European Journal of Operational Research 258 (2017) 70–78

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

The late acceptance Hill-Climbing heuristic

Edmund K. Burke, Yuri Bykov*

School of Electronic Engineering and Computer Science, Queen Mary University of London, Mile End Road, London, E1 4FZ, UK

ARTICLE INFO

Article history:
Received 16 September 2014
Accepted 8 July 2016
Available online 15 July 2016

Keywords:
Combinatorial optimization
Metaheuristics
Late acceptance hill climbing
Timetabling
Travelling salesman problem

ABSTRACT

This paper introduces a new and very simple search methodology called Late Acceptance Hill-Climbing (LAHC). It is a local search algorithm, which accepts non-improving moves when a candidate cost function is better than it was a number of iterations before. This number 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. Also, we present a fair comparison of LAHC with well-known search techniques, which employ a cooling schedule: Simulated Annealing (SA), Threshold Accepting (TA) and the Great Deluge Algorithm (GDA). In addition, we discuss the success of the method 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 most of the studied instances (especially for the large sized ones), its average performance is better than competitor methods. Moreover, the LAHC approach has an additional advantage (in contrast to the above cooling schedule based methods) in its scale independence. We present an example where the rescaling of a cost function in the Travelling Salesman Problem dramatically deteriorates the performance of three cooling schedule based techniques, but has absolutely no influence upon the performance of LAHC.

1. Introduction

A family of algorithms, often labeled under the term *local search*, represents a wide range of techniques, across a broad spectrum of problems. Generally, these algorithms have the following advantages: they are relatively simple in implementation, moderate in CPU time requirement and quite effective for large-scale problems. A typical local search procedure starts from a random initial solution, which is iteratively improved by accepting or rejecting candidate solutions. The simplest local search algorithm is the greedy Hill-Climbing (HC), which was one of the earliest search techniques (Appleby, Blake, & Newman, 1960). HC accepts only candidates with the same or better cost than the current one. This method usually produces relatively low quality results in a quick CPU time. The better results (in a longer CPU time) can be achieved using alternative techniques, which allow the limited acceptance of worsening moves. The common property of these methods is that their acceptance condition is regulated by a *control parameter* (such as *temperature*, *threshold* or *water level*), which is varied in the course of the search. The shape of this variation (schedule) has a significant impact on the overall performance of these techniques and this has been extensively investigated over the years.

One of the most well studied local search techniques is Simulated Annealing (SA) proposed by Kirkpatrick, Gelatt, and Vecchi (1983). It is a stochastic algorithm, which accepts worse candidates with probability $P=exp[(C-C^*)/T]$ (here and everywhere below we assume a minimization problem). In this expression, 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 is called 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 percent and 90 percent (Johnson, Aragon, McGeoch, & Schevon, 1989), 75 percent (Thomson & Dowsland, 1996) and 95 percent (Cohn & 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 ith 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 alternatives, such as the "quadratic cooling schedule" (Anderson, Vidal, & Iverson, 1993) or even increases in the temperature, such as adaptive cooling (Thompson & Dowsland, 1996) or reheating (Osman, 1993).

* Corresponding author.
E-mail addresses: e.burke@qmul.ac.uk (E.K. Burke), mail@yuribykov.com (Y. Bykov).

http://dx.doi.org/10.1016/j.ejor.2016.07.012
0377-2217/

In addition to Simulated Annealing, a number of other similar search techniques have been proposed. One that is particularly close to SA is the Threshold Accepting (TA) method, which is also known as "Deterministic Simulated Annealing" (Dueck & Scheurer, 1990). Here, the candidate solution is accepted if C^*- $C \leq T$ where T is a control parameter (called the "threshold"), which again is varied based on its schedule. Another deterministic variant (proposed by Dueck 1993) is the Great Deluge Algorithm (GDA). In contrast to TA, its control parameter B (called the "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 $C^* \leq B$. 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 & Newall, 2003), non-linear level lowering (Obit, Landa-Silva, Ouelhadj, & Sevaux, 2009), reheating-like mechanisms (McMullan 2007) or modified acceptance condition (Burke & Bykov, 2016). There are several other methods based on this pattern such as "Old Bachelor Acceptance" (Hu, Kahng, & Tsao, 1995) or "Weight Annealing" (Nino & Schneider, 2005).

In all of these algorithms, the variation of the control parameter is regulated by arbitrarily defined schedules: in SA it is called the "cooling schedule". In our study, we also generalize this term for TA and GDA because their schedules are undetermined.

The cooling schedule has its strong and weak points. A strong point is that it enables an explicit way of processing time management. This feature is important in many situations where increasing the search time can lead to better final results. However, in order to effectively use the long searches in practice, their processing time should be pre-defined (see Burke, Bykov, Newall, & Petrovic, 2004). A weak point of the cooling schedule is that its optimal form is problem/instance-dependent and generally indefinite. That is the reason for the existence of a number of empirical recommendations 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 evaluated cooling-schedule based methods.

