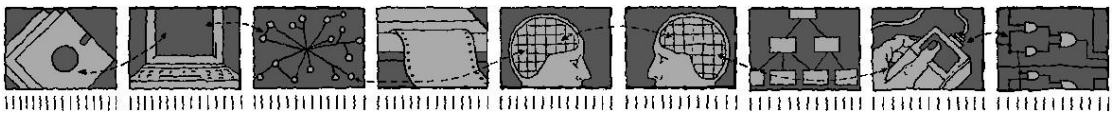
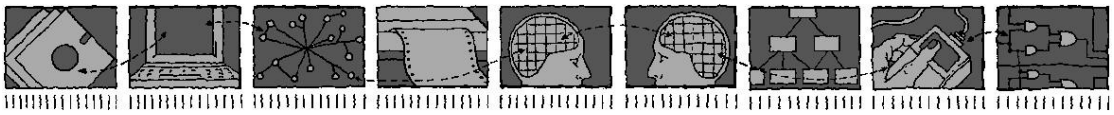


Department of Computing Science and Mathematics
University of Stirling



The Late Acceptance Hill-Climbing Heuristic

Edmund K. Burke, Yuri Bykov

Technical Report CSM-192

ISSN 1460-9673

June 2012

Department of Computing Science and Mathematics
University of Stirling

The Late Acceptance Hill-Climbing Heuristic

Edmund K. Burke

Department of Computing Science and Mathematics
University of Stirling
Stirling FK9 4LA, Scotland
Telephone +44-1786-467020, Facsimile +44-1786-467016

Email e.k.burke@stir.ac.uk

Yuri Bykov

School of Computer Science
University of Nottingham, Jubilee Campus, Wollaton Road
Nottingham NG8 1BB, UK
Telephone +44-115-846-8376

Email yuri.bykov@nottingham.ac.uk

Technical Report CSM-192

ISSN 1460-9673

June 2012

Abstract

This paper introduces a new and very simple search methodology called Late Acceptance Hill-Climbing (LAHC). It is a one-point iterative search algorithm, which accepts non-improving moves when a candidate cost function is better (or equal) than it was a number of iterations before. This value appears as a single algorithmic input parameter which determines the total processing time of the search procedure. The properties of this method are experimentally studied in this paper with a range of Travelling Salesman and Exam Timetabling benchmark problems. In addition, we present a fair comparison of LAHC with well-known search techniques, which employ different variants of a cooling schedule: Simulated Annealing (SA), Threshold Accepting (TA) and the Great Deluge Algorithm (GDA). Moreover, we discuss the method's success in winning an international competition to automatically solve the Magic Square problem. Our experiments have shown that the LAHC approach is simple, easy to implement and yet is an effective search procedure. For all studied problems, its average performance was distinctly better than GDA and on the same level as SA and TA. One of the major advantages of LAHC approach is the absence of a cooling schedule. This makes it significantly more robust than cooling-schedule based techniques. We present an example where the rescaling of a cost function in the Exam Timetabling Problem dramatically deteriorates the performance of three cooling-schedule based techniques, but has absolutely no influence upon the performance of LAHC.

Keywords: combinatorial optimization, heuristic, local search, travelling salesman, timetabling, hill climbing, simulated annealing, threshold accepting, great deluge algorithm.

1. Introduction

A heuristic search paradigm, known as the *one-point iterative search* is one of the earliest and most well-studied heuristic approaches in the field of Operations Research. Indeed, it is associated with a wide range of techniques (metaheuristics), across a broad spectrum of problems. Generally, one-point search algorithms have the following advantages: they are relatively simple in implementation, computationally inexpensive and quite effective for large-scale problems. All of these techniques have certain common properties, which distinguish them from other heuristic methods. In general, such a search is started from some (usually random) initial solution, which plays the role of "*current solution*" at the first iteration. The stochastic search procedure performs as a sequence of iterations (moves). At each iteration, the current solution is somehow modified in order to produce a candidate solution. The candidate can be accepted or rejected according to a given acceptance condition. If it is accepted, then it serves as the current solution for the next iteration. If the candidate is rejected, then the next iteration is carried out with the same current solution. Usually the search lasts until no further improvement is possible (i.e. convergence). Different one-point search methods are distinguished by their acceptance condition, which is usually based on an evaluation of the cost functions of generated solutions. The simplest one-point search algorithm is the greedy Hill-Climbing (HC) methodology, which was one of the earliest studies undertaken in this field (Appleby, Blake and Newman 1960). The HC accepts only candidates with the same or better cost function value than the current one. This method is regarded to be very fast, but not sufficiently powerful as it usually tends to get stuck quickly in a local optimum.

The effectiveness of such a search procedure can be increased by employing an alternative acceptance condition, which allows the acceptance of candidates with lower (worse) scores of the cost function. Of course, if all candidates are accepted then the search degenerates into just a series of random perturbations. However, over the years, several methodologies have been introduced into the literature for selecting candidates with worse cost function scores, which can be accepted. In many one-point search algorithms, this mechanism is based on a so-called *cooling schedule* (CS). The common property of these methods is that their acceptance condition is regulated by an arbitrary *control parameter* (such as *temperature*, *threshold* or *water level*), which is varied in the course of the search. At the beginning of the search, this parameter has some initial value, which should activate further iterations. To terminate the search procedure this parameter must reach its final value where no worsening moves are accepted and the search converges. The shape of the variation of the control parameter (cooling schedule) is defined by a user and it can have a significant impact on the overall performance of the algorithm. Over the years, a number of empirical recommendations have been put forward for composing effective cooling schedules for different problems.

One of the most well studied one-point search metaheuristics is Simulated Annealing (SA) proposed by Kirkpatrik, Gellat and Vecci (1983). It is a stochastic algorithm, which accepts a worse candidate with probability $P = exp[(C - C^*)/T]$. Here C and C^* are respectively the cost functions of a current and a candidate solution and T is the control parameter (called "temperature"). Its variation during the search characterizes the cooling schedule. Several authors have proposed initial temperatures so that a certain percentage of worsening moves are accepted at the beginning. Different sources suggest different values for this percentage. Examples include between 40% and 90% (Johnson et al 1989), 75% (Thomson and Dowsland 1996) and 95 % (Cohn and Fielding 1999). The final value of the temperature should be close to zero. One of the most popular cooling schedules (called "geometric cooling") is represented by the following expression: $T_i = T_{i-1}*\beta$, i.e. the temperature at the i^{th} iteration is equal to the previous temperature T_{i-1} multiplied by a user-defined cooling factor β ($0<\beta<1$). However, some authors have suggested the use of alternative functions, such as the "quadratic cooling schedule" (Anderson, Vidal and Iverson 1993) or even temporary increases in the temperature, for example, adaptive cooling (Thompson and Dowsland 1996) or reheating (Osman 1993).

In addition to Simulated Annealing, a number of other similar search techniques have been proposed. One that is particularly close to Simulated Annealing is the Threshold Accepting (TA) method, which is also known as "Deterministic Simulated Annealing" (Dueck and Scheurer 1990). Here, the candidate solution is accepted if $C^* - C \leq T$ where T is a control parameter (called the "threshold") and the cooling schedule represents the process for its modification. It should be noted that the above formula is given for minimization problems. This is the type of problem that we will consider in the rest of our discussion. Another deterministic variant (proposed by Dueck 1993) is the Great Deluge Algorithm (GDA). In contrast to TA, its control parameter B (called the "water level") serves as an upper bound of the candidate cost function. Thus, the algorithm accepts worse candidates with the cost equal or less than the current value of the level, i.e. when $C^* \leq B$. Now, the variation of the level

plays the role of the cooling schedule. In classical GDA, it was recommended that the initial level be equal to the initial cost function and that it should be lowered linearly during the search. However, other propositions have also appeared in the literature, such as: initialization with a higher level (Burke and Newall 2003), non-linear level lowering (Obit et al 2009) or reheating mechanisms (McMullan 2007). There are several other methods based on this pattern such as "Old Bachelor Acceptance" (Hu, Kahng and Tsao 1995) or "Weight Annealing" (Nino and Schneider 2005).

