When Simpler is Better: Automated Configuration of a University Timetabling Solver

Thomas Feutrier, Nadarajen Veerapen, Marie-Éléonore Kessaci

When Simpler is Better: Automated Configuration of a University Timetabling Solver

Thomas Feutrier
Univ. Lille, CNRS, Centrale Lille
UMR 9189 - CRIStAL
F-59000 Lille, France
thomas.feutrier@univ-lille.fr
0000-0003-0854-3176

Nadarajen Veerapen
Univ. Lille, CNRS, Centrale Lille
UMR 9189 - CRIStAL
F-59000 Lille, France
nadarajen.veerapen@univ-lille.fr
0000-0003-3699-1080

Marie-Éléonore Kessaci
Univ. Lille, CNRS, Centrale Lille
UMR 9189 - CRIStAL
F-59000 Lille, France
mkessaci@univ-lille.fr
0000-0002-4372-5162

***Abstract*—The Curriculum-Based Course Timetabling problem, which involves scheduling classes in a curriculum, is a prevalent issue in university timetabling. A number of papers have been published, offering various solutions to the problem by utilizing its unique characteristics. A current leading method is a hybrid approach combining different local search techniques. Our study aims to deconstruct this method by breaking it down into smaller components that can be adjusted through the use of parameters. In order to achieve this, we employ a configurator, more specifically irace, to identify the most optimal configurations. Our findings reveal that better configurations exist, and notably, these configurations are simpler than the original method. This indicates that the state-of-the-art method can be outperformed by simpler methods that involve fewer algorithms. This paper presents a comprehensive analysis of the new configurations, which sheds light on the least important components of the original method. Additionally, we conduct an ablation analysis to identify the most crucial parameters or sub-parts that contribute to the most effective solvers.**

***Index Terms*—Automatic Configuration, Parameter Tuning, Local Search, Timetabling**

I. Introduction

Timetabling is a common problem for all universities in the world. This field has an active research community due notably to events such as the International Timetabling Competition (ITC) organized by the EURO Working Group on Automated Timetabling and the associated PATAT conference. Some editions of the competition formalize a problem, as in ITC 2007 with the Curriculum-Based Course Timetabling (CB-CTT) problem [14] that we consider in this paper.

These complex problems require significant computation time if exact methods are employed. Thus, the preferred solvers are often metaheuristics that offer reasonably good results in a relatively short time [1]. There are several types among the most effective heuristics, including genetic algorithms, local search [13], and hybrids [2]. New articles also focus on more complex methods in terms of structure: these are hyperheuristics [16] that show good results and adapt to the problems they encounter. Our own work has also looked at predicting solver performance by analyzing the problem [7].

In most cases, state-of-the-art solvers use a fixed configuration, and some users might assume that that configuration is the optimal one. Nevertheless, some papers show that other configurations offer better results. For instance, Song et al. [15] showed that a local search method for CB-CTT [13] can increase its performance by slightly modifying the operator selection process. In particular, they changed the roulette wheel probabilities that select the next operator. This change may seem small but is significant nonetheless.

There exist a number of configurators to find new promising configurations, such as ParamILS [8] or irace [10]. One of their main advantages is to automate the search for new configurations. They have already been used to tune new methods that are efficient on timetabling problems [17]. Some methods are dedicated to comparing two configurations, highlighting the most relevant parameters [5].

In this paper, we focus on the Hybrid Local Search (HLS) method, which was the winner of ITC 2007 for the Curriculum-Based Course Timetabling (CB-CTT) problem [13] and remains the state-of-the-art solver for this problem [3]. The HLS method comprises three main algorithmic components: Hill Climbing, Great Deluge, and Simulated Annealing, and utilizes five different neighborhood operators.

We use irace [10], a configurator, to provide us with new configurations for HLS by training on over 3 000 recently proposed synthetic instances [3]. These configurations vary according to parameters controlling the neighborhood operators and the global structure of the algorithm. In particular, we do not assume, as has been done before, that all the components of HLS are required and allow each of them to be toggled on or off, in addition to changing the values of its numeric parameters.

The new algorithm variants are tested on real-world instances to evaluate their performance. Using statistical tests, we obtain the ranks of each method. Our results clearly show

that the state-of-the-art HLS solver can be outperformed as one might expect when tuned with irace . In particular and more surprisingly, because we introduced the possibility to deactivate each main algorithmic component, simpler versions of the method emerge as viable competitors to HLS , including one that only consists of the Great Deluge algorithm. This is especially interesting in light of recent papers [3], [15] that build upon HLS as the *de facto* state-of-the-art method, and shows that there is room for improvement of the method.

We analyze each configuration to determine the optimal structures that emerge. The paper focuses on these new configurations by comparing them with baselines. We investigate why some configurations do not have the same performances while they differ only in one parameter, that might be considered as minor. Our work attempts to show that it is be relevant to use parameters that expose and allow for changes in the structure of the solver even before tuning more precise numerical parameters. That provides important information on how to build a new solver. Finally, we use ablation analysis [5] to analyze each partial configuration between two configurations to highlight the local importance of the parameters.