In (Burke & Bykov, 2008), we proposed the initial idea of a non-arbitrary control parameter, the value of which is obtained from the previous history of the search. We have called this method the Late Acceptance strategy. This also builds upon work presented at the CEC'09 conference (Ozcan, Birben, Bykov, & Burke, 2009). By drawing on our initial presentation, a number of researchers from different institutions have taken up the idea of late acceptance and started separate studies on this method applied to different problems such as: lock scheduling (Verstichel & Vanden Berghe, 2009), liner shipping fleet repositioning (Tierney, 2013) and balancing two-sided assembly lines (Yuan, Zhang, & Shao, 2015). Some of the researchers went beyond the case studies and have proposed their own modifications of our approach, such as: Late Acceptance Randomized Descent algorithm (Abuhamdah 2010) or Multiobjective Late Acceptance algorithm (Vancroonenburg & Wauters, 2013). In addition, this method was hybridized with other techniques (Alzaqebah & Abdullah, 2014) and investigated in respect of the recently developed hyper-heuristic approach (Jackson, Özcan, & Drake, 2013).

Furthermore, in December 2011, a late acceptance based algorithm won the 1st place prize in the International Optimisation Competition. In July 2012, two research teams ranked 4th and 7th places in the ROADEF/EURO Challenge 2012 whilst employing the idea of late acceptance in their entry algorithms. In June 2014 a research group (called CODeS) from KU Leuven, Belgium won the 1st place in the VeRoLog 2014 International Competition (http://verolog.deis.unibo.it) using the Late Acceptance method. In addition, our method has been embedded into at least two real-world software systems: the Rasta Converter project hosted by GitHub Inc. (US) (https://github.com/ilmenit/RastaConverter) and OptaPlanner, an open source project by Red Hat (http://www.optaplanner.org).

In this paper, we adapt our algorithm, investigate its properties and compare its performance with related methods (SA, TA and GDA). In order to provide prompt comprehensive information about LAHC to other researchers, we made an earlier version of this paper available as an institutional technical report (Burke & Bykov, 2012). Subsequent communication with a range of readers (especially with those who decided to implement our technique) revealed improvements that were required in that report. Therefore, this paper represents a significantly altered version of that institutional report: the methodological description is rewritten, unnecessary reasoning is removed, the experimental layout is changed and all experiments are completely re-calculated.

The description of our technique is presented in the next section. In Section 3, we present an experimental study of the properties of the proposed method. Section 4 contains a comparison of the new algorithm with existing techniques. In Section 5 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 6.

2. Late Acceptance Hill Climbing

The initial idea of the *late acceptance* heuristic is rather simple: the control parameter in the acceptance condition is taken from the history of the search. This heuristic could be viewed as an extension of HC with just one difference: in greedy Hill Climbing a candidate solution is compared with the immediate current one, but in the Late Acceptance Hill Climbing (LAHC) a *candidate is compared with that solution, which was the current several iterations before*. Except for the new acceptance condition, the other details of LAHC are the same as in other local search methods (such as HC, SA, TA or GDA), i.e., the algorithm is started from a random initial solution and iteratively accepts or rejects candidates until a stopping condition occurs.

Basically, LAHC can employ its acceptance rule while maintaining a list of a fixed length L_h (history length) of previous values of the current cost function. The candidate cost is compared with the last element of the list and if better, then the candidate is accepted. After the acceptance procedure, the list is updated i.e., the new current cost is added to the beginning of the list and the last element is removed from the end of the list. Note that the added current cost is equal to the candidate cost in the case of accepting only, but in the case of rejecting it is equal to the previous value.

The length L_h appears as a single algorithmic (and user-specified) parameter for this technique. The actual initialization of the list can be done automatically, even there is no previous history at the first iteration. We see here two possible variants: we can either produce L_h always accepted moves and record the values of objective function or just assign all elements of the list to be equal to the initial cost. Preliminary tests did not show any visual difference in the performance between these variants, but we consider the second variant to be preferable as it saves CPU time. It should be noted that when the initial list contains all values which are much higher than the initial cost, the second variant just turns into the first one. However, we do not recommend the initialization of the list by values which are much less than the initial cost, as in that case LAHC turns into HC at the beginning.

```
Produce an initial solution s
Calculate initial cost function C(s)
Specify L_h
For all k∈{0...L_h-1} f_k:=C(s)
First iteration I:=0; I_idle:=0
Do until (I>100000) and (I_idle>I*0.02)
   Construct a candidate solution s*
   Calculate a candidate cost function C(s*)
   If C(s*)≥C(s)
      Then increment the idle iterations number I_idle:=I_idle+1
      Else reset the idle iterations number I_idle:=0
   Calculate the virtual beginning v:=I mod L_h
   If C(s*)<f_v or C(s*)≤C(s)
      Then accept the candidate s:=s*
      Else reject the candidate s:=s
   If C(s)<f_v
      Then update the fitness array f_v:=C(s)
   Increment the iteration number I:=I+1
```

Fig. 1. The pseudocode of the final version of Late Acceptance Hill Climbing.

In addition to the basic variant, the idea of the late acceptance can be implemented into practice in a wide ranging variety of ways. Indeed, the proposed LAHC can have a significant number of variations and extensions. Although many of them are still unexplored, our tests have revealed some modifications where we have noticed a certain improvement in the performance of LAHC. The results of these tests are presented in Section 4.2.

At first, we suggest the combination of the late acceptance rule with the greedy rule (as is done in SA, TA and GDA) i.e., always accept non-worsening moves. Thus, its final acceptance condition at the ith iteration can be expressed by Formula 1.

$$C_i^* < C_{i-Lh} \text{ or } C_i^* \leq C_{i-1} \tag{1}$$

In this formula, C_i^* is the candidate cost, C_{i-1} is the current cost and $C_{i\text{-}Lh}$ denotes the cost of the current solution L_h iterations before, which is equal to $f_{(i \bmod Lh)}$. Obviously, when L_h is equal to 1 or 0, LAHC is simply greedy HC. Hence, LAHC obtains its unique properties with L_h equal to 2 and higher.