In all of these algorithms, the variation of the control parameter can be viewed as a type of cooling schedule. The cooling schedule has its strong and weak points. The strong point is that it enables an explicit way of processing time management. A range of methods, like the Genetic Algorithm (GA), the Ant Colony Optimization (ACO) or the Particle Swarm Optimization (PSO) have no such advantage. Obviously, having a completely user-controllable search procedure, the cooling schedule can be composed so that convergence will be achieved in exactly (or approximately) predefined processing time. This feature is important because in many situations increasing the search time can lead to better final results. However, in order to effectively use this property in practice, the time being employed by long search procedures should be put to best effect during the search (see Burke et al 2004).

A weak point of the cooling schedule is that its optimal form is problem-dependent. Even the tuning of scalar problem-dependent parameters is usually seen as a complex task. If we consider the performance of a method as an objective function, then the optimal tuning of the cooling schedule might represent a continuous optimization problem of even larger scale than the original problem. Of course, it is impossible to find this optimal cooling schedule manually. That is the reason for the existence of general empirical suggestions regarding cooling schedules, which are more or less effective for a range of studied problems. However, there is no guarantee that such a proposition will work for a new problem. In this paper, we present an example, where just a rescaling of the cost function dramatically deteriorates the performance of all the evaluated cooling-schedule based methods.

Another technique of interest is Tabu Search (TS) proposed by Glover (1986). This method evaluates the complete set of possible modifications of the current solution and the candidate with the best cost is accepted. To avoid cycling, it was proposed to also reject solutions, which were already (relatively recently) accepted at previous iterations. For this purpose, TS maintains a list of previous solutions (or moves) known as the "tabu list", where all elements are compared with the complete set of candidates at each iteration. The method has two interesting properties. Firstly, it does not employ any version of a problem-dependent cooling schedule. Secondly, as noted by Laguna and Glover (1996), the employing of a tabu list follows the idea of the "intelligent" use of information collected during the search. Later, this idea was expanded and called Adaptive Memory Programming (AMP), see Taillard et al (2001). In addition to Tabu Search, the authors also considered the GA and ACO as examples of AMP.

In this paper we introduce a new AMP-related technique. This keeps the advantages of the one-point procedure, but does not employ a cooling schedule. This technique was first publicly presented at the PATAT 2008 conference by Burke and Bykov (2008). This also builds upon work presented at the CEC'09 conference (Ozcan et al 2009). The description of our technique is given in the next section. In Section 3, we present a comprehensive experimental study of the properties of the proposed method, including a description of benchmark datasets, experimental software and several series of experiments. Section 4 contains a comparison of the new algorithm with existing techniques and the results are discussed in Section 5. In Section 6 we discuss the practical effectiveness of the proposed heuristic as evidenced by its success in the International Optimisation Competition. A summary, conclusions and further perspectives are presented in Section 7.

2. Late Acceptance Hill Climbing

In developing this search methodology, we had three clear goals in mind. Our first goal was to propose a one-point search procedure which does not employ an artificial cooling schedule. The second goal was to effectively use the information collected during previous iterations of the search. The third goal was to employ a new acceptance mechanism that was not complicated, i.e. to make it almost as simple as Hill-Climbing. As described above, in the greedy Hill-Climbing approach a candidate solution is compared with the immediate current one (and accepted if it is not worse). In respect of the specified goals, this paper is focused around the idea of delaying this comparison, i.e. *to compare a new candidate with that solution, which was the current several iterations before*. In other words, the only difference between our idea and greedy Hill-Climbing is that in the new algorithm each current

solution is employed during the later (not immediate) acceptance procedure. Therefore, we have named this technique the Late Acceptance Hill Climbing (LAHC) algorithm.

In a similar way to other one-point search metaheuristics (such as HC, SA, TA or GDA), LAHC is started from a randomly generated initial solution and at each iteration it evaluates a new candidate in order to accept or reject it. In order to employ its acceptance rule, LAHC maintains a list (of a fixed length) of previous values of the current cost function. The candidate cost is compared with the last element of the list and if not worse, then accepted. After the acceptance procedure, the cost of the new current solution is inserted into the beginning of the list and the last element is removed from the end of the list. Note that the inserted current cost is equal to the candidate's cost in the case of accepting only, but in the case of rejecting it is equal to the previous value.

"Memorizing" previous information in the form of a list is reminiscent of a similar mechanism employed in Tabu Search. However, the lists in TS and LAHC have a different nature and purpose: Firstly, in TS, we memorize solutions (or moves), but in the LAHC approach the list contains the values of a cost function. Secondly, in the TS at each iteration we compare its candidate solutions with the complete list, but in the LAHC approach only one value from the end of the list is used. These distinctions in memory utilization mechanisms also suggest that the list operations in LAHC are much less time consuming than in TS. Moreover, it is possible to make the processing time of LAHC genuinely independent of the length of the list by eliminating the shifting of the whole list at each iteration. For this purpose, we propose the *first improvement* of the initial idea, i.e. to use the "virtual" shifting of the list. Now the elements of the list are unmovable and the list appears as a fitness array F_a of length L_{fa} ($F_a = \{f_0, f_1, f_2, \ldots f_{Lfa-1}\}$). Its virtual beginning v, at the i^{th} iteration, is calculated as:

$$v = i \bmod L_{fa} \tag{1}$$

where "mod" represents the remainder of integer division. At each iteration, the value of f_v is compared with the candidate cost and after accepting or rejecting, the new value of the current cost is assigned to f_v.

The length L_{fa} appears as a single genuine input parameter for this algorithm. No other parameters are required. The LAHC performance is not affected by the initial values of fitness array. At the beginning of the search, the initial list can contain (generally speaking) any arbitrary values. If these are much higher than the initial cost, the algorithm will produce a corresponding number (equal to the L_{fa}) of random perturbations while filling the list with current costs. If all elements of the initial fitness array are too low, then the algorithm will produce the same number of non-accepted moves and again, will fill the fitness array with the value of the initial cost. Either variant can cause just a small delay in the search procedure. If we do not wish to wait until the algorithm does it automatically, then we could set up all elements of the fitness array to be equal to the initial cost before starting the search.

It should be noted that LAHC employs a greedy acceptance rule (rejects all worse candidates) only in the sense of the delayed comparison. However, if we consider a current solution and its immediate candidate, LAHC (in a similar way to SA, TA and GDA) allows the acceptance of worsening moves. This can happen in the situation where the current cost is better than the value from the list and the candidate cost is located between them. Taking into account that accepting worsening moves usually increases the strength of a search procedure, it could be expected that LAHC has a better performance than greedy HC. On the other hand, there are possible situations where the current cost is worse than the value from the list. Here (using the initial idea of LAHC), even a non-worsening move can be rejected. Such algorithmic behavior is usually not regarded as being desirable in computational search (SA, TA, and GDA always accept non-worsening moves). To be consistent with this practice, we propose a *second improvement* of the initial idea, i.e. to use the "late acceptance" rule for the worsening moves, but also to accept all non-worsening ones. Our initial experiments have revealed certain advantages to both improvements of the initial idea. All experiments in this paper were carried out with the final (improved) version of LAHC. Thus, its final acceptance condition at the i^{th} iteration can be expressed by Formula 2.

$$C_i^* \leq C_{i\text{-}Lfa} \quad \text{or} \quad C_i^* \leq C_{i\text{-}1} \tag{2}$$

In this formula, C_i^* is the candidate cost, $C_{i\text{-}1}$ is the current cost and $C_{i\text{-}Lfa}$ denotes the cost of the current solution L_{fa} iterations before, which is equal to $f_{(i \bmod Lfa)}$. Obviously, when L_{fa} is equal to 1 or 0, LAHC is simply greedy HC. Hence, LAHC obtains its unique properties with L_{fa} equal to 2 and higher. The pseudocode of the complete search procedure is shown in Figure 1.

```
Produce an initial solution s
Calculate initial cost function C(s)
Specify Lfa
For all k∈{0...Lfa-1} f_k:=C(s)
First iteration I=0;
Do until a chosen stopping condition
   Construct a candidate solution s*
   Calculate its cost function C(s*)
   v := I mod Lfa
   If C(s*)≤f_v or C(s*)≤C(s)
   Then accept the candidate (s:=s*)
   Else reject the candidate (s:=s)
   Insert the current cost into the fitness array f_v:=C(s)
   Increment the iteration number I:=I+1
```

Figure 1: The pseudocode of a general purpose Late Acceptance Hill Climbing

3. An investigation of the properties of the LAHC technique

3.1. Benchmark problems

The proposed LAHC does not employ the properties of any particular type of problems. Therefore, it can be positioned as a general purpose technique. We expect that it can be applied to any problem, where other one-point searches (HC, SA, TA, GDA) are applicable. To confirm its generality we present an experimental study with two types of problems: the Travelling Salesman problem and the Exam Timetabling problem.