The paper is organized as follows: Section II presents our problem; Section III introduces instance sets, and the way they have been selected; Section IV-A presents the Hybrid Local Search solver and its initial structure; we detail different parameters available for our study in Section IV-B; we develop our experimental protocol in Section V; Section VI presents the new configurations and their performance comparing to two known methods. Finally, Section VII concludes the paper and outlines the potential for future research.

II. Curriculum-Based Course Timetabling

The International Timetabling Competition 2007 proposed a CB-CTT benchmark that included 21 different instances. These represent real-world problems from the University of Udine. Moreover, ITC 2007 formalized aspects of CB-CTT, now used in the literature.

The main characteristic of this problem is the use of the notion of "curriculum". CB-CTT does not consider each student individually. In the CB-CTT system, a student chooses a curriculum, which corresponds to a training course and includes a set of Courses. The notion of curriculum simplifies the problem for the solver as it reduces the number of conflicts by grouping students with similar behaviors. Courses correspond to a subject taught by a single teacher with a set number of lectures. A course can belong to several curricula. In this case, all the students in those curricula will attend the same lectures of this course. Most CB-CTT problems take place over a week, with the number of days varying between 4 to 7, depending on the specific instance. Each day is divided into time slots or periods, with a lecture typically lasting one period.

A CB-CTT solution consists of scheduling lectures in timeslots and available rooms following hard and soft constraints. Hard constraints must always be respected. A timetable is said to be feasible when all the hard constraints are met. For example, one hard constraint forbids one teacher from teaching two lectures at once. On the contrary, soft constraints can be violated. The violations of each soft constraint are represented as a function to minimize.

The objective function to optimize for Curriculum-Based Course Timetabling is a weighted sum of the soft constraint violations:

$$f(s) = \sum_{i=1}^{4} \mathit{SoftConstraints}_i(s) * \omega_i \quad (1)$$

s represents a timetabling solution. The weights, as used for ITC 2007, are set to 1, 5, 2, and 1 for ω_1, ω_2, ω_3 and ω_4 respectively. $\mathit{SoftConstraints}_i(s)$ represents the number of violations for the soft constraints listed below.

1) *RoomCapacity*: One violation for each student without a seat during one timeslot.
2) *MinWorkingDays*: A course has lectures that should be scheduled within a minimum number of days.
3) *CurriculumCompactness*: A student should have always two consecutive lectures before a gap.
4) *RoomStability*: Lectures of a course should be in the same room; one violation for every extra room used.

III. Instance Set

Instances of the Curriculum-Based-Course Timetabling problem used in our experiments come from a recent article proposed by de Coster et al. [3]. These instances are of two classes: real-world and generated. The real-world instances correspond to the real problems of several universities. This set includes the initial benchmarks of 21 instances provided by ITC 2007. Once we remove the problems where no initial solution can be constructed in a reasonable time (5 minutes), 77 instances remain. The second set corresponds to the generated data, i.e. instances generated by a model simulating real-world problem behavior. [3] details the generation process. Preliminary results have shown that an important part of these instances is infeasible. So we use these results to remove the impossible instances and keep almost 3 000 artificial instances. These generated instances are used by the irace configurator in our work.

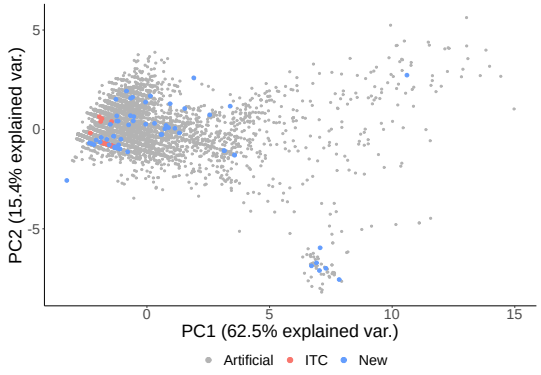

Fig. 1: PCA projection of instance sets.

Figure 1 shows the spatial distribution of the instances when plotted in a two-dimensional space. Principle Component Analysis (PCA) is used for dimensionality reduction and is applied on the descriptive data of the instances (i.e., the number of rooms, professors, ...). This shows that the artificial data, our training set, are similar to the real-world instances and fill the gaps between them. Furthermore Figure 1 shows the two subsets among the test data real-world instances: the original ITC instances and the newer ones from [3]. These do not have the same distribution.

IV. HLS Framework

A. HLS Description

The major event that formalized the Curriculum-Based Course Timetabling problem was ITC 2007 and especially its competition. The winning solver of this event was a Hybrid Local Search (HLS) proposed by Müller [13]. HLS is still the state-of-the-art method of the literature to solve the CB-CTT instances [3]. Moreover, it also competes in recent ITC editions on variants of the University Course timetabling problem [11]. HLS is split into two independent phases: the first one constructs an initial solution, namely the constructor [12]. Then the second one, called the solver, optimizes and improves results until a time limit is reached. The remainder of the paper focuses on the optimization phase.

