Simulated Annealing with Penalization for University Course Timetabling

Edon Gashi · Kadri Sylejmani · Adrian Ymeri

1 Introduction

In this paper we present a simulated annealing algorithm for solving the problem of University Course Timetabling as formulated in the International Timetabling Competition (ITC) 2019 [6].

The presented algorithm is based on a modified version of simulated annealing. It utilizes a suitable cooling function [5] and an adaptive evaluation function [7], both of which have proven useful for normalizing the varying characteristics of problem instances.

The algorithm searches both feasible and infeasible regions of the solution space, where the latter is dealt with by using a combination of incremental penalization and narrowed search on specific hard constraints.

The solver based on this algorithm has placed among the five finalists of the International Timetabling Competition 2019.

2 Solution approach

Solution representation A given solution is always modeled to be complete. This means that variables are always assigned to some value, even if their configuration evaluates to an invalid solution. A solution is the list of all variables V_i (where $i = 1..n$) and their values. A sample representation of

E. Gashi
University of Prishtina
E-mail: edon.gashi@uni-pr.edu

✉ K. Sylejmani
University of Prishtina
E-mail: kadri.sylejmani@uni-pr.edu

A. Ymeri
University of Prishtina
E-mail: adrian.ymeri@studentet.uni-pr.edu

a given solution is $S = \{V_1, V_2, V_3, ..., V_i, ..., V_n\}$, where the assignment of variable V_i in a solution S is modeled to have three components CT - Class Time, CR-Class Room or SE- Student Enrollment (i.e. $V = \{CT, CR, SE\}$), where CT = {Class ID, Time slot}, CR = {*Class ID*, *Room Index*}, and SE = {(*Student ID*, *Course Index*), *Class Chain Index*}. Variable n represents the total number of all course configurations and the *Class Chain Index* represents a particular assignment combination of a given course.

We maintain three separate penalties to determine the state of the solution: hard penalty, class overflows, and soft penalty. Soft penalty is calculated using the identical rules described by [6], and hard penalty is calculated as follows:

- A conflict between a pair of classes gives 1 hard penalty point.
- A time assignment conflicting with a room's unavailable schedule gives 1 hard penalty point.
- An unsatisfied required constraint gives hard penalty points equal to the soft penalty the constraint would give if it were not required.

Class overflows penalty is the sum of all over-enrollments on classes. We maintain this as a separate penalty because it is not as constrained as hard penalty and is easier to satisfy.

Neighborhood function A mutation is an operation that changes a single variable. We hold two lists of possible mutations: feasible mutations and infeasible mutations. Infeasible mutations are applied on solutions with non-zero hard penalty, and they do not contain operations on students.

In concrete terms, a mutation is a single operation that either changes: (1) the schedule of a class, (2) the room of a class, or (3) the class configuration of a student attending a particular course.

The neighborhood operator has a 50% chance of performing a single mutation (selected randomly from the possible mutations described above), and a 50% chance of performing up to a maximum of three mutations, where the number of mutations is selected at random with uniform probability.

Initial solution The initial solution S is deterministically assigned by giving each variable an iterating natural number in the domain from 1 to 3, which map the set of possible variable values $V = \{CT, CR, SE\}$, respectively. If the attempted number exceeds the domain size, then the last number in the domain is taken.

Cooling function The cooling schedule in Equation (1) is based on [5], where the value of β has been assigned empirically. Because of the varying amount of time it takes to solve different instances, a fixed restart temperature is used once the hard penalty hits 0. In Equation (1), variable t represents the temperature in the current iteration, whereas variable t' represents the temperature for the next iteration.

$$t' = \frac{t}{1 + \beta t} \tag{1}$$

Evaluation function There are a few types of evaluation functions, depending on the phase and context. In equations (2) and (3), p_h stands for hard penalty, p_s for soft penalty, and p_c for class overflows penalty, whereas c_1 and c_2 are empirically defined constants. The function $round_2$ means rounding to 2 digits after the decimal separator. The worst soft penalty of a problem is calculated statically by taking the worst case penalty of all constraints and assignments.

$$\text{searchPenalty}(s) = \begin{cases} c_1 p_h + \text{round}_2(c_2 p_c + \text{normalize}(p_s)), & p_h > 0 \\ c_2 p_c + \text{normalize}(p_s), & p_h = 0 \end{cases} \tag{2}$$

$$\text{normalize}(p_s) = \frac{p_s}{\text{worst soft penalty of problem}} \tag{3}$$