Secondly, the method for history recording can be also enhanced. Here, we suggest the updating of the list with better values only and exclude any updating with worse values, i.e., C_i is assigned to f_v when $C_i < f_v$., otherwise f_v is not changed.

These modifications affect the LAHC behavior at the very final stage of the search: in a similar way to other local searches (such as SA or GDA) it "becomes" HC (ceases to accept worsening moves). This means that we can apply here the stopping condition widely used with HC (and the above techniques) where the search is terminated when no further improvement is possible. This is usually detected by counting non-improving (idle) iterations I_{idle}. Our algorithm stops when I_{idle} reaches 2 percent over the total number of iterations. To avoid earlier termination, the minimum length of the search is limited by 100000 iterations. See (Burke & Bykov, 2012) for a more detailed discussion of the possible LAHC stopping conditions.

Finally, we suggest the standard *fifo* optimization of the list maintenance mechanism. Here the elements of the list are unmovable and the list appears as a fitness array F_a of length L_h ($F_a = \{f_0, f_1, f_2, \ldots f_{Lh\text{-}1}\}$). Its virtual beginning v, at the ith iteration, is calculated as:

$$v = i \bmod L_h \tag{2}$$

where "mod" denotes the remainder of integer division. Now, the value of f_v is compared with the candidate cost and after accepting or rejecting, the new current cost value is assigned to f_v. The pseudocode of the final version of LAHC is shown in Fig. 1. All experiments in this paper were carried out with this variant.

Table 1
TSP benchmark instances.

Dataset	Size
Rat783	783
U1060	1060
Fl1400	1400
U1817	1817
D2103	2103
Peb3038	3038
Fl3795	3795

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 problem. Therefore, it can be thought of as a general purpose search method, i.e., a metaheuristic. We expect that it can be applied to any problem, where other local search methods (HC, SA, TA, GDA) are applicable. To show its generality we present an experimental study with two types of problem: 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 & McGeoch, 1997) or (Applegate, Bixby, Chvátal, & Cook, 2006). There is also a well-known collection of TSP datasets (TSPLIB) available at http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/. It contains 111 datasets among which we have chosen for our tests 7 instances with sizes from 783 to 3795 cities. Their names and sizes are given in Table 1.

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, Elliman, Ford, & Weare, 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 providing a sufficient interval between exams taken by a student or larger exams should be scheduled earlier, etc. Over the last 10-15 years, this problem has been intensively studied. More information about the wide range of algorithmic approaches to this problem can be found in the following survey papers: (Carter & Laporte, 1996), (Schaerf 1999), (Lewis 2008) and (Qu, Burke, McCollum, Merlot, & Lee, 2009). In this study we use the specification of the Exam Timetabling Problem given at the 2nd International Timetabling Competition (ITC2007). Its description is presented in (McCollum et al., 2010) and on the original competition web site: http://www.cs.qub.ac.uk/itc2007/. This site also contains 12 datasets, which we use in the experiments in this paper. Some characteristics of these datasets are shown in Table 2. It should be noted that both the TSP and the Exam Timetabling problem are minimization problems, so in all our experiments, the lower the result, the better.

Table 2
Exam timetabling benchmark instances.

Dataset	Exams	Timeslots	Rooms
Exam_1	607	54	7
Exam_2	870	40	49
Exam_3	934	36	48
Exam_4	273	21	1
Exam_5	1018	42	3
Exam_6	242	16	8
Exam_7	1096	80	15
Exam_8	598	80	8
Exam_9	169	25	3
Exam_10	214	32	48
Exam_11	934	26	40
Exam_12	78	12	50

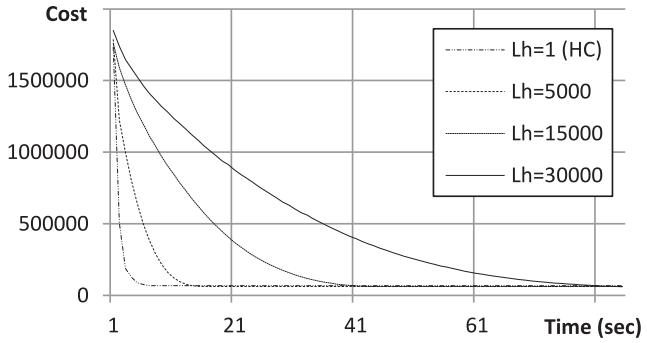

Fig. 2. The cost drop diagrams of LAHC for *U1817* instance with different L_h.

3.2. Experimental software

The proposed technique was evaluated using our software developed in Delphi 2007 and run on a PC Intel Core i7-3820 3.6 gigahertz, 32 GB RAM under OS Windows 7 64bit. Two dedicated models were created: the first for TSP and the second for the Exam Timetabling Problem.

Our TSP software is quite simple: the initial tour (solution) is randomly generated. At each iteration, a current solution is modified by the "double-bridge" move studied by Lin and Kernighan (1973). Here a current tour is randomly divided into two parts, which are reconnected in a reverse order. We have also made a special effort to prepare our TSP software for examination. For this purpose, the initial Delphi source code was literally translated into Java. Now, both versions are available for download at http://www.yuribykov.com/LAHC/. Using any of these programming languages, the interested reader can obtain all necessary information about our implementation and can also repeat our experiments. In addition, our LAHC algorithm for TSP is included in the "Multiheuristic Solver" software system available at: http://www.yuribykov.com/MHsolver/. Using this solver, the performance of LAHC can be easily compared with 5 other metaheuristics without the examination of the source code.