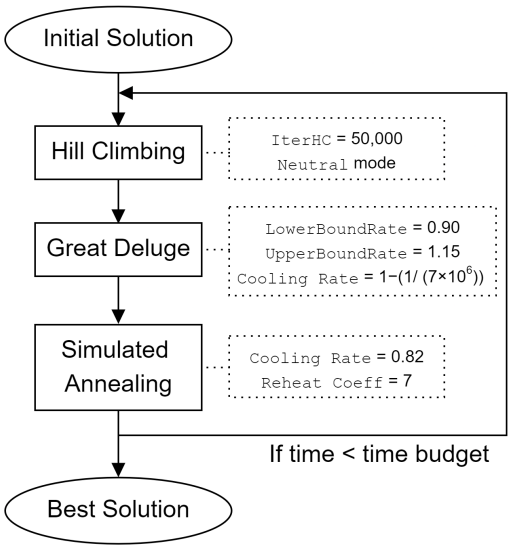

Fig. 2: Composition of HLS method.

HLS sequentially executes three heuristics and loops until the termination criterion is reached, for specifically, a maximal runtime (Figure 2). Hill Climbing is first applied, followed by Great Deluge [4] and finally Simulated Annealing [9] ends the sequence. These heuristics are neighborhood-based methods that use neighborhood operators to build a set of candidate solutions for each current solution. In HLS, five neighborhood operators are used. First, there are two simple operators `TimeMove` (Op1) and `RoomMove` (Op2). Each one randomly selects a lecture from the timetable and `TimeMove` selects also a timeslot and `RoomMove` a room. Then, the operator reschedules the lecture with the second item, timeslot or room chosen. If there is a conflict, another random item is selected. `RsMove` (Op3) and `MwMove` (Op4) are considered as specific constraint operators. Each one changes the timetable to decrease the number of violations of a specific constraint. For example, `MwMove` selects a course whose lectures are not well distributed, i.e., it does not respect the minimum number of different days. Then, it performs several `TimeMoves` to make sure that there are no more violations. Finally, the fifth operator is the `LMove` (Op5). It selects one lecture, a timeslot, and a room at random. Then it schedules the chosen lecture in the timeslot and the room. If there is a conflict, `LMove` swaps the other conflicting lecture.

The three neighborhood-based heuristics used in HLS are tunable. The strategic components and the parameter values of the original HLS have been fixed manually and experimentally to give good performance on the initial 21 instances presented during the competition ITC 2007. Even on the other instances of the literature (see Section III), HLS remains the best solver for the CB-CTT. In this paper, we are interested to see if the strategic components and parameters are well tuned. Therefore, we will describe each neighborhood-based heuristic in order to highlight the main components and parameters to give a general framework of HLS. We limit this study keeping the same sequential order of the three heuristics, namely HC, GD and SA; but we add a mechanism to activate or not each heuristic during the execution of the HLS framework.

Moreover, the definition of the neighborhood is shared by the three heuristics. In HLS, the five operators are used together. We allow each operator to be activated or not across the three heuristics.

Hill Climbing is executed until a number of successive candidate solutions without improvement has been visited controlled by the parameter `IterHC` set to 50 000 in the original HLS. Moreover, in the implementation of HLS, better (strictly or equivalent) neighboring solutions are accepted during the neighborhood exploration phase. We add an activation mechanism `NeutralStrict` to give the possibility of only accepting the strictly better neighboring solutions.

Great Deluge accepts all candidate solutions below an upper bound that decreases as the search progresses following a cooling schedule fixed to $1 - 1/(7 \times 10^6)$. Great Deluge is stopped when the value of the upper bound is below that of the lower bound. As soon as Great Deluge starts, the bounds are computed using an upper bound rate (`UpperBoundRate`) and a lower bound rate corresponding to a percentage of the current solution fitness. In HLS, these rates are set to 1.15 for the upper bound and 0.95 for the lower bound meaning that it accepts candidate solutions until 15 % of degradation at worst. If no improvement occurs during an iteration the upper bound is raised by a power factor for the next iteration.

Simulated Annealing uses a cooling schedule controlled with a `CoolingRate` set to 0.82. As for HC, SA is executed

until a number of successive candidate solutions without improvement has been visited. This number is controlled by the `Reheat Coeff` set to 7 in the original HLS. SA presents many other parameters linked to the temperature that have been experimentally set. However, in preliminary experiments, we observed that SA was mostly deactivated in the best configurations provided by the configurator.