The simulated annealing acceptance condition compares the difference between values of f_{stun} [7], which is shown in Equation (4). The constant γ is defined empirically, whereas f_0 denotes the quality of the best solution. Thus, the energy difference ΔE in simulated annealing is defined in Equation (5), where s and s' represent the current and next solution, respectively.

$$f_{\text{stun}}(x) = 1 - \exp\left[-\gamma(f(x) - f_0)\right] \tag{4}$$

$$\Delta E(s', s) = f_{\text{stun}}\left(\text{searchPenalty}(s')\right) - f_{\text{stun}}\left(\text{searchPenalty}(s)\right) \tag{5}$$

Particular class-time and class-room combinations can become penalized over time. The *modifiedPenalty* function defined in Equation (6) is expressed as the search penalty combined with the total sum of all penalties of present features in a solution s.

$$\text{modifiedPenalty}(s, \mathit{penalties}) = \text{searchPenalty}(s) + \sum_{x}^{\mathit{features}_s} \mathit{penalties}_x \tag{6}$$

Penalization is performed after local timeouts for each feature by incrementing a constant multiplied by the hard penalty contribution of that particular feature.

Sometimes hard constraints persist for many timeouts. In such cases, we pivot the solution by performing hill-climbing with a different penalization function, as shown in Equation (7). A limited subset of persistent constraints (*focus*) is taken and random walks are performed until the solution has reached a sufficiently different shape.

$$\text{focusedPenalty}(s, \mathit{focus}) = f_{\text{stun}}\left(\text{searchPenalty}(s)\right) + \sum_{x}^{\mathit{focus}} \text{penalty}(x) \tag{7}$$

Algorithms 1 and 2 provide a simplified overview of the solver.

Algorithm 1 Simulated annealing with penalization

```
 1: procedure SOLVE(initial solution)
 2:     t ← initial temperature
 3:     penalties ← initial penalties
 4:     best ← initial solution
 5:     local best ← ∞
 6:     local timeout ← 0
 7:     current ← best
 8:     while stopping criteria not met do
 9:         t ← cool(t)
10:         candidate ← mutate(current)
11:         if candidate better than best then
12:             best ← candidate
13:         end if
14:         if searchPenalty(candidate) < local best then
15:             local best ← searchPenalty(candidate)
16:             local timeout ← 0
17:         else
18:             local timeout ← local timeout + 1
19:         end if
20:         if modifiedPenalty(candidate) < modifiedPenalty(current) then
21:             current ← candidate
22:         else if accept(current, candidate, t) then
23:             current ← candidate
24:         end if
25:         if local timeout > limit then
26:             local best ← ∞
27:             local timeout ← 0
28:             t ← restart temperature
29:             persistent constraints ← {constraints with age > age limit}
30:             if infeasible(current) ∧ persistent constraints ≠ ∅ then
31:                 focused constraints ← oldest 3 persistent constraints
32:                 current ← ConstraintSearch(current, focused constraints)
33:             else
34:                 penalties ← scale(penalties)
35:             end if
36:         end if
37:     end while
38:     return best
39: end procedure
```

Algorithm 2 Focused search on particular constraints

```
 1: procedure CONSTRAINTSEARCH(solution, focused constraints)
 2:     timeout ← 0
 3:     while timeout < timeout limit do
 4:         candidate ← RandomWalk(solution, distance)
 5:         if focusedPenalty(candidate) < focusedPenalty(solution) then
 6:             solution ← candidate
 7:             timeout ← 0
 8:         else
 9:             timeout ← timeout + 1
10:         end if
11:     end while
12:     return solution
13: end procedure
```

3 Results and conclusion

The presented algorithm managed to solve all instances of the ITC2019 in time limit of maximum 24 hours. It has won first place in both 1st and 2nd milestone of the competition [6], and has placed third in the final round. In addition, our solver has won the prize as the best approach in the open source category.

In Table 1, we present the comparison results of our approach against the results of four of the other finalists, namely Holem et al.[3], Rappos et al. [1], Er-rahaimini [2] and Lemos et al. [4]. In addition, we have also included the results presented by Müller (one of the organizers of the competition), who, after the end of the competition, has published the results obtained by the solver that is used by UniTime timetabling system. The presented results of our approach (tagged as Edon et al.) are the best results that have been achieved when running the solver for 24 hours for each instance. The results in Table 1 show that our approach is outperformed, in all of the instances, by the the approaches of Holem et al.[3] and Müller, whereas Rappos et al.[1] performs better in 23 (out of 30) instances. Our approach performs better than the approach of Rappos et al.[1] in seven instances, better than the approach Er-rahaimini [2] in 19 instances, and better than the approach of Lemos et al. [4] in 21 instances. In addition, the solutions of our solver have a gap of less than 15% from the best known solutions in 5 (out of 30) instances. Furthermore, the average gap from the best known solution for early, middle and late chunks of instances are about 80%, 90% and 230%, respectively.