The Travelling Salesman Problem (TSP) is a classical (and probably, the most well-studied) NP-hard combinatorial optimization problem. It has a range of real-world applications: from printed circuit board assembling to X-ray crystallography. Usually, this problem is formulated as a number of cities with different distances between them. The goal is to find a shortest closed tour while visiting each city only once. Over the years, most search techniques have been applied to the TSP. A detailed description of these studies can be found in a range of surveys, such as (Johnson and McGeoch 1997) or (Applegate et al 2006). There is also a well-known collection of TSP benchmark datasets of different sizes known as TSPLIB available at http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/. It contains 111 datasets among which we have chosen for our tests seven instances with sizes from 783 to 3795 cities (the sizes are given in Table 4 later in this paper).

Exam Timetabling is another well-studied NP-hard combinatorial optimization problem. It represents the real-world task of allocating university exams to a limited number of timeslots (and usually rooms). This problem can be viewed as an extension of the Graph Coloring Problem, with a high number of additional soft and hard constraints. Although there are different versions of the specification of the Exam Timetabling Problem (representing different institutional requirements, see (Burke et al 1996)) the most common hard constraint is that no one student should sit two exams at the same time. Other hard constraints specify room availability, matching the durations of exams and corresponding timeslots. The soft constraints represent the students and administrative preferences, such as a sufficient interval between exams taken by a student should be allowed or larger exams should be scheduled earlier. The goal is to allocate all exams while satisfying all hard constraints and minimizing the violations of the soft ones. Over the last 10-15 years, this problem has been intensively studied. For more information about the wide range of algorithmic approaches to this problem, the following survey papers can be consulted: (Carter and Laporte 1996), (Schaerf 1999), (Lewis 2008) and (Qu et al 2009). In this study we use the specification of the exam timetabling problem given at the 2nd International Timetabling Competition (ITC2007). Its description can be found in (McCollum et al 2010) and on the original competition web site: http://www.cs.qub.ac.uk/itc2007/. This site also contains 12 benchmark datasets, which we use in the experiments in this paper. Some characteristics of these datasets will be also given in Table 3. It should be highlighted that both the Travelling Salesman problem and the Exam Timetabling problem are minimization problems, so in all our experiments, the lower the result, the better.

3.2. Experimental software

The proposed technique was evaluated using special purpose software developed in Delphi 2007 and run on PC Intel Core 2 Duo, 1.86 GHz, 2 GB RAM under OS Windows XP. Two dedicated models

were created: the first one for the TSP and the second one for the Exam Timetabling Problem. Each of the models employs separate initialization procedures and a specially selected set of moves, which are explained below. Of course, the problem representation and procedures of the calculation of cost functions are also different in the models, but these are just a matter of implementation and do not affect the performance of the method.

The initialization of the TSP is quite simple: the initial tour (solution) is just randomly generated. At each iteration, a current solution is modified using a move, whose type is selected randomly. Currently the algorithm employs three types of move:

- The "double-bridge" move studied by Lin and Kernighan (1973). Here a current tour is randomly divided into two parts, which are reconnected in reverse order.
- The swapping of two randomly selected cities.
- The replacement of a random city into a different (randomly chosen) position in the tour.

The source code of our TSP software (its simplified variant, which employs only the first type of move) is available online at http://www.cs.nott.ac.uk/~yxb/LAHC/.

For our experiments with the Exam Timetabling problem, we have adapted software used in (Burke et al 2004). The major difference with that used for the TSP is a variety of hard constraints, which should not be violated in the final solution. To guarantee this, our model operates only with feasible solutions throughout the whole search process. Correspondingly, the feasible initial solution is generated by the "Saturation Degree" Graph Coloring heuristic, which was slightly adapted in order to take into account the allocation of exams to rooms. The range of applied moves was also chosen with the intention of preserving the feasibility. Hence, in a similar way to that undertaken in our TSP model, this algorithm randomly selects a move among the following types:

- The reallocation of a random exam into a different (randomly chosen) room.
- The replacement of a random exam into a different (randomly chosen) timeslot. Here the new room is also chosen randomly. If this move causes an infeasible solution, the algorithm runs a repairing procedure using Kempe Chains (see Johnson et al 1991).
- The swapping of two randomly selected timeslots including all their exams while keeping the room allocation invariable.

In addition we have to select a proper termination criterion for LAHC. The literature suggests three usual variants of the stopping condition:

- The algorithm stops after reaching some target solution (or the target value of cost function).
- The algorithm stops after a certain amount of time (or the number of iterations).
- The algorithm stops when reaching the convergence state (i.e. no further improvement is possible).

The choice of the termination criterion depends on the type of the problem, the properties of the search technique and on practical requirements. We will discuss LAHC stopping criteria in the next section.

3.3. Experiments

When a new algorithm is proposed an obvious question immediately arises: how do its input parameters affect its performance? Fortunately, for LAHC this question is not too difficult as it has a single parameter: the length of the fitness array L_{fa} . Therefore the initial series of experiments is started from this perspective.

In this first series of experiments, we investigate the basic properties of LAHC using time-cost diagrams. Such a diagram illustrates how the cost function drops throughout the search process. The algorithm was run with our benchmark problems several times while varying L_{fa}. During the search, the current cost was recorded every second. All these points are depicted (connected with a curve) in a plot, where the axes represent the current time and the current cost. Figure 2 presents an example of four curves drawn for the *Exam_1* instance: with L_{fa} =1 (which is the Greedy HC), 5000, 15000 and 30000. It should be noted that the time-cost diagrams plotted for other benchmark instances are similar to this example. This demonstrates the typical behavior of LAHC.

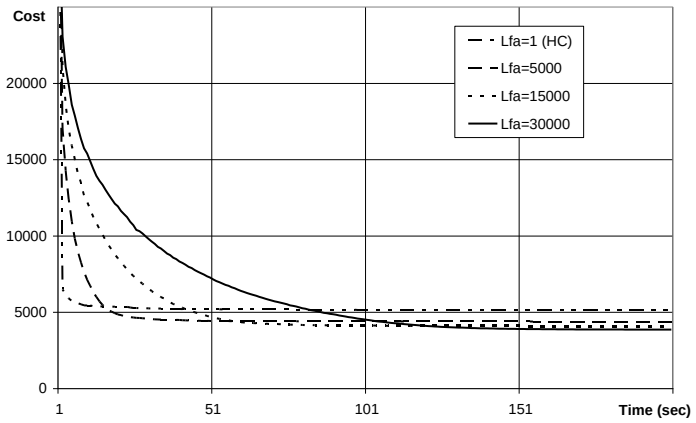


Figure 2: The time-cost diagrams of LAHC for *Exam_1* instance with different L_{fa}.

In this figure, all diagrams are started from the same initial solution with the cost function around 25000. The HC improves the cost almost immediately. The LAHC with L_{fa} =5000 improves the cost more slowly; with L_{fa} =15000 even more slowly and L_{fa} =30000 yields the slowest (over the four curves) improvement of the cost function. This observation suggests one of the major properties of LAHC: the longer the fitness array - the slower the cost drop. This behavior has been observed in all experiments with LAHC (either the presented in this study or other ones). Moreover, this behavior is typical of many other one-point search heuristics: in SA, TA and GDA the cost drop can be also regulated by adjusting their cooling schedules (see Burke et.al. 2004).

To explain the benefit of slowing the cost drop (by increasing the L_{fa}) we present, in Figure 3, a part of the above plot in a larger vertical scale.

