Noname manuscript No.
(will be inserted by the editor)

Optimizing Educational Schedules Using Hungarian Algorithm and Iterated Local Search

Peter Karich

Received: date / Accepted: date

Abstract In this paper a technique called No-Collision-Principle will be introduced to avoid hard-constraint violations of resources in an educational schedule by construction. The technique is a heuristic approach which is based on an iterated local search and uses matching algorithms from graph theory such as the Hungarian algorithm to assign events to locations very efficiently. The developed solver is compared to the constraint based solver of the UniTime project and is applied to data of the University of Bayreuth (Germany).

Keywords Hungarian Algorithm · Iterated Local Search · Timetabling · No-Collision-Principle

Introduction

Course timetabling problems are well studied and a good example of the problem can be seen in track two and three of the International Timetabling Competition [McCollum(2007)]. The success of constraint based techniques in the competition was clearly shown by the UniTime solver [Müller(2005), Müller(2007)], which won track one and track three.

In this paper a different heuristic approach is introduced to solve such course timetabling problems. It does not utilize constraint based techniques like the UniTime solver, but it is especially fast in reducing conflicts of hard constraints, because it uses the well studied Hungarian algorithm [Clark and Holton(1994), Kuhn(1955), Munkres(1957)] to assign events to locations. It does this through using iterated local search to improve the solution [Stützle et al(2003)Stützle, Lorenco, and Martin].

The Hungarian algorithm is also known as Kuhn-Munkres algorithm and solves the assignment problem in polynomial time $O(n^3)$ [Edmonds and Karp(1972)]. This complexity makes the application in NP-complete problems like timetabling very suitable. In the current paper n is the number of locations available for an event in one time-slot. The assignment algorithm is used to assign one event to one location for one time-slot. For

Peter Karich, www.timefinder.de
Pannous GmbH, Hagenkampsweg 10, 25474 Hasloh, Germany
E-mail: pkarich@pannous.de

every time-slot a quadratic[1] assignment matrix H is created, which stores the weights of all possible assignments. The weights are calculated from required features of an event (e.g. use only labs for chemistry) and from the fact that seat utilization should not be wasted (i.e. schedule "small events into only small locations"). After this H is used from the assignment algorithm, which calculates a perfect matching in case of an exact algorithm. See [Edmonds(1965),Gabow(1990)] for a list of assignment algorithms which can be used in weighted general graphs and [Gabow et al(1989)Gabow, Galil, and Spencer, Goldfarb(1985)] for algorithms which are optimized for weighted bipartite graphs.

There are approximative algorithms which solve the weighted matching problem very fast and at the same time produce very good results, although these results are not optimal. These algorithms could be better suited as subroutines in an heuristic where an exact solution is not always necessary. For that reason an approximative assignment algorithms with a complexity of $O(n^2)$ and with a performance ratio $c = 1/2$ is used too and is compared to the algorithm performance of the exact assignment algorithm. The performance ratio $c = 1/2$ for maximal weighted matching defines that the weight is at least half times the weight of an optimal solution for all graphs. For example the maximal sum of the weights is W for the exact algorithm, then the approximative algorithm with c will calculate maximal weights with at least $c \cdot W$. More approximative assignment algorithms are listed in the paper of Pettie et al. [Pettie and Sanders(2004)].

To our knowledge only one paper utilizes an assignment algorithm in educational scheduling [Bahurmoz(2004)]. The author of this paper uses the Hungarian algorithm to assign events into time-slots. This approach uses a modified Hungarian algorithm as the base for the timetabling process and in every iteration of the Hungarian algorithm a check of other constraints is applied. This approach is not studied here, because then other assignment algorithms couldn't easily be used as replacement for the Hungarian algorithm.

The present paper will be structured as follows. In the first section the naming of common data entities will be described. After this in section two the No-Collision-Principle can be formulated and the implementation of this principle in the TimeFinder project will be explained [Karich(2008)]. In section three it will be shown how the algorithm optimizes the data sets of the International Timetabling Competition and how it was applied to data of the University of Bayreuth.

Definitions

In order to describe the algorithm in the next section the following terms need to be defined:

- A common educational timetable has a specific number of days and time-slots per day. Where the conversion to real-world time can be different (e.g. one slot equates to one hour). In this paper a **timetable** will be a sequence of **time-slots** $t \in [0,m)$, where m is the number of time-slots per week.
- An **event** e has a start time-slot s and an end time-slot $t > s$, where the duration $d(e)$ is $t - s$.
- There are **resource** entities like person and location:
  - a **person** p is assigned to a set of events $e(p)$. This is a static set, which is known right from the start of the algorithm. Persons are the event's visitors $p(e)$.

[1] If more locations than events are available then events with invalid weights are introduced to make H quadratic.