Overall, this model shows that it can solve complex and large instances with distinct features in terms of number of courses, number of students, number of rooms, as well as other distribution and special constraints.

References

1. Rappos Efstratios, iemard Eric, Stephan Robert, and Jean-Francois Heche. International timetabling competition 2019: A mixed integer programming approach for solving university timetabling problems. 2021.
2. Karim Er-rhaimini. Forest growth optimization for solving timetabling problems. 2021.
3. Dennis Søren Holm, Rasmus Ørnstrup Mikkelsen, Matias Sørensen, and Thomas Jacob Riis Stidsen. A mip formulation of the international timetabling competition 2019 problem. 2020.
4. Alexandre Lemos, Pedro T. Monteiro, and Ines Lynce. Itc-2019: A maxsat approach to solve university timetabling problems. 2021.
5. Miranda Lundy and Alistair Mees. Convergence of an annealing algorithm. *Mathematical programming*, 34(1):111–124, 1986.
6. Tomáš Müller, Hana Rudová, Zuzana Müllerová, et al. University course timetabling and international timetabling competition 2019. In *Proceedings of 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, pages 5–31, 2018.
7. Wolfgang Wenzel and Kay Hamacher. Stochastic tunneling approach for global minimization of complex potential energy landscapes. *Physical Review Letters*, 82(15):3003, 1999.

Table 1 Presentation of the gap (in percentage) from the best known results

Instance name	Best	Holm et al.(%)	Müller (%)	Rappos et al.(%)	Gashi et al.(%)	Er-rhaimini(%)	Lemos et al.(%)
agh-fis-spr17	3039	0	12.2	49.9	123.7	87.8	N/A
agh-ggis-spr17	34285	0	6.2	6.7	127.3	65.5	N/A
bet-fal17	289965	0	0.2	1.8	3.1	N/A	2.1
iku-fal17	18968	0	24.5	41.5	166.8	134.5	57.7
mary-spr17	14910	0	1.4	0.7	6.5	11.9	N/A
muni-fi-spr16	3756	0	7.5	2.3	33.2	38.6	N/A
muni-fsps-spr17	868	0	1.2	1.7	123.2	376.3	N/A
muni-pdf-spr16c	33724	0	18.3	11.1	72.5	130	59.5
pu-llr-spr17	10038	0	7.5	33.3	68.1	91.5	N/A
tg-fal17	4215	0	0	0	90.8	74.5	60.7
agh-ggos-spr17	2864	0	19.3	120.6	225.6	169.7	2684.3
agh-h-spr17	22175	0.05	0	17.9	13.1	16.1	N/A
lums-spr18	95	0	3.1	20	12.6	87.3	475.7
muni-fi-spr17	3825	0	3.6	12.1	22.6	42	372.6
muni-fsps-spr17c	2596	0	16	27.2	255	806	23714.2
muni-pdf-spr16	17208	0	16.8	41.3	132.8	125.6	1707.2
nbi-spr18	18014	0	3.7	5.7	47.2	68.2	177.1
pu-d5-spr17	15204	4.6	0	23.7	27.8	33.1	3.4
pu-proj-fal19	117425	25.7	0	377.9	102.6	49.9	9.32
yach-fal17	1074	15.3	0	71.6	60.8	196.1	N/A
agh-fal17	118038	24.5	0	N/A	20.8	55.9	29.8
bet-spr18	348524	0	0.2	3.3	1.5	3.4	7
iku-spr18	25868	0	39.1	41.9	76	232.3	174.2
lums-fal17	349	0	5.7	10.6	132.9	39.2	59.8
mary-fal18	4422	0	8.8	27.5	897.2	62.8	57
muni-fi-fal17	2999	0	8.2	26.5	38.7	57.1	60.7
muni-fspsx-fal17	10123	68.3	0	226	900.8	314.2	933.5
muni-pdfx-fal17	98373	13.7	0	53.9	53.9	61.8	95.1
pu-d9-fal19	39942	0	11.9	235.5	19	107.2	76.3
tg-spr18	12704	0	14.5	1.2	151.1	25.8	55