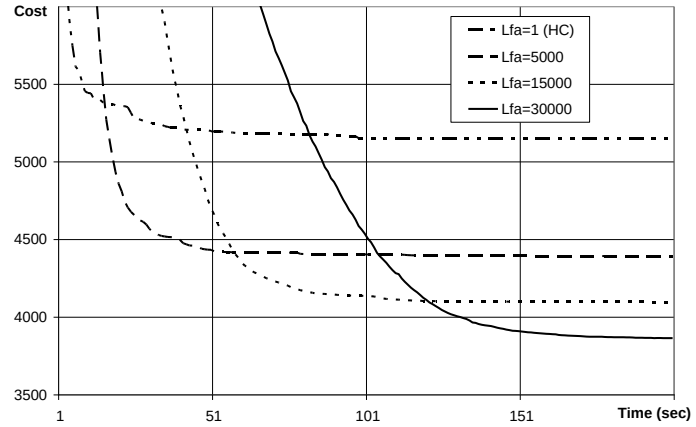


Figure 3: Part of the plot in Figure 2 in a larger vertical scale

First of all, the diagrams in Figures 2 and 3 demonstrate the second major property of LAHC: its convergence is quite clear. I.e. after reaching some value the cost drop is slowed dramatically and the diagram becomes flat. It is assumed that after convergence no further improvement is possible (or the improvement is negligible). It should be noted that, once again this is typical of other one-point methods.

Secondly, Figure 3 shows an interesting tendency: the runs with the slower cost drop tend to converge with better final result. The opportunity of producing better results by increasing the convergence time has been studied in the literature with respect to other one-point search methods applied to different problems, e.g. (Johnson et al 1989), (Burke et al 2004). Drawing upon this work, we can see that an increase in the length of the fitness array in LAHC could help to achieve better results.

Thirdly, investigation of the time-cost diagrams in Figure 3 can also suggest suitable stopping conditions for LAHC (see the variants in Section 3.2). In the first variant, we stop the search when reaching some target solution. Usually, the algorithm has to reach such a solution in the shortest time. This stopping condition is common for problems where the target solution is known in advance. For example, when we have to satisfy just hard constraints, then the target could correspond to a solution with zero cost. However, our tests are run with problems of a different type: the major goal is to minimize the soft constraints (which cannot usually be satisfied completely) and the target cost is not so clear. Therefore, if we set arbitrary some target cost, then in order to minimize the search time we need to adjust L_{fa} to the given target value. In the example in Figure 3, we can see that if the target value is more than 5500, then HC (L_{fa} =1) is the fastest option. LAHC with larger L_{fa} will also achieve this value, but in a longer time. However, if the target is set to 4500, then HC will never reach that. The best (over four curves) option here is LAHC with L_{fa} =5000. Again, LACH with larger L_{fa} will be less effective. Finally, if the target is 4000, the only option is L_{fa} =30000.

The second variant suggests stopping the search after some given time (or a given number of iterations). This stopping condition is usual for those search techniques, which do not show a clear convergence, but which continue to steadily improve over time. For example, GAs often terminate after a given number of generations. In contrast, if we have an algorithm with a clear convergence criterion, then a decision to terminate it at a point in time could well prove to be ineffective. In this case, such a stopping condition should be applied with particular care in order to spend the available time in an effective way as possible. Let us return to the example in Figure 3. If we decide to stop the search after 50 seconds, then the best result will be achieved by the search with L_{fa} =5000. Greedy HC converges earlier, but with a worse result; LAHC with larger L_{fa} does not yet reach convergence and its intermediate result is also worse. The same reasoning can reveal that if the search lasts 100 seconds, the best value of L_{fa} is 15000 and for a search of 150 seconds the best L_{fa} =30000.

This analysis of two variants of the stopping condition demonstrates that in both cases, the most effective approach is to terminate the search at the moment of convergence. If we terminate it before the convergence then we do not employ the full power of the method. If we let the search run after the convergence, then we just waste computing time (this time could be spent more effectively with a larger L_{fa}). Hence, we suggest a third variant of the stopping condition for LAHC: to terminate it exactly at the point of convergence. In all of our experiments below (with both models and all studied heuristics) we use this stopping condition (this also helps to maintain the compatibility of the results). This stopping condition is quite common to one-point search methods and has been widely studied in the literature. In general practice, the convergence is detected when the current solution is not improved for a significantly long time. This time can be either fixed or it can be calculated as a percentage of the total search time. In this study, we employ the second variant: our search procedure is terminated after passing 2% of consequent non-improving (idle) iterations over the total number of iterations. For example, if the search has produced (overall) 10^6 iterations from the beginning and during the last 20000 iterations, there was no improvement, then the search is stopped. It should be noted, that a detailed examination of the curves in Figure 3 reveals that the maximum effectiveness can be achieved when the search is stopped exactly at the point of the intersection of the time-cost curve and the envelope of all these curves. However, the form of the envelope is problem dependent and we currently have no reliable method to detect these points during the search. Therefore, the points of convergence are considered to be an acceptable choice.

As noted, the above basic properties of LAHC are very similar to other one-point search heuristics. To demonstrate this, we have produced similar time-cost diagrams for Simulated Annealing (also applied to the *Exam_1* instance) with a geometric cooling schedule while varying initial temperature T_i and cooling factor β. Four examples of these curves are shown in Figure 4. For comparability, the scale of this plot is the same as in Figure 3.

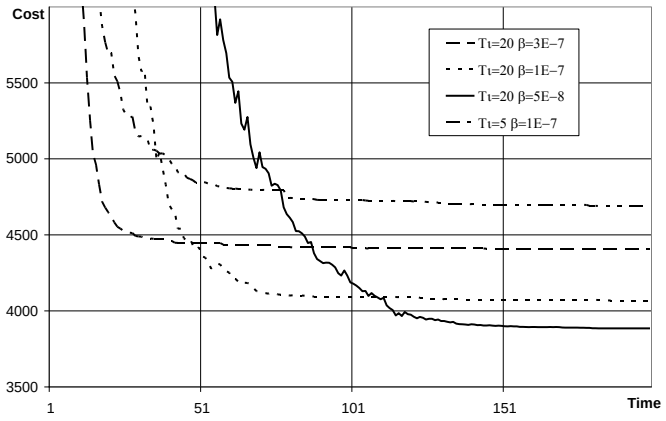


Figure 4: The time-cost diagrams of SA for the *Exam_1* instance with different T_i and β

In this figure, the first three curves (plotted for runs with T_i =20 and different β) are quite similar to the ones produced by LAHC and presented in Figure 3. Correspondingly, all previous reasoning (including the termination condition) is applicable to this case. The variation of the cooling factor β affects just the convergence time (this is analogously to L_{fa} in LAHC), whilst the balance between the search time and the quality of the results is always kept. As was the case in Figure 3, the points of convergence in these diagrams can be thought of as composing an analog of the Pareto front in the time-cost coordinates. These points are non-dominated, which means that any value of β in SA or L_{fa} in LAHC can be considered to be the best value depending on particular circumstances.

However, at this stage the similarity ends. If the initial temperature is set up to an incorrect value (for example T_i =5), then SA loses its power without any counterbalancing profit. In Figure 4, any point on the fourth curve (with T_i =5) is dominated by some point on the diagrams produced with T_i =20. It is clearly the case that the initial temperature plays a completely different role in the performance of the SA search than the cooling factor. The cooling factor can be viewed as a "time affecting" parameter, which should be tuned only if the situation requires it. On the other hand, the initial temperature can be thought of as a "power affecting" parameter, which must always be tuned because SA cannot achieve high quality performance without an appropriate value. This could be seen as a major reason for the extensive and wide ranging study of SA parameterization in the literature.

In contrast, the performance of LAHC search procedure cannot be damaged by the incorrect setting of parameters. This is because the "power affecting" parameters are simply absent in LAHC. I.e. when we have no tool for adjusting an algorithm's performance, we can assume that this performance is always optimal (for this algorithm). This property of LAHC distinguishes it from search techniques such as SA (or TA). This property is quite important for practical applications where a search is run in a fully-automated mode and we have no chance to verify manually the appropriateness of the parameterization. It should be noted that GDA is also free from "power-affecting" parameters (similar to LAHC). However, our experiments below have revealed that GDA is clearly outperformed by LAHC. Although L_{fa} and β both affect the convergence time, their internal mechanisms for time regulation are completely different. Therefore, the next part of our study contains an investigation of the correlation between L_{fa} and the search time.

In the second series of experiments, we employ the "convergence" stopping condition within LAHC and run this algorithm on all benchmark problems with three values for L_{fa}: 1, 5000 and 50000. With each value, our algorithm was run 20 times. Table 1 shows the average results and the average run times for 12 exam timetabling benchmark datasets. The same results for 7 TSP datasets are presented in Table 2.