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. The feasible initial solution is generated by the "Saturation Degree" Graph Coloring heuristic, which was 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, 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, Aragon, McGeoch, & Schevon, 1991).
- The swapping of two randomly chosen exams. Again, the new rooms are chosen randomly here. This move employs the Kempe Chains procedure in a way that is similar to the previous one.
- The swapping of two randomly selected timeslots including all their exams while keeping the room allocation invariable.

3.3. Experiments

The investigation of the properties of LAHC is started from examining its response to the variation of its algorithmic parameter: the history length L_h. In the first series of experiments, we demonstrate how the cost function drops throughout the search process. The algorithm was run with our benchmark problems several times while varying L_h. 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. Fig. 2 presents an example of four curves drawn for the *U1817* instance: with $L_h = 1$ (which is the Greedy HC), 5000, 15000 and 30000. It should be noted that the cost drop diagrams plotted for other benchmark instances are similar to this example.

In this figure, all diagrams are started from the same initial solution with the cost function around 1800000. The HC improves the cost almost immediately. The LAHC with $L_h = 5000$ improves the cost more slowly; with $L_h = 15000$ even more slowly and $L_h = 30000$ 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. Moreover, this behavior is typical of many other one-point search heuristics: in SA, TA and GDA the speed of the cost drop can be also regulated by adjusting their cooling factors (see Burke et al., 2004).

In the second series of experiments, we investigate how the slowing of the cost drop affects the final results. We run LAHC on all benchmark problems with three values for L_h: 1, 5000 and 50000. With each value, our algorithm was run 20 times. Tables 3 and 4 show the average results and the average run times for our TSP and exam timetabling datasets.

These tables confirm that the above property of LAHC (the longer the fitness array, the longer the CPU time) holds for all of the tested benchmark instances. Moreover, it could be observed that in these tables, the average processing time of runs with $L_h = 50000$ is approximately 10 times longer than that with $L_h = 5000$. Based on this, we can propose that the CPU time is linearly dependent on the history length. In addition, the tables demonstrate that an increase in the CPU time leads to better final results (again for all benchmark instances).

To visually demonstrate the above properties of LAHC, a more extensive examination was carried out in our third series of experiments. We conducted a high number of runs (around 500) while randomly varying L_h 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 *U1817* dataset only. Fig. 3

Table 3
Results for TSP instances produced by LAHC with different L_h.

Dataset	$L_h = 1$ (HC)		$L_h = 5000$		$L_h = 50000$	
	Cost	Time (sec)	Cost	Time (sec)	Cost	Time (sec)
Rat783	10054	0.10	9352	3.9	9109	38
U1060	260336	0.19	235343	7.5	228749	74
Fl1400	22887	0.33	20759	12.1	20405	115
U1817	68842	0.98	62034	19	59469	178
D2103	98435	1.54	89791	23	86228	221
Pcb3038	159451	4.1	150393	42	144255	391
Fl3795	33244	9.9	31193	71	30171	696

Table 4
Results for exam timetabling instances produced by LAHC with different L_h.

Dataset	$L_h = 1$ (HC)		$L_h = 5000$		$L_h = 50000$	
	Cost	Time (sec)	Cost	Time (sec)	Cost	Time (sec)
Exam_1	5988	1.16	4340	18	3787	174
Exam_2	638	0.31	465	7.2	402	68
Exam_3	11942	3.2	8226	44	7378	439
Exam_4	21545	1.44	14407	33	13278	340
Exam_5	3513	0.42	2761	9.5	2491	96
Exam_6	27116	0.30	25935	4.0	25461	41
Exam_7	5427	1.8	4086	25	3589	245
Exam_8	9210	0.91	7432	14	6701	145
Exam_9	1322	0.11	1060	1.3	997	12.9
Exam_10	13967	0.20	13132	2.4	13013	24
Exam_11	35406	4.0	24869	61	22959	612
Exam_12	5634	0.095	5309	0.74	5234	7.6

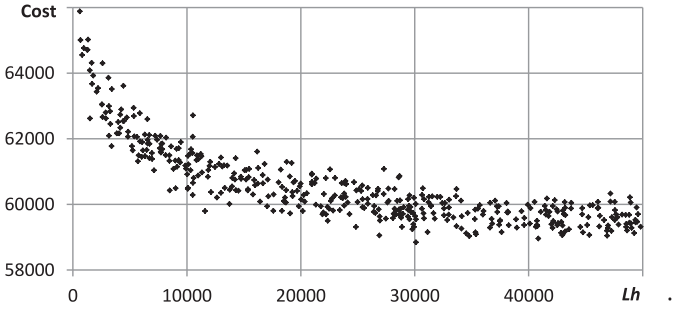

Fig. 3. Dependence of the final cost on L_h for the *U1817* instance.

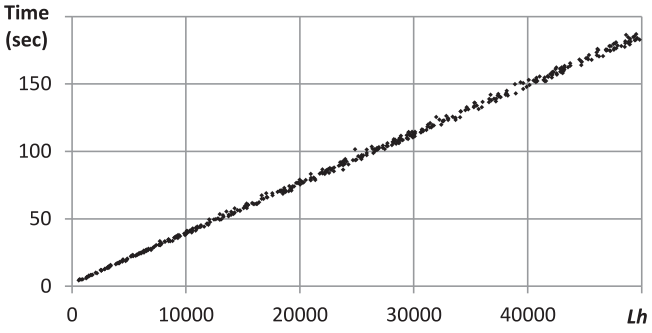

Fig. 4. Dependence of run time on L_h for *U1817* instance.

illustrates the dependence of the final cost on the L_h. Here, each point represents the result of a single run, while its coordinates show the corresponding value of L_h and the final cost.

