CLAHC - custom late acceptance hill climbing first results on TSP

Sylvain Clay, Lucien Mousin, Nadarajen Veerapen, Laetitia Jourdan

CLAHC – Custom Late Acceptance Hill Climbing: first results on TSP

Sylvain Clay
sylvain.clay.etu@univ-lille.fr
Univ. Lille, CNRS, Centrale Lille, *UMR 9189 - CRIStAL*
F-59000 Lille, France

Lucien Mousin
lucien.mousin@univ-catholille.fr
Lille Catholic University, FGES
Lille, France

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

Laetitia Jourdan
laetitia.jourdan@univ-lille.fr
Univ. Lille, CNRS, Centrale Lille, *UMR 9189 - CRIStAL*
F-59000 Lille, France

ABSTRACT

The Late Acceptance Hill Climbing heuristic is a Hill Climbing algorithm that uses a record of the history of objective values of previously encountered solutions in order to decide whether to accept a new solution. Literature has shown that Late Acceptance Hill Climbing is generally better at not getting stuck in local optima because of the history. In this paper, we propose and investigate a simple, yet effective, modification to Late Acceptance Hill Climbing, where we change how values in the history are replaced. In our tests, referring to the Traveling Salesman Problem, we analyze the behavior of the proposed approach for different history sizes. We also show that the simple change in the algorithm allows the heuristic to find better solutions than the original one on most of the instances tested.

CCS CONCEPTS

• **Computing methodologies** → **Search methodologies**.

KEYWORDS

Local Search, Hill Climbing, Late Acceptance Hill Climbing

1 INTRODUCTION

Combinatorial optimization aims to find good solutions to NP-hard problems within an affordable execution time. Local searches are among the simplest methods to achieve that. They involve trying to reach a solution with the best possible objective function value, starting from an initial solution and moving from solution to solution using a neighborhood operator. Numerous local search methods have been investigated in the literature, including Simulated Annealing (SA) [8], Threshold Accepting (TA) [4] or the Great Deluge Algorithm (GDA) [3]. Late Acceptance Hill Climbing (LAHC) [2] is another related method, and it has been shown to be very competitive in previous work [5]. It is inspired by the simplest local search method, Hill Climbing (HC) [7].

In this paper, we present Custom Late Acceptance Hill Climbing (CLAHC), a new version of LAHC which produced an improvement in the majority of cases we examined.

The paper is structured as follows. We first give more details about the difference between these two heuristics in Section 2. Then we present the experimental protocol and results in Section 3. Finally we present the conclusion and further possible investigations into CLAHC in Section 4.

2 CLAHC - AN IMPROVED LAHC

LAHC uses the same principle as a simple HC, and improves it by using a record of the history of objective function values of the accepted solutions. We will refer to this as the *history* for short. In practice, this is a sliding window of objective function values that were encountered earlier during the execution of the heuristic. The history is used as part of an acceptance criterion for any new solution that we encounter. If a candidate solution has a better value than the oldest value in the history, then this solution is accepted and we carry on from there. This idea allows LAHC to avoid, to some extent, the local optimum problem of HC, which quickly reaches a final solution that can be far from the global optimum. This problem occurs because HC is not allowed to accept solutions that have a worse value than the current solution.

Beside its effectiveness, one of the most interesting properties of LAHC is that it requires only one parameter, the history size, so it is fairly simple to use. It has been shown in previous works [2] that increasing the size of the history has an impact on both the final solution and the execution time. More complex metaheuristics have been experimented with using LAHC as a base, like PLAHC [1] which consists in an Iterative Local Search [6] that uses LAHC with an increasing size of history, starting at 1 and doubling at each iteration of LAHC. The intention of this version is to free oneself from the parameter of LAHC, the size of the history, and still be able to find a good solution in the end.

In contrast, CLAHC works almost exactly like LAHC, with a single difference which is the replacement criterion for a value in the history at each iteration of the algorithm (Algorithm 1). With LAHC, first we decide if a solution is accepted as our new current solution to continue the algorithm by comparing it with our current solution and with the value in the history at the current index. If the new value is better than any of these two, then we replace the current solution by this new solution. Then we do the same thing whether we have accepted a new solution or not : we compare the value of the current solution with the one in the history at the current index. If the value in the history is worse than the value of our solution, then we replace the value in the history with the value of our solution.