B. HLS Parameters

Group	Parameter	Value
Neighborhood Operators	Op1 (`TimeMove`)	Boolean
	Op2 (`RoomMove`)	Boolean
	Op3 (`RsMove`)	Boolean
	Op4 (`MwMove`)	Boolean
	Op5 (`LMove`)	Boolean
Hill Climbing	`HC`	Boolean
	`NeutralStrict`	Boolean
	`IterHC`	$(10^4; 5 \cdot 10^4; 10^5)$
Great Deluge	`GD`	Boolean
	`UpperBoundRate`	(1.10; 1.15; 1.50; 2)
Simulated Annealing	`SA`	Boolean
	`Reheat Coeff`	(1; 7; 14)

TABLE I: Parameters set

Table I lists the 12 parameters we have extracted for this work and their ranges. They are classified into 4 groups: the neighborhood operators and one group per heuristic. Each neighborhood operator and each heuristic can been activated (1) or not (0), so it is managed with a boolean value. If a heuristic is deactivated, HLS then moves to the next heuristic activated heuristic. Note that a security function prevents from deactivating all heuristics together. In addition, the parameter `NeutralStrict` that controlled the criterion acceptance during the neighborhood exploration is a Boolean value where 0 means that equivalent neighbors are accepted while 1 means that only strictly better neighbors are accepted. The other parameters are numerical. In order to control the size of the configuration space, we decided for this first study to limit the possible values. For each parameter, we allow at least one smaller and one larger value than the value set in the original HLS. Finally, the configuration space contains 2 449 configurations.

V. Experimental Protocol

In the previous section, we gave a detailed description of the HLS framework starting from an initial solution to provide an optimized solution. Our goal is to investigate if specific configurations of HLS can improve the performance of the original HLS on the 77 real-world instances used in the literature (see Section III). In order to find these specific configurations, a configurator is run over the configuration space presented in Table I. The process of automatic configuration needs training instances independent of ones used to evaluate the final performance of the configurations. In Section III, we identified about 3 000 artificial instances that have been synthetically generated from the real-world instances which make them good candidates for the training. For both real-world and artificial instances, we applied the construction heuristic of the original HLS [13] to build and obtain solutions that will be used as initial solutions of the optimization process.

In this paper, we choose irace [10] to be the automatic algorithm configurator. irace implements an iterated racing procedure and uses statistical tests to identify the best configuration from the configuration space. It starts the first iteration with sampled configurations and runs them on some instances (often equal to five). At the end of the iteration, it eliminates the statistically dominated configurations. Then, for the next iterations, it generates new configurations derived from those selected in the previous iteration. irace iterates these processes until its allocated budget is reached.

For our experiments, we give to irace a budget of 2 000 runs using artificial instances. One run corresponds to one execution of one configuration on one instance. The total number of possible configurations is equal to 2 449 and takes into account the conditional values between the parameters. The configuration corresponding to the original HLS (see Table II) is given to irace for the first iteration. When the budget is reached, irace provided elite configurations that are statistically equivalent on the training instances used. We keep five of these elite configurations to validate the performance on the test instances. The original configuration and the five elites are executed 30 times with different seeds on each of 77 real-world instances The performance is the best solution reached for each run after 5 minutes. All performance runs are performed on an Intel(R) Xeon(R),W3520 @ 2.67GHz processor.

VI. Results

The primary objective of our study is to determine if it is possible to create a solver that performs better than the original HLS using the five top elite configurations returned by irace. In this section, we present and analyze the five elites and we compare their performance with the original HLS configuration.

A. Analysis of Configurations

Table II reports the values of the parameters of the five elites configurations. The "Method" column lists all the configurations, including the original HLS and gives an insight of which neighborhood-based heuristics are activated. The five next columns correspond to the neighborhood operators and whether they are activated or not (1 for activated, 0 for not). Then, three columns concern the Hill Climbing phase. The next two columns are about the Great Deluge component. The remaining columns are about the Simulated Annealing component. In order to reduce the potential noise, we have launched several irace runs with the same parameters but different seeds. We obtained the same configuration structures. Only some small parameters varied and the order of the configurations. We selected the most frequent one.

Method	Op1	Op2	Op3	Op4	Op5	HC	Neutral0Strict1	IterHC	GD	UpperBoundRate	SA	Reheat Coeff
original HLS	1	1	1	1	1	1	0	50,000	1	1.15	1	7
C1 (GD)	1	1	1	0	1	0	NA	NA	1	2	0	NA
C2 (HC+GD)	1	1	1	0	1	1	0	50,000	1	1.50	0	NA
C3 (HC+GD)	1	1	1	0	1	1	0	10,000	1	1.50	0	NA
C4 (GD)	1	1	1	0	1	0	NA	NA	1	1.50	0	NA
C5 (HC+GD)	1	1	0	0	1	1	0	10,000	1	2	0	NA

TABLE II: Parameter values of the original HLS and the five elites.

A first take-away result is that non of the elite configurations returned by irace features all three algorithmic components of HLS and, actually, Simulated Annealing is never chosen. This result is in contrast to the observations made by Feutrier et al. [6]. This paper considers that Simulated Annealing has a positive impact on performance. Moreover, Table II underlines that only Great Deluge with a strong perturbation is always privileged by irace. Indeed, all the proposed configurations use 1.5 or 2, the two largest values for the GD `UpperboundRate` parameter. Great Deluge is a heuristic which, like Simulated Annealing, accepts non-improving solutions if they have a fitness lower than a bound. The latter decreases during the iterations. The larger the `UpperboundRate`, the more time the Great Deluge takes and the more tolerant it is of large perturbations. That represents a clue to understand what makes better solvers, but should not be considered only individually. That may be due to several combined effects.