Even though the results are scattered (which is typical for search algorithms), this diagram shows the clear and distinct dependence of the final cost on L_h in this case. 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 Fig. 3, any result with $L_h = 50000$ is better (in spite of the scatter) than any result with $L_h = 5000$.

Using the same collection of results, the diagram in Fig. 4 represents the dependence of the CPU time on the L_h. Here each run is depicted as a point with coordinates corresponding to L_h and the CPU time.

For the majority of points, this diagram appears as a straight line, which demonstrates the linear dependence between L_h and the CPU time. This property can be employed in practice, for example, knowing the time of a short run, it is possible to calculate an angle coefficient $Time/L_h$ and then use it to set up the necessary history length for a long run within a limited time interval.

4. An assessment of the LAHC performance

The experiments in this section begin with a comparison of the different variants of LAHC. The purpose of this study is to justify the first and the second improvements presented in Section 2. Taking different combinations of these improvements, we obtain four variants of LAHC. A preliminary investigation of these variants has revealed an interesting property: even with the same benchmark instance, all variants of LAHC have different coefficients $Time/L_h$. This suggests that when launching these variants with the same history length, the run time will be different. Presumably, the variant with the longer time will provide the better result. In order to avoid this unfairness, we use a special methodology for the comparison of search algorithms.

4.1. The assessment methodology

In this methodology the performances are expressed by curves rather than scalar numbers. It was introduced by Burke and Bykov (2012) and then used by Bykov and Petrovic (2016). In a similar manner to experiments carried out in the previous section, each

Table 5
The comparison of different variants of LAHC for the *exam1* dataset.

CPU time (seconds)	LAHC-b	LAHC-1	LAHC-2	LAHC
1–20	4844	4805	4643	**4603**
20–40	4414	4378	4293	**4279**
40–60	4242	4222	4156	**4143**
60–80	4133	4128	4067	**4054**
80–100	4089	4075	4011	**3998**
100–120	4046	4021	3963	**3950**
120–140	3987	3982	3920	**3903**
140–160	3981	3952	3890	**3879**
160–180	3931	3915	3861	**3858**
180–200	3915	3892	3848	**3835**

Table 6
The comparison of time cut-offs for SA, TA, GDA and LAHC for the *U1817* dataset.

CPU time (seconds)	SA	TA	GDA	LAHC
1–20	65854	64328	67555	**63325**
20–40	63252	61875	66877	**61644**
40–60	62154	61090	66552	**60982**
60–80	61518	60544	66069	**60485**
80–100	60913	60297	66052	**60231**
100–120	60735	60022	65673	**59889**
120–140	60338	59838	65601	**59744**
140–160	60354	59659	65462	**59634**
160–180	60305	59688	65150	**59596**
180–200	60026	59482	64889	**59439**

competitor variant was run a high number of times (around 500). The history length of each variant was randomly varied so that the resulting CPU times were evenly distributed in the interval of 1…200 seconds. In this way, we produced plots similar to that presented in Fig. 3. Then the horizontal axis (which represents the CPU time) was divided into 10 equal segments (by 20 seconds) and, for each segment, we calculated an average cost among all the results belonged to that segment. Thus, the algorithmic performance is represented as a series of cut-offs by time segments of equal size, having approximately 50 runs for each cut-off.

4.2. The performance of different variants of LAHC

Using the above methodology, we compare four variants of LAHC: the basic one without improvements (LAHC-b), the one where the greedy rule is added (LAHC-1), the variant with enhanced history recoding (LAHC-2) and the final variant with both additions (LAHC). In this experiment, all these variants were tested with the *Exam1* dataset. The resulting cut-offs are presented in Table 5. The best result for each cut-off over the four variants is highlighted in bold.

As expected, these results justify the suggestions given in Section 2: each of the improvements to the basic LAHC provides a certain positive effect, while the final variant of LAHC (with both of them) shows the best performance.

Although we present here the study with just one dataset, the same tendency was shown with all other benchmark instances. So, we investigated the internal mechanisms behind these improvements and have found the following explanation:

- The adding of the greedy rule simply expands the search space. Without this, the algorithm skips some moves which are quite acceptable. Generally speaking, the skipping of good moves gives the same effect as reducing the search time.
- The updating of the fitness array with worse values does not have an overall effect on the main part of the search. However, at the final stage, the search significantly slows down and sometimes can converge prematurely. In contrast, the “better only” updating strategy prevents this slowing, which enables the method to avoid premature convergence.

4.3. A comparison of LAHC with other techniques

Following the above conclusions, we now choose the final variant (LAHC) for the comparison with existing local search methods: Simulated Annealing, Threshold Acceptance and the Great Deluge algorithm. We consider that the results of these techniques are also dependent on the processing time and represent a scattered curve (see Bykov & Petrovic, 2016). This encourages us to use the above methodology once again. It should be noted that the processing time is pre-defined differently (and rather approximately) in different methods (see Burke et al., 2004). Therefore, we now randomly varied the cooling factors in SA, TA and GDA, while this variation was organized to keep an even distribution of CPU times within a necessary interval (1…200 seconds in our case).