Table 1: Results for exam timetabling instances produced by LAHC with different L_{fa}

	L_{fa} = 1 (HC)		L_{fa} = 5000		L_{fa} = 50000	
Dataset	Cost	Time (sec)	Cost	Time (sec)	Cost	Time (sec)
Exam_1	5737	5.0	4368	41	3818	412
Exam_2	571	1.4	459	15	395	149
Exam_3	11720	8.4	8359	103	7479	1065
Exam_4	18620	2.3	14043	26	12652	269
Exam_5	4000	1.5	2748	36	2527	323
Exam_6	26963	0.74	25818	9.1	25460	101
Exam_7	5303	5.9	3988	147	3592	1367
Exam_8	9147	2.9	7331	41	6736	434
Exam_9	1315	0.31	1073	3.3	988	39
Exam_10	13786	0.43	13161	4.5	13024	41
Exam_11	35445	9.9	25775	146	23299	1333
Exam_12	5677	0.19	5313	1.07	5244	11.2

Table 2: Results for TSP instances produced by LAHC with different L_{fa}

Dataset	L_{fa} = 1 (HC)		L_{fa} = 5000		L_{fa} = 50000	
	Cost	Time (sec)	Cost	Time (sec)	Cost	Time (sec)
Rat783	11082	0.10	9226	8.9	8991	95
U1060	263196	0.42	232840	14	227987	134
Fl1400	22656	0.90	20548	30	20316	282
U1817	68709	2.2	61253	43	58920	386
D2103	97283	3.4	88339	52	85210	473
Pcb3038	159299	8.2	148648	95	142346	929
Fl3795	33067	18	30692	155	29892	1331

These tables demonstrate that the proposed properties of LAHC hold for all of the tested benchmark instances: the longer the fitness array, the longer the search (convergence) time and, consequently, the better the final results. Moreover, it could be observed that in these tables, the average processing time of runs with L_{fa}=50000 is approximately 10 times longer than that with L_{fa}=5000. Based on this, we propose a hypothesis that the convergence time is linearly dependent on the length of the fitness array.

To provide more evidence to support this proposed hypothesis, a more precise examination was carried out in our third series of experiments. We conducted a high number of runs (around 500) while randomly varying L_{fa} in the interval from 1 to 50000. The produced results were visualized in the form of diagrams. As in the first series, the experiments were run with all benchmark datasets and, in all cases, the algorithm showed a similar behavior. Therefore, we present here the examples for the *Exam_1* dataset only. Figure 5 illustrates the dependence of the final cost on the L_{fa}. Here, each point represents the result of a single run, while its coordinates show the corresponding value of L_{fa} and the final cost.

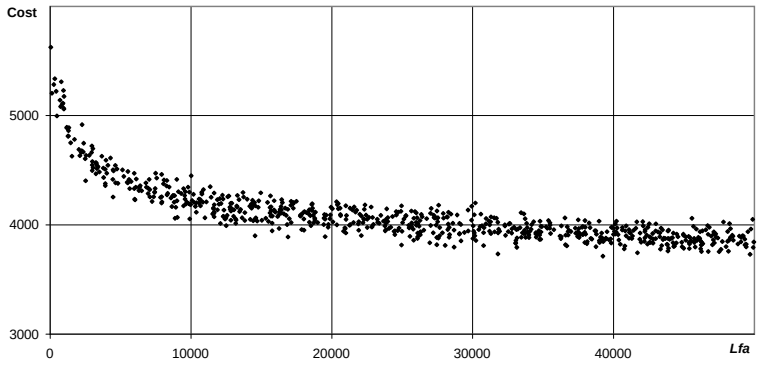


Figure 5: Dependence of the final cost on L_{fa} for the *Exam_1* instance

This diagram confirms the proposition regarding the dependence of the final cost on L_{fa} in this case. Even though the results are scattered (which is typical for search algorithms), this dependence is clear and distinct. Correspondingly, the observation expressed in (Johnson et al 1989) for SA that "up to a certain point, it seems to be better to perform one long run than to take the best of a time-equivalent

collection of shorter runs" looks to be also appropriate for LAHC. For example, in Figure 5, any result with L_{fa}=50000 is better (in spite of the scatter) than any result with L_{fa}=5000.

The diagram in Figure 6 represents the dependence of the convergence time on the L_{fa}. Here each run is depicted as a point with coordinates corresponding to L_{fa} and the convergence time.

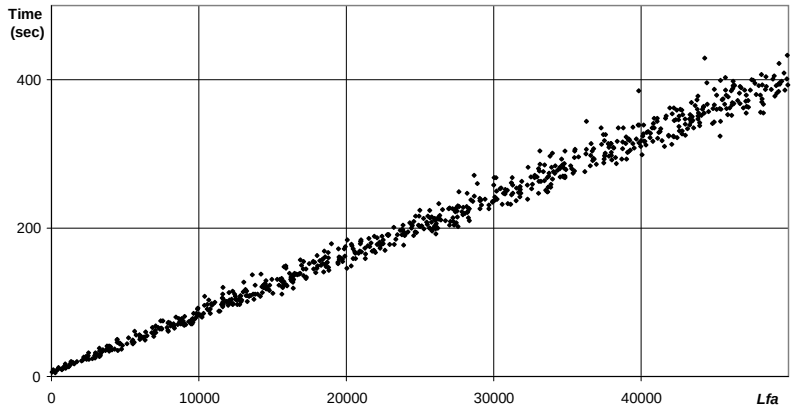


Figure 6: Dependence of run time on L_{fa} for *Exam_1* instance

For the majority of points, this diagram appears as a straight line, which provides further evidence to support our hypothesis regarding the linear dependence between L_{fa} and the convergence time. This property can be employed in practice, for example, knowing the processing time of a short run, it is possible to calculate an angle coefficient $Time/L_{fa}$ and then use it to set up the necessary length of the fitness array for a long run within a limited time interval.

The examination of run times presented in Tables 1 and 2 could reveal that the above mentioned angle coefficient $Time/L_{fa}$ is different for each particular instance. Factually, it appears as a unique constant of each dataset, which can be thought of as an amount of time (or a number of iterations) passed within the unit of the fitness array. Its particular value might somehow reflect the properties of a problem and a search algorithm. To investigate this hypothesis, we present Table 3, which shows the dimensions (in terms of exams, timeslots and rooms) of our exam timetabling instances and the corresponding angle coefficients. It should be noted, that although the processing time of a single iteration is not dependent on L_{fa} , there is a strong differentiation between different instances, implementations and hardware. To avoid these issues, in the rest of the paper, we express the computing time as the total number of iterations. However, the actual computing time can be easily evaluated: the number of iterations per second, which our algorithm produced in the presented experiments is shown in the last column of Table 3. This table also contains the angle coefficients in the form of ratios of the total number of iterations N_{it} to L_{fa}.

Table 3: Dimensions and angle coefficients of exam timetabling instances

Dataset	Exams	Timeslots	Rooms	$Time/L_{fa}$	N_{it}/L_{fa}	N_{it}/sec
Exam1	607	54	7	0.0082	2918	360000
Exam2	870	40	49	0.003	1371	460000
Exam3	934	36	48	0.021	4188	200000
Exam4	273	21	1	0.0054	321	60000
Exam5	1018	42	3	0.0065	3515	540000
Exam6	242	16	8	0.002	915	460000
Exam7	1096	80	15	0.027	11652	430000
Exam8	598	80	8	0.0087	5066	580000
Exam9	169	25	3	0.00078	765	980000
Exam10	214	32	48	0.00082	665	810000
Exam11	934	26	40	0.027	3622	134000
Exam12	78	12	50	0.00022	103	470000

This table demonstrates a certain dependence between the angle coefficient and the dimensions of a problem: generally the larger problems impose the higher coefficients. However, even for similar sized problems the angle coefficients can be very different. The problem's set of hard and soft constraints may also influence its value. Obviously, even a smaller, but more highly constrained problem can be

more difficult to solve. Here, it would be possible to express a hypothesis that the angle coefficient might somehow serve as a measure of the "hardness" of a problem. To check this proposition, we present Table 4 where we examine (in the same way) our TSP instances, which are free from constraints and their hardness is mostly dependent on their size (number of cities).