The original HLS uses all the operators. The elites activate all operators except Op4 which corresponds to the MinWorkDaysMove. It focuses on a subpart of a timetable that violates one constraint and generates only improving solutions regarding this constraint. It is thus more complex than others. Operators Op1, 2, and 3 are like random kicks. We hypothesize that this operator is slower because of the complexity of its task. Indeed, it must find the lectures of a course scheduled on the same day and place them on different days. It may seem more efficient to perform a succession of simple random kicks.

B. *Analysis by Fitness Scores*

The result of the experiments of the configurations executions on the test set is an array of data with 77 real-world instances, 6 solvers (original HLS and the 5 elites), and 30 seeds, corresponding to 13 860 fitness scores. In order to exploit these data and compare the performance globally, we normalized the fitness scores by real-world instance using a Min-Max scaling method. In this subsection, we will compare the configurations, the original one and the five elites, according to their fitness scores. Then in the next subsection, we will focus the analysis on the computed ranks to re-rank the configurations in order to reduce outlier behaviors.

Figure 3, 4 and 5 show the distributions of normalized fitness scores of the configurations, respectively over all, ITC, and newly added instances. The elites C1, C2, C3, and C4 seem to perform better than the original HLS and the elite C5. Moreover, these first four configurations seems to give a more stable performance than the others. In order to rigorously compare the configurations, we calculate the ranks of the methods following the procedure below. We first use a Friedman statistical test, with a significance level of 0.05, to check that the distributions of the normalized fitness scores for each method are different. The Friedman test obtains a p-value $< 2.2 \times 10^{-16}$, which means there is a statistical difference in global performance between configurations. This test only tells that a difference is present. To obtain actual ranks, we proceed as follows. The mean of scaled fitness values for each solving method is computed in order to sort configurations by ascending order of mean fitness. Recall that CB-CTT is a minimization problem, so we look for the lowest values. The sorted algorithms are compared using a Wilcoxon test (also known as the Mann-Whitney test) to check

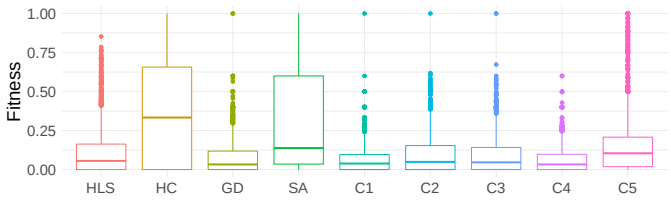

Fig. 3: Distribution Fitness Scaled per configuration over all instances.

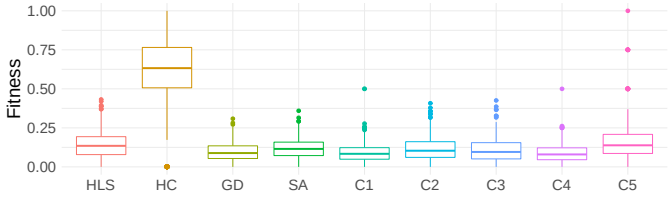

Fig. 4: Distribution Fitness Scaled per configuration over ITC instances.

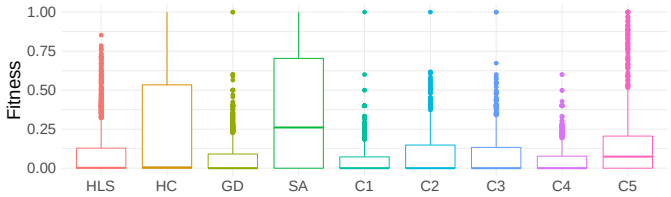

Fig. 5: Distribution Fitness Scaled per configuration over New real-world instances.

for actual statistical difference. Again a significance level of 0.05 is used. The first ordered configuration (the one with the best mean) is ranked 1 and is statistically compared with the following ordered configurations until the Wilcoxon test rejects the equality hypothesis for one configuration. Then, this configuration is ranked 2 and the previous ones are ranked 1. Then, we test the configuration ranked 2 with the ordered configurations that follow. The procedure is detailed in Algorithm 1. Table III reports the ranks calculated for each configuration. The activated heuristics are specified between parentheses.

Algorithm 1 Ranking Procedure

```
rank ← 1
index_ref ← 1
methods ←list of methods sorted by mean fitness
rank_list[methods[1]] ← rank
for index in index_ref + 1 to |methods| do
    method1 ← methods[index_ref]
    method2 ← methods[index]
    result ←Wilcoxon.test(method1, method2)
    if result : methods are not equivalent then
        rank ← rank + 1
        index_ref ← index
    end if
    rank_list[method2] ← rank
end for
return rank_list
```

Instance Set	HLS	HC	GD	SA	C1	C2	C3	C4	C5
All	4	6	2	6	1	3	3	1	5
ITC	4	5	2	3	1	3	2	1	4
New	3	4	1	5	1	2	2	1	4

TABLE III: Ranks on Normalized Fitness values.