Algorithm 1 CLAHC

```
Require: l, the history size
Ensure: s, the final solution
  s = new solution
  c(s) = score of s
  for i in [0, ... , l-1] do
    f[i] = c(s)
  end for
  i = 0 //number of evaluations
  count = 0 //number of successively non-improving evaluations
  while i < 100000 or count < i * 0.02 do
    s' = new solution //neighbor of s
    c(s') = objective value of s'
    if c(s') ≥ c(s) then
      count++
    else
      count = 0
    end if
    v = i mod l //the index where we are in the history
    if c(s') < f[v] or c(s') ≤ c(s) then
      s = s' //we use this solution to continue
    end if
    if c(s') < f[v] then //use s' instead of s in LAHC
      f[v] = c[s'] //s' instead of s
    end if
    i++
  end while
  return s
```

With CLAHC, things are different: we follow the same approach to decide whether to accept a new solution to continue the algorithm, however to decide whether we replace the value in the history, we do not compare it with the value of our current solution, but with the one of the solution that was candidate (highlighted in red in Algorithm 1). In other words, if the candidate solution was accepted to continue the algorithm, then CLAHC does exactly the same thing as LAHC. But if the solution was not accepted, then CLAHC compares the value in the history with the value of the solution that was not accepted, and replaces the value in the history with the value of this solution if this one is better. In other words, this means that CLAHC actually never replaces the value in the history if the candidate solution was not accepted.

This behavior allows CLAHC to keep worse values than LAHC would have allowed in its history. At first glance, it could seem like a problem, but previous works [1] have posited the idea that diversity in the history is necessary to allow the algorithm to continue its execution. With CLAHC, we propose the idea that diversity in the history also improves the performance of the algorithm, because it increases the diversification strength of the algorithm. Conversely, this behavior also has an impact on the execution time. For a history of the same size, CLAHC is slower than LAHC. But that is not a problem since CLAHC seems to perform better than LAHC even with a smaller history.

Table 1: Budget of the number of evaluations for each instance

Instance	Budget
rat783	54,000,000
u1060	82,500,000
fl1400	110,000,000
u1817	315,000,000
d2103	370,000,000
pcb3038	830,000,000
fl3795	1100,000,000

3 EXPERIMENTAL RESULTS

3.1 Protocol

We consider the Traveling Salesman Problem (TSP) for our experiments and, in particular, instances from TSPLIB [9]. Briefly, in the TSP, we have a set of cities and we try to find the shortest cycle that links all of them while passing through every city only once. The neighborhood operator we used is the 2-opt operator, which breaks two edges in the solution and reconnects the resulting paths in the opposite order. The initial solution is random.

A configuration is an association of an algorithm (LAHC or CLAHC), a history size and an instance. For every configuration, we performed 30 runs using 30 different random seeds. We used 7 instances: rat783, u1060, fl1400, u1817, d2103, pcb3038 and fl3795. The number in the instance name indicates the respective number of cities, i.e. the size of the instance. We also chose a limited budget of function evaluations for every instance, as detailed in Table 1, in order to take the size and difficulty of the instance into consideration.

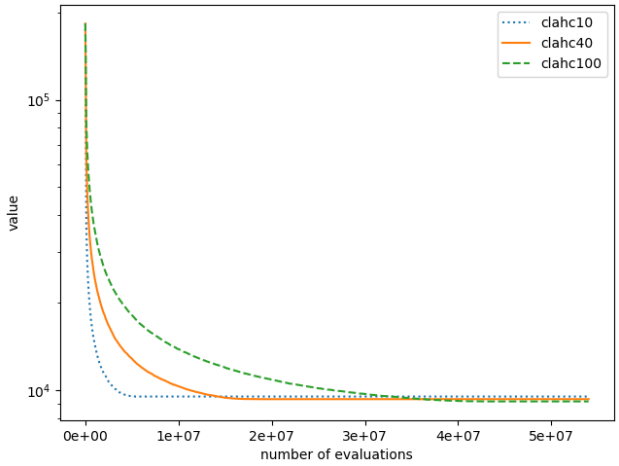
Both LAHC and CLAHC were implemented using the same MH-Builder platform. This is a C++ metaheuristics platform that is under development in our research team. As described previously, and as implemented, the only difference between LAHC and CLAHC is the history replacement criterion. Experiments were carried out on a Lenovo Z50-70 laptop, with a 4-core Intel Core i7-4510U 2.00GHz CPU and 8 GB of RAM.