In addition, we recognize that in order to achieve a good performance the initial temperatures in SA (and thresholds in TA) require preliminary tuning. Here we followed general suggestions from the literature: The initial temperature of SA was selected so that, at the starting point, the algorithm accepted 85 percent 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. With SA and TA, we applied the geometric cooling schedule whilst, with GDA, we used the linear lowering of the level.

We carry out this comparison in our fifth series of experiments, where we use the same software whilst varying just the acceptance conditions. The results produced by four competitor techniques for the *U1817* dataset are presented in Table 6.

The experiments have shown that for this benchmark instance, a distinctly worse performance is shown by GDA, while LAHC outperforms all competitors in all cut-offs. TA performs slightly worse than LAHC and SA is slightly worse than TA. The performance of the competitor techniques with other benchmark instances can be compared using the similar tables available in the journal online supplement (also from http://www.yuribykov.com/LAHC_supplement/). In this paper, we present just an example of the cut-offs for a single interval. Tables 7 and 8 contain the cut-offs in the middle interval of 100-120 seconds for TSP and exam timetabling instances respectively.

These tables demonstrate that when CPU time is limited by 2 minutes, LAHC performs more strongly than its competitors with 5 out of 7 TSP instances and with 6 out of 12 Exam Timetabling instances (for *Exam_10* instance LAHC performs in the same way as SA). When comparing these results with problem sizes (given in Tables 1 and 2) it can be observed that LAHC outperforms other methods with the larger sized problems (except for the *Exam_2* dataset), while SA or TA show the best performance with the relatively smaller sized problems (see the sizes in Tables 1 and 2).

4.4. The scale independence of LAHC

In the previous experiments, we investigated the LAHC properties, which are rather similar to cooling schedule based techniques (SA, TA and GDA). However, our algorithm has an additional property, which distinguishes LAHC from the above methods: *scale independence*. To demonstrate this property we extend the pure Travelling Salesman Problem in order to bring it closer to real-world tasks. Let us assume that the salesman is interested in minimizing his own travelling expenses (C_r) rather than a distance. The total expenses (C) are proportional to the distance, but the salesman has a company budget (limited by low and upper bounds) which is regulated by the following rules: (a) If the expenses are less than the low bound (B_{lo}) then the salesman should pay the full amount

Table 7
The comparison of cut-offs in the interval of 100–120 seconds for SA, TA, GDA and LAHC for TSP datasets.

Dataset	SA	TA	GDA	LAHC
Rat783	9031	9051	9415	**9029**
U1060	229331	**228080**	240833	228423
Fl1400	20474	**20401**	21146	20421
U1817	60735	60022	65673	**59889**
D2103	88484	87504	94746	**87296**
Pcb3038	151061	147882	158516	**147826**
Fl3795	31718	31176	32369	**30997**

Table 8
The comparison of cut-offs in the interval of 100–120 seconds for SA, TA, GDA and LAHC for the exam timetabling datasets.

Dataset	SA	TA	GDA	LAHC
Exam_1	4164	4131	4358	**4022**
Exam_2	**397**	399	436	404
Exam_3	8330	8169	8750	**8037**
Exam_4	13008	**12607**	13496	12938
Exam_5	2619	2603	2903	**2582**
Exam_6	25435	**25409**	25566	25463
Exam_7	4065	4002	4334	**3942**
Exam_8	7139	7123	7688	**6900**
Exam_9	956	**948**	999	963
Exam_10	**12993**	13012	13081	**12993**
Exam_11	25686	25445	26926	**25048**
Exam_12	**5160**	5189	5216	5179

by himself. (b) Expenses over B_{lo} are returned to him. However, he has to pay a small processing charge of 1 percent of the returned amount. (c) If the expenses exceed the upper bound (B_{up}) the salesman has to pay the overspending amount (plus a processing charge of the whole return). Overall, the combination of rules for C_r can be mathematically expressed by Formula (3).

$$\begin{aligned} C_r &= C && when\, C \le B_{lo} \\ C_r &:= B_{lo} + 0.01(C - B_{lo}) && when\, B_{lo} < C \le B_{up} \\ C_r &= C - 0.99(B_{up} - B_{lo}) && when\, B_{up} < C \end{aligned} \tag{3}$$

These rules taken together compose a monotonically increasing function, which transforms C into C_r. Such monotonic transformation appears as just a nonlinear rescaling of the cost function, illustrated in Fig. 5.

It should be noted that the rescaling in Fig. 5 does not affect 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 $C_r(s_1) > C_r(s_2)$ and if $C(s_1) = C(s_2)$ then $C_r(s_1) = C_r(s_2)$. Correspondingly, all original local and global optima are preserved in this new formulation. This implies that this rescaling does not affect the performance of greedy Hill-Climbing, i.e., having the same initial randomization HC will provide absolutely the same final solutions with the original and with the new problem. Namely, any solution, which HC accepts in the original problem, will be also accepted in the new problem (and vice versa). In that sense, we can consider HC to be a scale independent algorithm.

Furthermore, we investigate how the rescaling of the cost function affects our competitor algorithms. In the final series of experiments, we apply the rescaling (3) to *U1817* dataset (while assigning $B_{\text{lo}} = 200000$ and $B_{\text{up}} = 1000000$) and repeat the previous tests with this new problem. All experimental conditions remain untouched, even the initial temperature and threshold do not require a re-tuning for that new problem. The produced results are presented in Table 9.