Table III (line *All*) confirms that elite configurations C1, C2, C3 and C4 outperform the other two. Moreover, C1 and C4 are the elites that give the best performance to solve real-world instances. We can remark that the order of elites given by irace is not confirmed here. This is due to a difference between training and test instances. However, the main important result here is that it is possible with fine-tuning to find configurations of HLS that better perform than the original one. To support this conclusion, we compute the frequencies of the ranks of each configuration. Figure 6 shows the computed frequencies. If we focus on HLS ranks, we see that it is outperformed by at least one other configuration on 42 instances ($77 - 35$).

Table III also provides the rank if we separate the real-world instances into ones from the ITC 2007 competition and the others (called *New*). The ranks are globally the same, but, surprisingly, HLS gets better performance on the new instances even though it was manually tuned on the ITC ones.

We notice that elite configurations C1 and C4, the best ones, do not have a Hill-Climbing phase. The three other

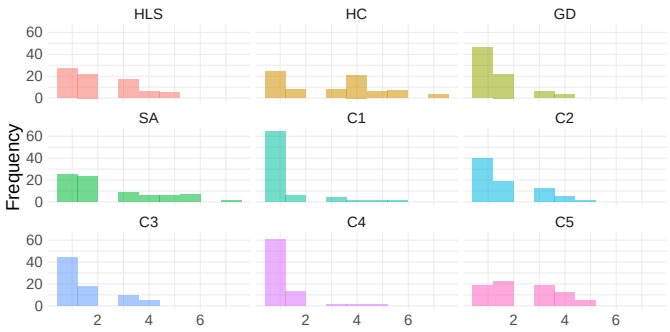

Fig. 6: Frequency of Rank by Config.

configurations use a relatively quick HC according to their value of `IterHC`, and all of them use neutral acceptance.

C. Analysis by Ranks

The disadvantage of considering the normalized fitness values across all instances is that it does not take into account the behavior w.r.t. each instance. In this section, we consider the per-instance rank in order to use instance-level information. To do so, we repeat the ranking procedure, one instance at a time, using the raw fitness scores.

The goal is to better assess whether a variant is really efficient on a subset of instances and just intermediate on others. We first analyze the distribution of per-instance ranks. Figure 7, 8, and 9 show these distributions for each configuration, respectively over all instances, ITC instances, and newly added real-world instances. They show similar behavior. The first four elites and the original HLS have a median rank equal to 1. If we analyze the raw data, there are many real-world instances where the configurations are equivalent, especially on the easier instances.

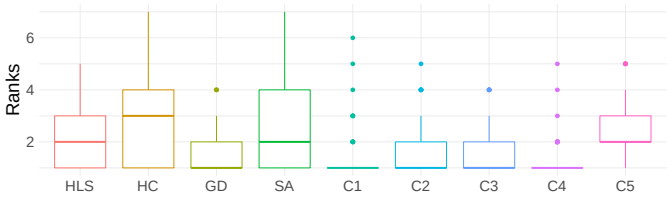

Fig. 7: Value of Ranking by Method over all real-world Instances.

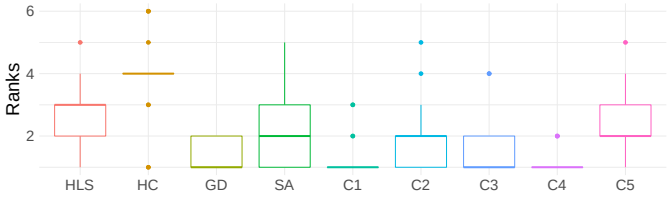

Fig. 8: Value of Ranking by Method over ITC real-world Instances.

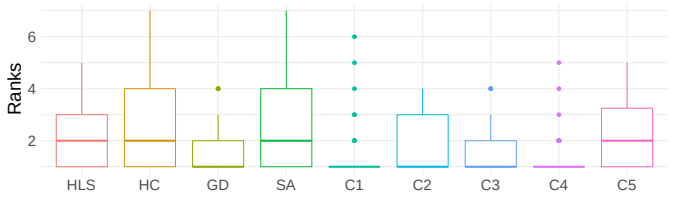

Fig. 9: Value of Ranking by Method over New real-world Instances.

We observe three main different behaviors. The first concerns C5. Its wider boxplot and median at 2 indicate worst performance than the other variants and match the rank obtained on the normalized fitness values. Original HLS, together with C2 and C3 have the same boxplots. However, this does not mean that the ranks are distributed in the same way, only that they share the same summary statistics. The two best variants are C1 and C4, both only use Great Deluge.