3.2 Results

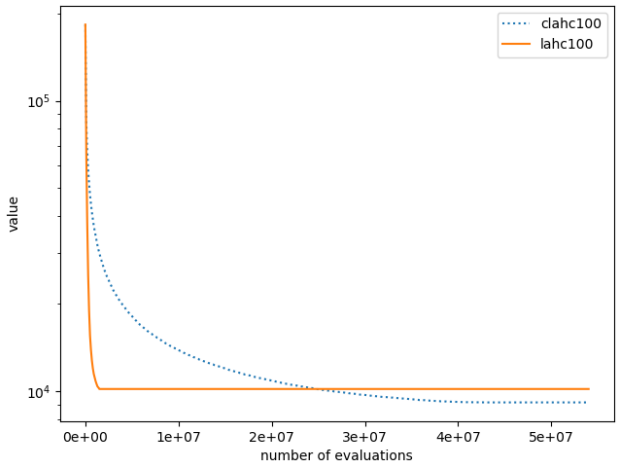
We first wanted to assess the impact of the history size on the performance and the execution time of CLAHC. In Table 2, we compare two versions: CLAHC_{10} with a history size of 10, and CLAHC_{100} with a history size of 100. We could have used a bigger history, but the algorithm would not have been able to converge with the limited budget we chose. This table shows that, just like LAHC, CLAHC gets better when we increase the size of its history.

In Figure 1 we compare execution traces on the rat783 instance. This single instance is presented here in order to remain within the paper's page limit constraint, but the general observed behavior is the same across the different instances tested. The three plots in the figure use a logarithmic scale for the y-axis (objective value) in order to better observe the inflection in the curves.

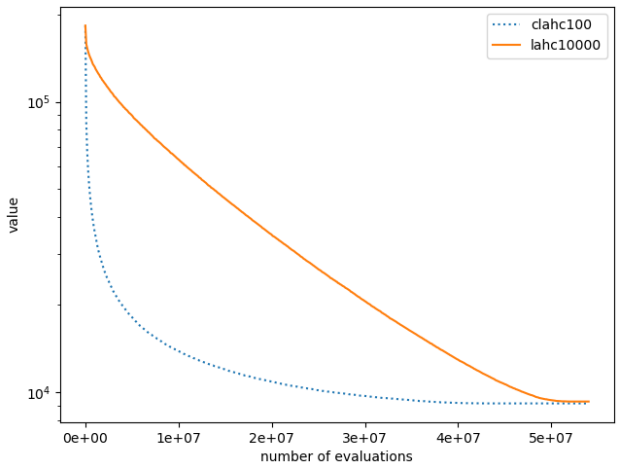
Figure 1a compares CLAHC execution traces for different sizes of history. It shows that even though our algorithm reaches a better



(a) Traces of $CLAHC_{10}$, $CLAHC_{40}$ and $CLAHC_{100}$ on rat783.



(b) Traces of $CLAHC_{100}$ and $LAHC_{100}$ on rat783.



(c) Traces of $CLAHC_{100}$ and $LAHC_{10000}$ on rat783.

Figure 1: Execution traces on the rat783 instance, with a logarithmic scale on the y-axis.

Table 2: CLAHC average results over 30 runs with standard deviation in subscript, and percentage deviation from the best known solution.