Table 4: Sizes and angle coefficients of TSP benchmark instances

Dataset	Size	$Time/L_{fa}$	N_{it}/L_{fa}	N_{it}/sec	$N_{it}/L_{fa}/Size^{1.2}$
Rat783	783	0.0019	10984	5800000	3.70
U1060	1060	0.0027	13117	4900000	3.07
Fl1400	1400	0.0056	22047	3900000	3.70
U1817	1817	0.0077	26616	3500000	3.27
D2103	2103	0.0095	30880	3300000	3.18
Peb3038	3038	0.019	56410	3000000	3.73
Fl3795	3795	0.027	74654	2800000	3.78

In contrast to Table 3, this table demonstrates a quite clear dependence between the angle coefficients and the size of the instances. We have proposed that this dependence might be expressed in the form of some non-linear function. Over several possible forms of such a function, we have selected a variant where the angle coefficient is proportional to the size of the problem to a degree of 1.2. The last column in Table 4 demonstrates that the coefficients of proportionality of this function are quite close to each other for all our TSP instances. Hopefully, in the future this proposition will be either proven or a more correct function will be discovered. Hypothetically, the existence of such a function could help to adjust L_{fa} automatically to a TSP instance of any size. We believe that the investigation of the LAHC angle coefficients with respect to different types of problems, the influence of constraints, and other related issues represents a very interesting subject of future research.

4. A Comparison of LAHC with other techniques.

In the previous section we have shown that the results produced by LAHC can be improved with a relatively long fitness array. However, we need to determine how well it performs compared to existing one-point search methods such as Simulated Annealing, Threshold Accepting and the Great Deluge algorithm? We carry out this comparison in our fourth series of experiments, where we use the same software whilst varying just the acceptance conditions.

However, a fair comparison of several one-point search methods is not straightforward because of three issues: firstly, the initial temperatures in SA (and thresholds in TA) should be properly tuned; secondly, the quality of results of all competing techniques is dependent on their processing time and thirdly, these results are quite scattered.

To address the first issue, we followed general suggestions from the literature: The initial temperature of SA was selected so that, at the starting point, the algorithm accepted 85% of non-improving moves. The same method was used for selecting the initial threshold in TA. In GDA, the initial level was equal to the initial cost function. In addition, in SA and TA, we applied the geometric cooling schedule whilst, in GDA, we used the linear lowering of the level. Note that the use of the common settings for the cooling schedules (without adjustment for particular instances) was undertaken in order to underpin the fairness of the comparison. As LAHC and GDA do not require a tuning of the "power affecting" parameters, spending extra time on tuning their competitors could be seen as a certain bias in favor of them.

The second and third issues are more complex. The performance of an algorithm appears as a scattered curve (see Figure 2) and we have to compare curves rather than scalar numbers. It should also be taken into account that the convergence time (the total number of iterations) is pre-defined differently (and rather approximately) in different methods (see Burke et al 2004). Therefore, we propose a methodology for the comparison of search techniques that we believe is quite fair. In a similar manner to that carried out in the experiments in the previous section, each competitor algorithm was run a high number of times (around 500). The time-affecting algorithmic parameters (cooling factors in SA, TA and GDA and L_{fa} in LAHC) were randomly varied so, that the resulting total number of iterations was distributed in the interval of $1...240(*10^6)$. In this way, we produced a plot similar to that presented in Figure 5. Then the horizontal axis (which represents a number of iterations) was divided into 12 equal segments and, for each segment, we calculated an average cost among all the results belonged to that segment. Thus, the function is represented as a series of cut-offs by segments of equal size. The resulting cut-offs produced by four competitor techniques for the *Exam_1* dataset are presented in Table 5. The best result for each cut-off over four competitors is highlighted in bold.

Table 5: The comparison of cut-offs for SA, TA, GDA and LAHC for the *Exam_1* dataset

N of iterations (*10^6)	SA	TA	GDA	LAHC
1-20	4907	**4723**	5160	4874
20-40	4357	4280	4579	**4276**
40-60	4216	4169	4447	**4136**
60-80	4125	4093	4330	**4059**
80-100	4088	4025	4299	**4020**
100-120	4028	3991	4213	**3966**
120-140	3992	**3918**	4198	3930
140-160	3956	3907	4154	**3888**
160-180	3926	3896	4129	**3882**
180-200	3912	3854	4098	**3848**
200-220	3912	**3839**	4073	3846
220-240	3877	3840	4053	**3807**

Surprisingly, a distinctly worse performance is shown by GDA. The three remaining competitor techniques perform almost in the same way. This tendency was observed in our experiments with all other benchmark instances. In this paper, we present an example of the cut-offs for a single interval. Table 6 contains the cut-offs in the interval of 80-100 (*10^6) iterations for 11 exam timetabling datasets. The cut-off results for our TSP instances in the interval of 380-400 (*10^6) iterations are shown in Table 7. The difference in the scale of the intervals is caused by the difference in the computational speed of our algorithms (see Tables 3 and 4).

Table 6: The comparison of cut-offs in the interval of 80-100 (*10^6) iterations for SA, TA, GDA and LAHC for the Exam Timetabling datasets.

Dataset	SA	TA	GDA	LAHC
Exam_2	**393**	**393**	413	397
Exam_3	7771	7718	7827	**7645**
Exam_4	12038	**11854**	12222	12037
Exam_5	2566	**2517**	2831	2592
Exam_6	**25380**	25430	25533	25411
Exam_7	3916	**3854**	4103	3914
Exam_8	7068	7029	7525	**7000**
Exam_9	961	**953**	994	964
Exam_10	**12993**	13011	13134	**12993**
Exam_11	23935	**23585**	24761	23682
Exam_12	**5154**	5190	5231	5200

Table 7: The comparison of cut-offs in the interval of 380-400 (*10^6) iterations for SA, TA, GDA and LAHC for TSP datasets.

Dataset	SA	TA	GDA	LAHC
Rat783	9017	9032	9285	**9003**
U1060	228679	**228255**	236406	228697
Fl1400	20425	**20372**	20776	20429
U1817	60260	**59831**	64116	60488
D2103	87510	87488	92470	**87221**
Pcb3038	147221	**146590**	156249	147558
Fl3795	**30865**	31041	32239	31021

When analyzing Tables 5, 6 and 7 it could be concluded that for three algorithms (SA, TA and LAHC) the results hardly represent strong evidence of the superiority of either of the techniques. It would be possible to be given the impression that only the allowing the worsening solutions provides an increase in the effectiveness of the search procedure and it does not matter in which way we allow them. In this context, the following question arises: what is the reason for proposing a new heuristic if it is not better than the existing ones? Is there any advantage in the absence of the cooling schedule in LAHC?

To answer this question we propose a fifth experiment. Let us assume that the *Exam_1* problem is reformulated so that its cost function C is transformed by a cubical polynomial expressed by Formula 3 and now we have to minimize a new cost function C_r instead of C.

$$C_r = C^3 - 32400*C^2 + 350*10^6*C \quad (3)$$

It should be noted that the discriminant of the first derivative of this function is always negative therefore this function is monotonic. We suppose that the monotonic transformation of an objective function does not affect the major properties of a problem and preserves the dominance relations between solutions, i.e. for any pair of solutions s_1 and s_2, if $C(s_1) > C(s_2)$ then $C_r(s_1) > C_r(s_2)$. Consequently, all original global and local optima are also preserved within this new formulation. Factually, such a transformation appears as just a nonlinear rescaling of the cost function. To reveal how our competitor algorithms perform with the rescaled cost function, we have repeated the previous experiment. All experimental conditions remain untouched, only the initial temperatures and thresholds were re-tuned in the context of the new problem. The produced results are presented in Table 8. For consistency with previous experiments, the figures in this table represent the values of the original cost function (before the rescaling).

Table 8: The comparison of cut-offs for SA, TA, GDA and LAHC for *Exam_1* dataset with the rescaled cost function.

N iterations ($*10^6$)	SA	TA	GDA	LAHC
1-20	5365	5240	5579	**4805**
20-40	5064	4992	5142	**4217**
40-60	4898	4919	4946	**4117**
60-80	4867	4882	4845	**4044**
80-100	4838	4788	4826	**4012**
100-120	4814	4834	4763	**3991**
120-140	4796	4789	4657	**3934**
140-160	4755	4748	4627	**3852**
160-180	4720	4719	4548	**3877**
180-200	4753	4729	4498	**3845**
200-220	4703	4721	4471	**3830**
220-240	4695	4687	4466	**3812**

We see in the table that the rescaling of the cost function has a dramatic impact on the performance of three competitor techniques: SA, TA and GDA have completely failed with this new problem. Their performance has deteriorated significantly. Indeed, they are not far from the greedy HC results (see Figure 2) while taking much more computing time. In contrast, the rescaling has no effect on the performance of LAHC. It has produced results of the same quality as they were without rescaling (small deviations are present because of a stochastic scatter).