These distributions of per-instance ranks seem to match to a certain extent the results obtained on the ranks of normalized fitness values. To further the observations from the analysis of Figure 7, we use a ranking method on the sum of ranks per instance by each solver. In addition, we explicitly isolate the ITC 2007 instances from the rest, to see whether HLS performs better on this subset of instances it was originally designed to solve.

Instance Set	HLS	HC	GD	SA	C1	C2	C3	C4	C5
All	6	9	3	8	2	5	4	1	7
ITC	8	9	3	6	2	5	4	1	7
New	6	9	3	8	2	5	4	1	7

TABLE IV: Global Ranks on the ranks per instance.

Table IV shows the ranking of each method considering ranks per instance. The ranking is also performed on subgroups of instances. The first subset, named ITC, includes the initial CB-CTT benchmark consisting of 21 instances from the University of Udine. The "New" group contains the remaining newer real-world instances.

The new ranks provide additional information to complement the boxplots and previous ranks. With this approach, the HLS, C2, and C3 are not considered as statistically similar. Indeed, HLS is ranked worse than four of the five elites configurations proposed by irace, all of which are algorithmically simpler.

D. Ablation

Ablation analysis [5] is a method used to study the effect of changing specific parameters of a solver on its overall performance. The ablation analysis takes a set of instances to test several configurations. Moreover, this process needs two initial configurations. A starting configuration and a target configuration. The method will test a part of the configurations that are between these two initial configurations. The algorithm tests the starting configuration, here HLS, on the given instances, here the real-world, test set. During the first iteration, the method generates new configurations based on the HLS default configuration, the starting method. But these configurations have one parameter value that differs from the starting configuration, that value is set to the same value as in the target. After the performance tests, statistical tests determine the best of the new configurations. The improvement or degradation compared to the previous starting configuration is memorized. On the next iteration, the new starting configuration is the one that is considered the best in the previous iteration. So, on the second iteration, the starting configuration is a configuration that has the same parameters as HLS except for one whose value is equal to the target. The program generates new configurations similar in terms of parameters to this new starting configuration except for one of them. The iterations continue until the target configuration is obtained again. At the end of the analysis, we get information about the change of values and parameters that allowed us to go from HLS to the best configuration.

Here we consider C1 as the target. Indeed, we keep C1, and not C4, because it is the most efficient according to irace on the train set. Anyway, C1 and C4 are very close as detailed previously.

Ablation analysis highlights the most important parameters and their values to improve performance. In this paper, ablation analysis takes the set of instances called *All* in the previous section, which contains all of the 77 real-world instances. The ablation compares two solvers that both have a Great Deluge phase, which is why this parameter is not studied by the algorithm. We previously highlighted that the best configurations all activate this feature.

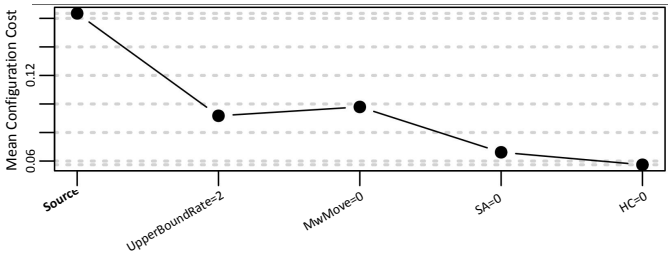

Fig. 10: Mean configuration cost computed by the ablation analysis.

Figure 10 shows the order in which the parameters were changed. This is equivalent to giving an order of importance for these two configurations. Furthermore Figure 10 gives the average scaled cost gain on the 77 real-world instances.

The ablation method shows that the most improving change of value from HLS is to set the `UpperBoundRate` to 2. That means increasing the power of perturbation of the Great Deluge phase contributes to the efficiency of GD. Additional runs showed that setting `UpperBoundRate` to 1.43 would increase performance in the case of C4 configuration.

The second most important feature highlighted by ablation is `MwMove`, which corresponds to the activation of the Min-

WorkingDays Move operator, Section IV-B. Ablation results conclude that the improvement due to the activation of this operator is less significant than for `UpperBoundRate`. However, it is still significant compared to the other two. That fact validates our analysis in the previous section. This operator must slow down the solver a lot or not be efficient enough to be worth it. The importance of deactivating SA is finally minimal. In Section IV-B SA, was like MinWorkingDays move, always deactivated. And yet, these are not the most important parameters, i.e. chosen first. So there may be combinations of parameters where SA and GD are effective. This would require further experimentation to ascertain.

The last parameter chosen by ablation is HC. That means Hill Climbing has no concrete impact on the performance when changing HLS to C1. Ablation results advise setting this parameter to 1. Moreover, it considers that if HC is deactivated that decreases the performance of the solver. However, our performance tests say that variants with HC are worse than GD only.

In summary, the ablation analysis showed that `UpperBoundRate` was the most important parameter in the success of C1 and C4. That offers the opportunity to work in the future on SA and HC, and their parameters. Consequently, these features become significantly important in this kind of study.