We see in the table that the extending of the TSP specification has a dramatic impact on the performance of SA and TA (and to a lesser degree on GDA). SA and TA have completely failed with

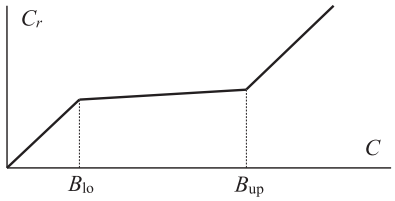

Fig. 5. The dependence between C and C_r.

Table 9
The comparison of cut-offs for SA, TA, GDA and LAHC for *U1817* dataset with the rescaled cost function.

CPU time (seconds)	SA	TA	GDA	LAHC
1–20	67594	67260	67618	**63356**
20–40	67102	66956	67064	**61639**
40–60	66995	66840	66383	**60978**
60–80	67032	66824	66317	**60545**
80–100	67003	66716	66029	**60321**
100–120	66785	66868	65687	**59929**
120–140	66862	66693	65364	**59985**
140–160	66687	66803	65004	**59657**
160–180	66790	66744	64839	**59599**
180–200	67008	66786	64728	**59526**

this new problem. They have ceased to improve the results with an increase in the search time and always performed quite close to the level achieved by HC. 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. These results demonstrate the scale independence of LAHC. It can be also deduced by the same reasoning as carried out for HC: if LAHC accepts a solution in the original problem then it also accepts this solution in the rescaled problem (and vice versa). Thus, being an extension of Hill Climbing, LAHC also inherits its scale independence. However, this is not the case for SA, TA and GDA.

5. 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 competitors 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 400×400 within one minute. The first runner-up had solved the 1000×1000 MS problem. The winning LAHC-based algorithm was able to solve the 2600×2600 constrained MS problem within one minute. Note 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 winning 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 runner-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 1000×1000 MS problem in 5 percent of runs, the algorithm failed to produce MS in 1 minute. Furthermore, with the 2600×2600 MS problem, up to 62 percent of runs (depending on the constraint position) were unsuccessful.

In contrast, the application of LAHC to the unstudied MS problem was successful. With constant $L_h = 20000$, LAHC has shown 100percent reliability on all runs with either 200×200, 1000×1000 and 2600×2600 MS problems. Moreover, the described LAHC-based algorithm has the potential to solve MS problems of much larger size than the 2600×2600 limit within 1 minute. In reality, this limit was caused by hardware restrictions rather than the algorithm 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.yuribykov.com/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 all these heuristics with the MS problem to verify the fact that LAHC represents the best choice among them.

6. Conclusions and future work

In this paper, we have proposed a new local search technique (LAHC), which has a range of unique properties that can be outlined as follows:

- It is almost as simple as the greedy HC, but much more powerful. So, it can be easily implemented in experimental and practical systems. Also, the developers can substitute HC in existing systems by a stronger search technique with minimum effort.
- It intelligently uses the information collected during previous iterations. It can be viewed as a further variant of Adaptive Memory Programming (see Taillard, Gambardella, Gendreau, & Potvin, 2001).
- It is dependent on a single algorithmic parameter, which regulates the CPU time. The presented experiments have revealed that the CPU time is approximately proportional to this parameter. Correspondingly, it can be well-tuned with less effort than most modern metaheuristics. This is especially attractive to practitioners.
- It outperforms SA, TA and GDA on most of the benchmark problems (especially the larger sized ones). Taking into account that the competitor algorithms are well studied, but the studies on LAHC have just begun, this suggests that LAHC has significant potential for further development.
- It is not dependent on scales in contrast to cooling-schedule based algorithms. This suggests its stronger reliability on nonlinear problems and this could be beneficial in new application areas.
- 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.

This paper represents the introduction of LAHC to the literature. Of course, it requires further intensive study. It may be possible to propose further 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 beneficial in further hybrid approaches where the greedy HC is applicable, for example, in Iterated Local Search or Memetic Algorithms.

In addition, the properties of LAHC should be also further investigated. The demonstrated scale independence might occur as an additional advantage of LAHC when applied to some specific nonlinear problems. Also, it would be worth investigating the LAHC angle coefficient revealed in Fig. 4. It is different for each particular instance and might reflect certain properties of the studied problems.

Finally, we have proposed and applied a methodology for examining the scale independence of search heuristics using the 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 methods with the same rescaled problem.

Acknowledgments

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 opinions about LAHC. We would also like to thank the anonymous referees for their valuable advice in significantly improving this paper.

References

Abuhamdah, A. (2010). Experimental result of late acceptance randomized descent algorithm for solving course timetabling problems. *International Journal of Computer Science and Network Security, 10*, 192–200.

Alzaqebah, M., & Abdullah, S. (2014). An adaptive artificial bee colony and late acceptance hill-climbing algorithm for examination timetabling. *Journal of Scheduling, 17*(3), 249–262.