Instance	$CLAHC_{10}$		$CLAHC_{100}$	
rat783	$9,518_{88}$	8%	$\mathbf{9,141}_{54}$	3.8%
u1060	$239,142_{1937}$	6.7%	$\mathbf{229,692}_{841}$	2.5%
fl1400	$21,616_{349}$	7.4%	$\mathbf{20,803}_{242}$	3.4%
u1817	$62,344_{525}$	9%	$\mathbf{59,378}_{279}$	3.8%
d2103	$89,942_{780}$	11.8%	$\mathbf{85,498}_{529}$	6.3%
pcb3038	$149,202_{711}$	8.3%	$\mathbf{142,672}_{506}$	3.6%
fl3795	$31,371_{660}$	9%	$\mathbf{29,990}_{368}$	4.2%

final value, it gets slower when we increase the size of its history. This is reminiscent of LAHC's original behavior. Then we compared CLAHC and LAHC. We first compared those two algorithms with the same size of history in Figure 1b. It shows that LAHC is faster in the beginning for the same history size, but is really far from a good objective function value at the end of its execution, to the point where it is not really better than a simple HC.

So we tried to find two different history sizes for LAHC and CLAHC that would allow the two algorithms to converge and be the most efficient with the same evaluation budget for every instance. We used $CLAHC_{100}$ and, depending on the instance, we compared it with LAHC with a history size between 10,000 and 25,000. Table 3 shows these results. In this table, $LAHC_*$ is the version of LAHC with the history size that performs the best on a given instance. For example, on instance rat783, $CLAHC_{100}$ is compared to $LAHC_{10000}$, because the number of evaluations that they need to converge is the same. This table also contains the standard deviation of the concerned algorithm's solution. We also added the HC solution and the percentage deviation from the best known solution for every instance, so we have two reference points with which we can compare our results. Wilcoxon and Friedman statistical tests were performed on our results, and the best ones are in bold. We see that, for a same budget, CLAHC performs better than LAHC on every instance with the exception of fl1400.

We also wanted to see which one of LAHC or CLAHC was the fastest to reach a good solution in the beginning of the execution. Figure 1c shows that $CLAHC_{100}$ is faster than $LAHC_{10000}$ to reach a good solution, and at the end, the final value of CLAHC is still better. Even though it is not presented graphically here, even on instance fl1400 for which CLAHC is not as good as LAHC in the end, CLAHC reaches a good value faster at the beginning of the execution. We can try to interpret the worse results on fl1400 by saying that since CLAHC is faster to converge in the beginning, then it might have a bigger chance to be trapped in a local minimum than LAHC.

Figure 2 contains boxplots also showing that $CLAHC_{100}$ is faster than $LAHC_{10000}$ to reach a good solution over 30 runs, and that its performance is very steady. This figure here uses a linear scale for the y-axis (instead of the logarithmic scale in previous figures), so we can see more clearly the difference of speed between $CLAHC_{100}$ and $LAHC_{10000}$.

Table 3: CLAHC vs LAHC average final solutions over 30 runs, standard deviation and percentage deviation from the best known solution. $LAHC_*$ is the best LAHC configuration (with a history size chosen between 10,000 and 25,000) for an instance with the fixed budget.

Instance	HC		$LAHC_*$		$CLAHC_{100}$	
rat783	$10,051_{84}$	14%	$9,296_{52}$	5.5%	$\mathbf{9,141}_{54}$	3.8%
u1060	$256,117_{3607}$	14%	$232,734_{1345}$	3.8%	$\mathbf{229,692}_{841}$	2.5%
fl1400	$22,106_{392}$	9.8%	$\mathbf{20,582}_{141}$	2.2%	$20,803_{242}$	3.4%
u1817	$69,225_{751}$	21%	$60,442_{448}$	5.6%	$\mathbf{59,378}_{279}$	3.8%
d2103	$99,338_{1025}$	23%	$87,380_{729}$	8.6%	$\mathbf{85,498}_{529}$	6.3%
pcb3038	$159,325_{1074}$	15.7%	$145,316_{529}$	5.5%	$\mathbf{142,672}_{506}$	3.6%
fl3795	$33,634_{716}$	17%	$30,406_{335}$	5.6%	$\mathbf{29,990}_{368}$	4.2%

