch	3
13	AVEG-Nep	0
13	GISS	0
13	SA-ILS	0
13	HAHA	0
13	Ant-Q	0
13	XCJ	0
13	KSATS-HH	0
13	MCHH-S	0

VRP

Rank	Algorithm	Score
1	PHUNTER	33
2	HAEA	28
3	KSATS-HH	23
4	ML	22
5	AdapHH	16
5	HAHA	16
7	EPH	12
8	AVEG-Nep	10
9	GISS	6
9	GenHive	6
9	VNS-TW	6
12	ISEA	5
12	XCJ	5
14	SA-ILS	4
15	ACO-HH	2
16	DynILS	1
17	**NA-SLS**	0
17	SelfSearch	0
17	Ant-Q	0
17	MCHH-S	0

Table 4. Comparison with CHESC 2011 contestants on personnel scheduling, TSP, and VRP.

the ranking and points would not change on MAX-SAT, bin packing, travelling salesman, and vehicle routing problem. On flow shop scheduling the algorithm would actually gain 3 points and still score fourth with 29 points; and for personnel scheduling it would lose the 2.5 points and rank fourteenth.

In order to analyse the impact of the algorithm selection, we implemented a version of NA-SLS in which an oracle would always choose the best off-line tuned candidate schemata for the four problems available during the tuning phase. Also in this case the algorithm would have scored fourth with 118.25 points. In the single problem domains, it would have ranked first in MAX-SAT with 45.25 points, third in bin packing with 27 points, fourth in flow shop scheduling with 24 points, and eighth in personnel scheduling with 10 points. This version of the algorithm has also more time available for the search, since it is not spending time for the algorithm selection phase. There could be still a small gap to improve our algorithm schemata selection, but as another benchmark it would be also interesting to see how it compares with a random selection of the schemata.

7 Conclusions

In this paper we presented in detail a further development of our submission for the CHeSC 2011 challenge. Our algorithm is composed by different schemata we

All problem domains

Rank	Algorithm	Score
1	AdapHH	176.53
2	ML	128.3
3	VNS-TW	125.83
4	**NA-SLS**	107.7
5	PHUNTER	93.05
6	EPH	88.25
7	HAHA	71.78
8	ISEA	68.1
9	KSATS-HH	62.5
10	HAEA	53
11	ACO-HH	38
12	GenHive	36.5
13	DynILS	26
14	SA-ILS	23.25
15	AVEG-Nep	22.1
16	XCJ	20.6
17	GISS	16.75
18	SelfSearch	8
19	MCHH-S	3.75
20	Ant-Q	0

Table 5. Comparison with CHESC 2011 contestants on all problem domains.

tuned on four sample problem domains supplied by the competition organisers. After tuning each algorithmic schema on each problem domain, we selected a pool of five schemata that would be part of the final algorithm and that would be selected at runtime with a simplistic algorithm selection mechanism. The experimental results shows that our algorithm would have ranked fourth at the competition.

There are several ad-hoc choices that were done without much analysis, for example the algorithm selection schema sounded a reasonable strategy to try, but it could be replaced with more sophisticated schemes that would probably allow for a faster and better selection. Heuristics and parameter adaptation schemes considering the results of the other algorithms could be another step to apply. Eventually, it would also be interesting to test a different version of the competition, in which the low level heuristics are not reshuffled and therefore their choice or the sequence of their execution could be directly tuned.

References

1. Burke, E., Curtois, T., Hyde, M., Ochoa, G., Vazquez-Rodriguez, J.A.: HyFlex: A Benchmark Framework for Cross-domain Heuristic Search. ArXiv e-prints 1107.5462 (Jul 2011)

2. Burke, E., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., Vazquez-Rodriguez, J.A., Gendreau, M.: Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms. In: IEEE Congress on Evolutionary Computation. pp. 1–8. IEEE Press, Piscataway, NJ (Jul 2010)
3. Chakhlevitch, K., Cowling, P.: Hyper-heuristics: Recent developments. In: Cotta, C., Sevaux, M., Sörensen, K. (eds.) Adaptive and Multilevel Metaheuristics, Studies in Computational Intelligence, vol. 136, pp. 3–29. Springer (2008)
4. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: Burke, E., Erben, W. (eds.) PATAT III, LNCS, vol. 2079, pp. 176–190. Springer Berlin / Heidelberg (2001)
5. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: Automatic configuration of state-of-the-art multi-objective optimizers using the TP+PLS framework. In: Krasnogor, N., et al. (eds.) GECCO 2011, pp. 2019–2026. ACM press, New York, NY (2011)
6. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. Computers & Operations Research 38(8), 1219–1236 (2011)
7. Glover, F.: Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 13(5), 533–549 (1986)
8. Hansen, P., Jaumard, B.: Algorithms for the maximum satisfiability problem. Computing 44, 279–303 (April 1990)
9. Hansen, P., Mladenovic, N.: Variable neighborhood search: Principles and applications. European Journal Of Operational Research 130(3), 449–467 (2001)
10. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Elsevier, Amsterdam, The Netherlands (2004)
11. Hussin, M.S., Stützle, T.: Hierarchical iterated local search for the quadratic assignment problem. In: Blesa, M.J., Blum, C., Gaspero, L.D., Roli, A., Sampels, M., Schaerf, A. (eds.) HM 2009, LNCS, vol. 5818, pp. 115–129. Springer Verlag, Heidelberg, Germany (2009)
12. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220(4598), 671–680 (1983)
13. Liao, T., Montes de Oca, M.A., Aydin, D., Stützle, T., Dorigo, M.: An incremental ant colony algorithm with local search for continuous optimization. In: Krasnogor, N., et al. (eds.) GECCO 2011, pp. 125–132. ACM press, New York, NY (2011)
14. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011), `http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf`
15. López-Ibáñez, M., Stützle, T.: The automatic design of multi-objective ant colony optimization algorithms. IEEE Transactions on Evolutionary Computation (2012), accepted
16. Lourenço, H.R., Martin, O., Stüttzle, T.: Iterated local search: Framework and applications. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 146, chap. 9, pp. 363–397. Springer, New York, NJ, 2 edn. (2010)
17. Mladenovic, N., Hansen, P.: Variable neighbourhood search. Computers and Operations Research 24(11), 71–86 (1997)
18. Moscato, P.: Memetic algorithms: a short introduction, pp. 219–234. McGraw-Hill Ltd., UK, Maidenhead, UK, England (1999)

19. Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177(3), 2033–2049 (2007)
20. Černý, V.: Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of Optimization Theory and Applications 45, 41–51 (1985)
21. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: Portfolio-based algorithm selection for SAT. Journal of Artificial Intelligence Research 32(1), 565–606 (2008)