Anderson, K., Vidal, R. V. V., & Iverson, V. B. (1993). Design of teleprocessing communication network using simulated annealing. *Springer Lecture Notes in Economics and Mathematical Systems, 396*, 201–216.
Appleby, J. S., Blake, D. V., & Newman, E. A. (1960). Techniques for producing school timetables on a computer and their application to other scheduling problems. *The Computer Journal, 3*, 237–245.
Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2006). *The traveling salesman problem: A computational study*. Princeton University Press.
Burke, E. K., & Bykov, Y. (2008). A late acceptance strategy in hill-climbing for exam timetabling problems (extended abstract). In *Proceedings of the 7th international conference on the practice and theory of automated timetabling (PATAT 2008)* August 2008.
Burke, E. K., & Bykov, Y. (2012). *The late acceptance hill-climbing heuristic*. Technical Report CSM-192, Computing Science and Mathematics,, UK: University of Stirling.
Burke, E. K., & Bykov, Y. (2016). An adaptive Flex-Deluge approach to university exam timetabling. *Informs Journal on Computing, 28*(4), 781–794.
Burke, E. K., Bykov, Y., Newall, J., & Petrovic, S. (2004). A time-predefined local search approach to exam timetabling problems. *IIE Transactions, 36*(6), 509–528.
Burke, E. K., Elliman, D., Ford, P., & Weare, R. (1996). Examination timetabling in British universities: A survey. *Springer Lecture Notes in Computer Science, 1153*, 76–90.
Burke, E. K., & Newall, J. (2003). Enhancing timetable solutions with local search methods. *Springer Lecture Notes in Computer Science, 2740*, 344–354.
Bykov, Y., & Petrovic, S. (2016). A Step counting Hill Climbing algorithm applied to university examination timetabling. *Journal of Scheduling, 19*(4), 479–492.
Carter, M. W., & Laporte, G. (1996). Recent developments in practical examination timetabling. *Springer Lecture Notes in Computer Science, 1153*, 3–21.
Cohn, H., & Fielding, M. (1999). Simulated annealing: Searching for an optimal temperature schedule. *SIAM Journal on Optimization, 9*(3), 779–802.
Dueck, G. (1993). New optimization heuristics. The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics, 104*, 86–92.
Dueck, G., & Scheurer, T. (1990). Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics, 90*, 161–175.
Hu, T. C., Kahng, A. B., & Tsao, C.-W. A. (1995). Old bachelor acceptance: A new class of non-monotone threshold accepting methods. *ORSA Journal on Computing, 7*(4), 417–425.
Jackson, W., Özcan, E., & Drake, J. H. (2013). Late Acceptance-based selection hyper-heuristics for cross-domain heuristic search. In *Proceedings of the 13th annual workshop on computational intelligence (UKCI), September 2013, Guildford, UK* (pp. 228–235).
Johnson, D. S., Aragon, C. R., McGeoch, L. A., & Schevon, C. (1989). Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning. *Operations Research, 37*(3), 865–892.
Johnson, D. S., Aragon, C. R., McGeoch, L. A., & Schevon, C. (1991). Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. *Operations Research, 39*(3), 378–406.
Johnson, D. S., & McGeoch, L. A. (1997). The Travelling Salesman Problem: A case study in local optimization. In E. H. L. Aarts, & J. K. Lenstra (Eds.), *Local search in combinatorial optimization* (pp. 215–310). London: J.Willey and Sons.
Kirkpatrick, S., Gelatt, J. D. C., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*, 671–680.
Lewis, R. (2008). A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum, 30*(1), 167–190.
Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the travelling-salesman problem. *Operations Research, 21*(2), 498–516.
McCollum, B., Shaerf, A., Paechter, B., McMullan, P. J., Lewis, R., Parkes, A. J., Di Gaspero, L., Burke, E. K., & Qu, R. (2010). Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing, 22*, 120–130.
McMullan, P. (2007). An extended implementation of the great deluge algorithm for course timetabling. *Springer Lecture Notes in Computer Science, 4487*, 538–545.
Ninio, M., & Schneider, J. J. (2005). Weight annealing. *Physica, 349*, 649–666.
Obit, J. H., Landa-Silva, D., Ouelhadj, D., & Sevaux, M. (2009). Non-linear great deluge with learning mechanism for solving the course timetabling problem. In *Proceedings of MIC 2009: The VIII metaheuristic international conference* July 2009, id-1-10.
Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research, 41*, 421–451.
Qu, R., Burke, E. K., McCollum, B., Merlot, L. T. G., & Lee, S. Y. (2009). A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling, 12*, 55–89.
Ozcan, E., Birben, M., Bykov, Y., & Burke, E. K. (2009). Examination timetabling using late acceptance hyper-heuristics. In *Proceedings of the 2009 IEEE congress on evolutionary computation (CEC 2009)* (pp. 997–1004). May 2009.
Schaerf, A. (1999). A survey of automated timetabling. *Artificial Intelligence Review, 13*(2) 187–127.
Taillard, E. D., Gambardella, L. M., Gendreau, M., & Potvin, J.-Y. (2001). Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operational Research, 135*, 1–16.
Thompson, J. M., & Dowsland, K. A. (1996). General cooling schedules for simulated annealing based timetabling system. *Springer Lecture Notes in Computer Science, 1153*, 345–363.
Tierney, K. (2013). Late acceptance hill climbing for the liner shipping fleet repositioning. In *Proceedings of the 14th EU/ME workshop* (pp. 21–27).
Vancroonenburg, W., & Wauters, T. (2013). Extending the late acceptance metaheuristic for multi-objective optimization. In *Proceedings of the 6th multidisciplinary international scheduling conference: Theory & applications (MISTA2013)* (pp. 652–655). August 2013.
Verstichel, J., & Vanden Berghe, G. (2009). A late acceptance algorithm for the lock scheduling problem. *Logistic Management*, (5), 457–478 2009.
Yuan, B., Zhang, C., & Shao, X. (2015). A late acceptance hill-climbing algorithm for balancing two-sided assembly lines with multiple constraints. *Journal of Intelligent Manufacturing, 26*(1), 159–168.