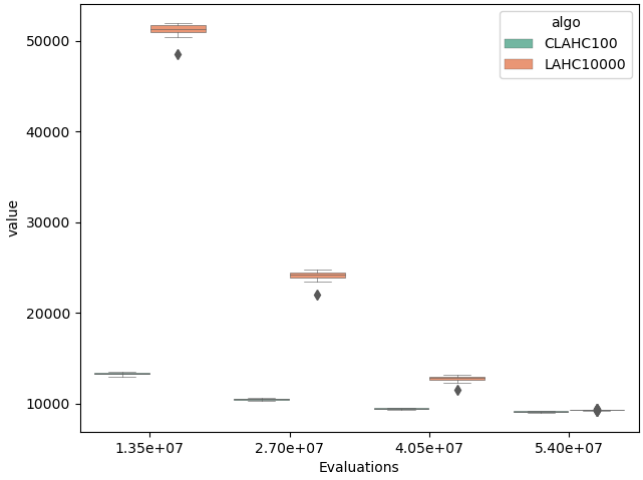


Figure 2: Boxplots for 30 runs of $CLAHC_{100}$ and $LAHC_{10000}$ over time on rat783.

All these observations tend to indicate that CLAHC performs better than LAHC, because 6 out of the 7 instances that we used during our tests got better results, and even for the only instance where it is not as good as LAHC, CLAHC is faster in the beginning, meaning it might be more effective to use it in more complex methods like iterative local searches with different restart mechanisms.

We can also add the fact that the history size of CLAHC to reach a solution as good as LAHC is smaller, meaning that CLAHC uses less memory.

4 CONCLUSION AND FUTURE WORKS

In this paper, we proposed to alter the behavior of Late Acceptance Hill Climbing by changing how new values are accepted in the history it maintains.

Whereas with LAHC the last accepted solution value was added to the history, our method adds the value of the last candidate solution. This naturally allows for worse values to stay in the history.

We tested this simple change on 7 TSP instances. Our results show that, on the instances we used, our contribution CLAHC reaches, in all cases but one, a better final solution, is faster in the beginning of the execution, and uses less memory than LAHC, which already is considered as one of the most robust local search metaheuristics by some previous works [5].

This highlights the importance of the variety in the history of LAHC, and we can imagine that using other and more complex replacement criteria than the one we used for CLAHC could give even better results at the end of the day.

Another interesting point to study would be the use of a greedy heuristic for the initial solution, and study the differences between CLAHC and LAHC in that case.

Also, CLAHC being a metaheuristic, it could be adapted easily to other problems, since the only thing that would need to be changed is the neighborhood operator. This will be investigated in further works.

REFERENCES

[1] Mosab Bazargani and Fernando G. Lobo. 2017. Parameter-less late acceptance hill-climbing. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017*. ACM, 219–226.
[2] Edmund K. Burke and Yuri Bykov. 2017. The late acceptance Hill-Climbing heuristic. *Eur. J. Oper. Res.* 258, 1 (2017), 70–78.
[3] Gunter Dueck. 1993. New Optimization Heuristics: The Great Deluge Algorithm and the Record-to-Record Travel. *J. Comput. Phys.* 104, 1 (1993), 86–92.
[4] Gunter Dueck and Tobias Scheuer. 1990. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *J. Comput. Phys.* 90, 1 (1990), 161–175.
[5] Alberto Franzin and Thomas Stützle. 2017. Comparison of Acceptance Criteria in Randomized Local Searches. In *EA 2017 Revised Selected Papers (Lecture Notes in Computer Science)*, Vol. 10764. Springer, 16–29.
[6] Fred W. Glover and Gary A. Kochenberger (Eds.). 2003. *Handbook of Metaheuristics*. International Series in Operations Research & Management Science, Vol. 57. Kluwer / Springer. https://doi.org/10.1007/b101874
[7] Holger H. Hoos and Thomas Stützle. 2004. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann.
[8] Scott Kirkpatrick, D. Gelatt Jr., and Mario P. Vecchi. 1983. Optimization by Simulated Annealing. *Sci.* 220, 4598 (1983), 671–680.
[9] Gerhard Reinelt. 1991. TSPLIB - A Traveling Salesman Problem Library. *INFORMS J. Comput.* 3, 4 (1991), 376–384. https://doi.org/10.1287/ijoc.3.4.376