5. Discussion

The above phenomenon can be discussed from different points of view. At first glance, the following naïve explanation can be used. For example, in TA, the acceptance of a worse solution is dependent on the particular value of $\Delta C = C^* - C$ (see the notations in Section 1), which is *numerically* compared with the threshold. Obviously, the value of ΔC is scale-dependent, i.e. it can be different in the original and the rescaled problems. This represents highly possible situations where, for the same current and candidate solutions, the original ΔC is lower but the rescaled ΔC is higher than the threshold. Correspondingly, this candidate will be accepted in the first case and rejected in the second one. As a result, the entire search process will be generally reshaped.

In contrast, LAHC does not employ the numerical comparisons, but operates with just a *ranking* of different values. In other words, the acceptance of a worse candidate is dependent only on a dominance relation (i.e. "less", "equal" or "more") between the candidate cost C_i^* and the corresponding element of the fitness array $f_{(i \bmod Lfa)}$ (see Formula 2). The candidate is accepted in all cases when the value $\Delta = C_i^* - f_{(i \bmod Lfa)}$ is non-positive, even though this value is different in the original and the rescaled problems. In LAHC, the value of Δ is not numerically compared with any other one: only its sign is taken into account during the acceptance procedure and this sign is scale-independent. Correspondingly, if the candidate is accepted in the original problem, then it will be also accepted in the rescaled one. Thus, having the same initial randomization, the original search process can be completely repeated within any rescaling as long as the original dominance relations are preserved.

The above naïve reasoning explains why the rescaling affects the performance of the competing methods. But why have they deteriorated so badly? This might be because the cooling schedule and the cost function are two highly interconnected entities. We can express a hypothesis that the application of an original cooling schedule to the rescaled cost function is equivalent to the application of an inverse rescaled cooling schedule to the original cost function. Of course, the reverse rescaled cooling schedule has a bizarre shape and cannot provide a good performance. To get a normal performance with the

rescaled cost, we should correspondingly rescale the cooling schedule. However, if the form of the rescaling is not known in advance, it is hardly possible to find this function empirically. From this point of view, we can think of LAHC as having a self-generated cooling schedule, i.e. the pattern of L_{fa} previous iterations serves as a cooling schedule for L_{fa} next iterations. This pattern is automatically adjusted to the properties of a particular region of the search space: when the cost tends to rapidly fall down, then the gradient of this "cooling schedule" is also increased and vice versa. It becomes more flat in those regions when an average reduction in cost is quite small.

It is also possible to observe certain fundamental principles behind the robustness of LAHC. A naive level of reasoning (similar to the above) can clarify that, in addition to LAHC, the rescaling of the cost function also does not affect a range of other techniques, such as HC, TS, and some variants of GA. The distinguishing property of such methods is that they are also based on the ranking of solutions (rather than numerical comparisons like in SA, TA or GD). There is sometimes an assumption among the research community that the "ranking-based" methods are relatively more natural and hence, more reliable (for example, have wider applicability, or are not sensitive to scales) than those employing numerical data. From this point of view, LAHC proposed here (even though it has many similarities with cooling-schedule based one-point searches) holds the properties and shows the behavior typical to algorithms from the "ranking-based" class. Thus, it could be supposed that LAHC is more reliable than SA, TA and GD because it represents a conceptually different methodology, which belongs to the class of relatively more natural and robust algorithms.

One can argue that the problem investigated here with the rescaled cost function is artificially created rather than taken from the real-world. This is true, of course. However, at first, this example is quite efficient for testing search algorithms: here the level of robustness of different methods is highly evident. Secondly, having a single precedent (even with the artificial problem) where a technique fails enables us to observe the limited application area of this technique. Thirdly, similar situations (maybe less distinctive) are quite possible in the real-world. For example, they might be caused by some specific constraints. If supposing that the rescaling somehow rearranges the density of solutions over different regions of the search space, then the exclusion of certain solutions (forced by the hard constraints) from the search space could cause the same effect. In addition, many highly non-linear problems in different application areas (such as bioinformatics or automated design) are well-known across the computational search literature. Thus, all the above reasoning suggests that LAHC might be especially effective with highly constrained and non-linear problems.

6. The success of LAHC in the International Optimisation Competition

The high level of practical effectiveness of LAHC has also been confirmed in the International Optimisation Competition (IOC), where a LAHC-based algorithm won the 1st place prize. The IOC was organized by SolveIT Software Pty Ltd in October-November 2011. The purposes of the competition were announced as follows: to promote modern heuristic optimization methods among research communities and to identify world-class talent in the area of optimization science. The competitor's goal was to develop a Java command-line application, which is able to solve the largest constrained Magic Square (MS) problem within one minute of the run time. The constraint in MS was represented as a pre-defined sub-matrix placed into a given position. It should be noted that there was just one month from the announcement of the competition to its deadline. Thus, the development time was very tight and there was not enough time for careful study of the properties of the problem. In fact, the entry algorithm had to manage a new and unstudied problem. More information about the IOC can be found on the official competition web site at: http://www.solveitsoftware.com/competition.

The competition results were announced on 19 December 2011. The second runner-up algorithm had solved the MS problem of size 400x400 within one minute. The first runner-up had solved the 1000x1000 MS problem. The winning LAHC-based algorithm was able to solve the 2600x2600 constrained MS problem within one minute. It should be noted that all top-3 entry algorithms employ different variants of the decomposition of the entire MS problem into a series of smaller sub-problems, which are solved separately. For example, in the winner's algorithm MS is firstly decomposed into a series of concentric frames and then LAHC is applied several times to each of the frames.

Taking into account a very short development time and the necessity of working in a fully-automated mode, the choice of optimization heuristic was based on three main criteria: the simplicity of the development, the simplicity of the parameterization and the reliability of the heuristic. Here it should be noted that many sophisticated and intensively studied techniques in the literature would not

satisfy the above criteria: both runners-up algorithms were based on just the greedy HC method. In this environment, LAHC has a definite advantage: it is as simple and reliable as HC, but it is much more powerful.

In addition to LAHC, during pre-competition testing of the winning algorithm there was an attempt to employ Simulated Annealing as a kernel optimization method. However, this attempt was unsuccessful. It was found that the optimal cooling schedule parameters are highly varied for MS problems of different sizes. As the entry algorithm should be able to solve MS of any size, the real-world application of SA technique de-facto requires an additional algorithm for its automatic parameterization, which requires extra development time and extensive investigation of the properties of the problem. Of course, this was impossible to complete in the tight competition time. However, with less sophisticated parameterization (manual or random) SA has shown quite a poor performance: with the 1000x1000 MS problem in 5% of runs, the algorithm failed to produce MS in 1 minute. Furthermore, with the 2600x2600 MS problem, up to 62 % of runs (depending on the constraint position) were unsuccessful.

In contrast, the application of LAHC to the unstudied MS problem was successful. With constant L_{fa}=20000, LAHC has shown 100% reliability on all runs with either 200x200, 1000x1000 and 2600x2600 MS problems. It should be noted that the described LAHC-based algorithm has the potential to solve MS problems of much larger size than the 2600x2600 limit within 1 minute. In reality, this limit was caused by hardware restrictions rather than the algorithm's performance.

All the above observations can be experimentally confirmed using the original entry competition algorithm. To check everything (including the implementation details) by himself/herself the interested reader can download the Java source code from: http://www.cs.nott.ac.uk/~yxb/IOC/. This source code was slightly modified in order to facilitate easy switching between different search heuristics (HC, SA, TA, GD and LAHC) by changing just one parameter. Now it is possible for anyone to carefully test the performance of all these heuristics with the MS problem to verify the fact that LAHC represents the best choice among them.