VII. Conclusion

In this paper, we work on the Curriculum-Based Course Timetabling problem, a variant of university course timetabling problems. We consider as a starting point a state-of-the-art method, Hybrid Local Search. First, we propose parameters whose changes create new configurations. Unlike the papers that have examined this solver, some parameters are focused on changing the structure of HLS. They disable entire heuristics. We also let more classical numerical parameters alter the behavior of the search, such as modulating the length of the optimization. Some parameters disable neighborhood operators. An automatic algorithm configurator, irace, returns five elite configurations using HLS as starting point. After a performance analysis on the 77 real-world instances of our test set, the first observation is that some configurations are better than HLS. Indeed, irace produces only one configuration that is worse than HLS. The two best configurations, as identified by statistical tests, both consist only of Great Deluge. That means that, in order to obtain better configurations, irace removed heuristics to propose a simpler method. Moreover, analysis of the chosen parameters shows that all the configurations are simpler than HLS, by removing SA essentially. Our work shows that the performance is increased while removing heuristics from HLS. That improvement is due to a combination of parameters. For example, the presence of some neighborhood operators is important: RoomStabilityMove. Finally, the ablation analysis emphasizes the importance of the choice of parameter values. Ablation determines which parameters allow increasing the performance of the configurations from HLS to C1. That shows us that it is not SA that is most important but the power of the GD perturbation. It does not seem from this analysis that SA is the most important to disable. Thus, the Great Deluge seems to be an efficient solver for the CB-CTT. Future work will focus on improving this solver. We intend to add new parameters to abstract from the limitations stated in this paper. At the same time, we will expand the range of existing variables, to continuous instead of discrete domains, to get a clearer idea of the optimal configuration. Consequently, we will use a higher budget for irace. Moreover, a deeper study of the combinations of parameter values is required to fully understand the factors at play.

References

[1] Bettinelli, A., Cacchiani, V., Roberti, R., and Toth, P. An overview of curriculum-based course timetabling. *23*, 2 (2015), 313–349.

[2] Chiarandini, M., Birattari, M., Socha, K., and Rossi-Doria, O. An effective hybrid algorithm for university course timetabling. *Journal of Scheduling 9*, 5 (2006), 403–432.

[3] De Coster, A., Musliu, N., Schaerf, A., Schoisswohl, J., and Smith-Miles, K. Algorithm selection and instance space analysis for curriculum-based course timetabling. *Journal of Scheduling 25*, 1 (2022), 35–58.

[4] Dueck, G. New optimization heuristics: The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics 104*, 1 (1993), 86–92.

[5] Fawcett, C., and Hoos, H. H. Analysing differences between algorithm configurations through ablation. *Journal of Heuristics 22*, 4 (2016), 431–458.

[6] Feutrier, T., Kessaci, M.-E., and Veerapen, N. Investigating the landscape of a hybrid local search approach for a timetabling problem. GECCO '21, Association for Computing Machinery, pp. 1665–1673.

[7] Feutrier, T., Kessaci, M.-E., and Veerapen, N. Analysis of search landscape samplers for solver performance prediction on a university timetabling problem. In *Parallel Problem Solving from Nature – PPSN XVII* (2022), G. Rudolph, A. V. Kononova, H. Aguirre, P. Kerschke, G. Ochoa, and T. Tušar, Eds., Lecture Notes in Computer Science, Springer International Publishing, pp. 548–561.

[8] Hutter, F., Stuetzle, T., Leyton-Brown, K., and Hoos, H. H. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research 36* (2009), 267–306.

[9] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by simulated annealing. *Science 220*, 4598 (1983), 671–680. Publisher: American Association for the Advancement of Science.

[10] López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., and Birattari, M. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives 3* (2016), 43–58.

[11] Muller, T. ITC 2019: Preliminary results using the UniTime solver.

[12] Müller, T. *Constraint-based Timetabling Ph. D.* phdthesis, Charles University, Faculty of Mathematics and Physics, 2005.

[13] Müller, T. ITC2007 solver description: a hybrid approach. *Annals of Operations Research 172*, 1 (2009), 429.

[14] PATAT. International Timetabling Competition 2007, 2007. Publication Title: International Timetabling Competition.

[15] Song, T., Chen, M., Xu, Y., Wang, D., Song, X., and Tang, X. Competition-guided multi-neighborhood local search algorithm for the university course timetabling problem. *Applied Soft Computing 110* (2021), 107624.

[16] Soria-Alcaraz, J. A., Ochoa, G., Swan, J., Carpio, M., Puga, H., and Burke, E. K. Effective learning hyper-heuristics for the course timetabling problem. *European Journal of Operational Research 238*, 1 (2014), 77–86.

[17] Teixeira, U. R., Souza, M. J. F., de Souza, S. R., and Coelho, V. N. An adaptive VNS and skewed GVNS approaches for school timetabling problems. In *Variable Neighborhood Search* (2019), A. Sifaleras, S. Salhi, and J. Brimberg, Eds., Lecture Notes in Computer Science, Springer International Publishing, pp. 101–113.