7. Conclusions and future work

Over the years, a wide range of different search heuristics have been proposed and applied for different problems. In this situation, it is quite difficult to discover a really new and workable idea. The proposed LAHC has a range of unique properties, which can be outlined as follows:

- It is almost as simple as the greedy HC, but much more powerful.
- It could be considered as a variant of Adaptive Memory Programming as it intelligently uses the information collected during previous iterations.
- It is a one-point iterative search, but does not employ any variant of a cooling schedule and, therefore, might have a wider applicability than cooling-schedule based algorithms.
- It is free from "power affecting" parameters and it is well-known that the inappropriate setting of these parameters can deteriorate the performance of the search.
- It is dependent on a single input parameter, which regulates the convergence time. The presented experiments have revealed that the convergence time is approximately proportional to this parameter. Moreover, the coefficient of proportionality is constant for each particular instance and seems to reflect the difficulty of the problem.
- It is almost not sensitive to initialization.
- It does not employ the properties of a particular type of problem and, therefore, could be positioned as a general-purpose metaheuristic.

To confirm the generality of LAHC we presented its experimental evaluation with the Travelling Salesman and Exam Timetabling problems. We expect that it can be applied to any optimization problem where other one-point search methods are applicable. In addition to our own tests with LAHC, a number of researchers from different institutions have started separate studies on LAHC, e.g. (Verstichel and Vanden Berghe 2009), (Abuhamdah 2010) by drawing on our initial presentation at PATAT conference (Burke and Bykov 2008). Furthermore, in December 2011 the LAHC-based algorithm has won the 1st place prize in the International Optimisation Competition.

This paper represents the introduction of LAHC to the literature. Of course, it requires further intensive investigation. It may be possible to propose different improvements and variations of this algorithm. For example, the compared value can be taken randomly from the fitness array or it can be calculated as an average value over all its elements. The fitness array can be of variable length or the values of its elements can be at some point reassigned (in analogue to "reheating" in cooling-schedule based methods). Also, LAHC might be useful in different hybrid approaches where the greedy HC is applicable, for example, in Memetic Algorithms. Although, in this paper, we have applied the idea of the late acceptance to Hill-Climbing, this idea can be embedded in any method where a candidate cost is compared with the current one. For example, we can imagine a Late Acceptance Simulated Annealing. In addition, we hope to develop a multiobjective version of the Late Acceptance algorithm.

Finally, we have proposed, applied and discussed a methodology for testing the robustness of optimization heuristics using the non-linear monotonic rescaling of a cost function. We believe that it represents quite a promising subject of future research. For example, it may be beneficial to investigate the performance of other search algorithms (such as ACO, PSO and hybrid methods) with the same rescaled problem.

Acknowledgements

The work described in this paper was carried out under grant (GR/S70197/01) awarded by the UK Engineering and Physical Sciences Research Council (EPSRC).

We would like to thank Ender Ozcan, Peter Demeester and John Woodward for helpful advice in addition to other colleagues who expressed their opinions about our method. We would also like to thank the anonymous referees for their valuable advice.

References

Abuhamdah, A. 2010. Experimental result of late acceptance randomized descent algorithm for solving course timetabling problems. *IJCSNS International J. of Compu. Sci. and Network Security* **10** 192-200.

Anderson, K., R. V. V. Vidal, V. B. Iverson. 1993. Design of teleprocessing communication network using simulated annealing. *Springer Lecture Notes in Econom. and Math. Systems* **396** 201-216.

Appleby, J. S., D. V. Blake, E. A. Newman. 1960. Techniques for producing school timetables on a computer and their application to other scheduling problems. *The Comput. J.* **3** 237-245.

Applegate, D. L., R. E. Bixby, V. Chvátal, W. J. Cook. 2006. *The Traveling Salesman Problem: A Computational Study*, Princeton University Press.

Burke, E. K., D. Elliman, P. Ford, R. Weare. 1996. Examination timetabling in British universities: a survey. *Springer Lecture Notes in Comput. Sci.* **1153** 76-90.

Burke, E. K., J. Newall. 2003. Enhancing timetable solutions with local search methods. *Springer Lecture Notes in Comput. Sci.* **2740** 344-354.

Burke, E. K., Y. Bykov, J. Newall, S. Petrovic. 2004. A time-predefined local search approach to exam timetabling problems. *IIE Trans.* **36**(6) 509-528.

Burke, E.K., Y.Bykov. 2008. A late acceptance strategy in hill-climbing for exam timetabling problems (extended abstract). In *Proceedings of the 7th International Conf. on the Practice and Theory of Automated Timetabling (PATAT 2008)*. Montreal, Canada, August 2008.

Carter, M.W., G. Laporte. 1996. Recent developments in practical examination timetabling. *Springer Lecture Notes in Comput. Sci.* **1153** 3-21.

Cohn, H., M. Fielding. 1999. Simulated annealing: searching for an optimal temperature schedule. *SIAM J. on Optim.* **9**(3) 779-802.

Dueck, G. 1993. New optimization heuristics. The great deluge algorithm and the record-to-record travel. *J. of Computational Phys.* **104** 86-92.

Dueck, G., T. Scheurer. 1990. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *J. of Computational Phys.* **90** 161-175.

Glover, F. 1986. Future paths for integer programming and links to artificial intelligence. *Comput. and Oper. Res.* **5** 533-549.

Hu, T.C., A. B. Kahng, C.-W. A. Tsao. 1995. Old bachelor acceptance: a new class of non-monotone threshold accepting methods. *ORSA J. on Computing* **7**(4) 417-425.

Johnson D. S., C. R. Aragon, L. A. McGeoch, C. Schevon. 1989. Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. *Oper. Res.* **37**(3) 865-892.

Johnson D. S., C. R. Aragon, L. A. McGeoch, C. Schevon. 1991. Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning. *Oper. Res.* **39**(3) 378-406.

Johnson, D. S., L. A. McGeoch. 1997. The Travelling Salesman Problem: a case study in local optimization. In E. H. L. Aarts and J. K. Lenstra (eds) *Local Search in Combinatorial Optimization*. J.Willey and Sons, London 215-310.

Kirkpatrick, S., J. D. C. Gellat, M. P. Vecci. 1983. Optimization by simulated annealing. *Sci.* **220** 671-680.

Laguna, M., F. Glover. 1996. What is tabu search? *Colorado Bus. Rev.* **61** 5-12.

Lewis, R. 2008. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum* **30**(1) 167-190.

Lin, S., B. W. Kernighan. 1973. An effective heuristic algorithm for the travelling-salesman problem. *Oper. Res.* **21**(2) 498-516.

McCollum, B., A. Shaerf, B. Paechter, P. J. McMullan, R. Lewis, A. J. Parkes, L. Di Gaspero, E. K. Burke, R. Qu. 2010. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS J. Computing* **22** 120-130.

McMullan, P. 2007. An extended implementation of the great deluge algorithm for course timetabling. *Springer Lecture Notes in Comput. Sci.* **4487** 538-545.

Ninio, M., J. J. Schneider. 2005. Weight annealing. *Physica* **349** 649-666.

Obit, J. H., D. Landa-Silva, D. Ouelhadj, M. Sevaux. 2009. Non-linear great deluge with learning mechanism for solving the course timetabling problem. In *Proceedings of MIC 2009: the VIII Metaheuristic International Conf.*, Hamburg, Germany, July 2009, id-1-10.

Osman, I. H. 1993. Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem, *Ann. Oper. Res.* **41** 421-451.

Qu, R., E. K. Burke, B. McCollum, L. T. G. Merlot, S. Y. Lee. 2009. A survey of search methodologies and automated system development for examination timetabling. *J. of Scheduling* **12** 55-89.

Ozcan, E., M. Birben, Y. Bykov, E. K. Burke. 2009. Examination timetabling using late acceptance hyper-heuristics. In *Proceedings of the 2009 IEEE Congress on Evolutionary Computation (CEC 2009)*, Trondheim, Norway, May 2009, 997-1004.

Schaerf, A. 1999. A survey of automated timetabling. *Artificial Intelligence Rev.* **13**(2) 187-127.

Taillard, E. D., L. M. Gambardella, M. Gendreau, J.-Y. Potvin. 2001. Adaptive memory programming: a unified view of metaheuristics. *Eur. J. Oper. Res.* **135** 1-16.

Thompson, J. M., K. A. Dowsland. 1996. General cooling schedules for simulated annealing based timetabling system. *Springer Lecture Notes in Comput. Sci.* **1153** 345-363.

Verstichel, J., G. Vanden Berghe. 2009. A late acceptance algorithm for the lock scheduling problem. *Logistic Management* **2009** (5) 457-478