  - a **location** l will be assigned to events $e(l)$ during the calculation of the algorithm. Only one event per time-slot can be assigned to a location.
- One **event-conflict** $ec = 1$ occurs, when a resource r has two events e_i and e_j which share one time-slot, i.e. the events overlap. More generally: $ec(r, e_i, e_j) = n, i > j$, where n is the number of shared time-slots. **Conflicting events** are events with common persons.
- A violation of a hard- and soft-constraint will be expressed in this paper as **hard- and soft-conflict**. Whether a constraint is hard or soft depends on the application. E.g. in this paper an event-conflict is handled as a hard-conflict for locations and persons. Other timetabling problems like the exam timetabling exists where locations' event-conflicts are not handled as hard-conflicts.
- The algorithm uses different types of **raster** entities which can be thought of sequences of time-slots:
  - Resources and events can have **bit-rasters**. A bit-raster R_{bit} is a boolean sequence of length m. If the resource (or event) can be assigned to the time-slot t then $R_{\text{bit}}(t) =$ true; otherwise it is false.
  - A **day-raster** $R_{\text{day}}(t)$ enforces that events with $d > 1$ won't be placed on more than one day, but $R_{\text{day}}(t) =$ true for all time-slots t.
  - An **event-raster** R_{ev} has a root-event e_r. All events which have persons in common with e_r will be stored in this R_{ev} to avoid event-conflicts for e_r. If e_i will be assigned all event-rasters of conflicting events has to be updated accordingly. To make sure that the root-events of those event-rasters won't conflict with e_i in later reassignments.
  - A **raster-collection** R_{coll} is a set of merged rasters. In the most cases the merging is a bitwise *or*-operation of the involved rasters.

No-Collision-Principle

The No-Collision-Principle was designed to solve post enrollment based course timetabling problems, like track 2 of the International Timetabling Competition, but it can be applied to curriculum based course timetabling too.

To solve real-world applications the limitation was removed that the event's duration has to be one. This is not a trivial change like it might look at the first glance, because time-slots are then no longer independent from each other, i.e. the complexity of the whole timetabling problem increases.

With the No-Collision-Principle event-conflicts will be avoided by construction. So in contrast to the approach of repairing strategies for other timetabling heuristics, the No-Collision-Principle leads to a set of assigned events without any hard-conflict and a set of unassigned events, which couldn't be assigned without introducing hard-conflicts.

The following algorithm is the TimeFinder-implementation of the No-Collision-Principle:

- *Step 1* "Initialize"
  Put all n events into an array A.
- *Step 2* "Spread Events"
  For every time-slot $t \in [0, m)$ an assignment-manager M_t exists, which uses the Hungarian algorithm to find a free location for event $e_i \in A$.
  If $d(e_i) > 1$ then more than one assignment-manager is involved in the assignment. All time-slots $t' \in [s, s + d)$ has to be conflict-free and the same location must be available for e_i, where s is the start time-slot.

Find such a time-slot s for e_i and then use one location $\forall t'$. If no such time-slot exists then left e_i unassigned and proceed with e_{i+1}. I.e. continue with step 2 or go to step 3, if e_i is the last event in A.

- *Step 3* "Compression"
  Try to place events which couldn't be assigned in the previous step and iterate through all possible time-slots t' of every event e_i. In *Step 2* an event isn't assigned if event-conflicts would have been introduced. Now assign event e_i to t' but remove the conflicting events of e_i before. The set of these conflicting events is calculated as $C(e_i,t')\ \forall t' \in [s, s + d(e_i))$. Then try to assign every event of $C(e_i,t')$ into other time-slots. This will be done recursively until all events in C could be assigned. The recursion will go until a maximal depth (e.g. 4) is reached. If this moving fails, then the changes will be rolled back.
- *Step 4* "Select Best"
  Choose the current solution as the new local optimum, if it has less unassigned events u then this is the best solution determined so far or - if u is zero - additionally compare the soft-constraints. Stop if there are no soft-constraints.
- *Step 5* "Prepare Next Iteration"
  Unassign a certain number of events to prepare the next iteration. For that a constant $\beta, \beta \leq 1$ is calculated to unassign $\beta \cdot n$ events and move them back to A. β depends on previous iterations and on the current solution's hard- and soft-conflicts. The initial value of β is 0.2. Go to step 2, if enough time is available; otherwise stop.

The calculation of β in the last step is according to the iterated local search to avoid that the solution for the timetable will be trapped in local minima: If the current solution's hard-conflicts are less than the best solution, then decrease β so that in the next iteration less events $n' < n$ will be rescheduled. If the current solution's hard-conflicts are only less than the last solution then the decrease is smaller. Otherwise increase β slightly. The interesting point about this simple implementation of the iterated local search is that it leads to a relative strong optimization compared to all other, more complicate local search methods that were applied to the same instances.

Application

TimeFinder is an Open Source project started in 2008. It offers a simple graphical user interface and the optimization algorithm presented in the previous section. It is simple to use the existing timetabling infrastructure like importing/exporting files, constraint checking methods, editing data or presenting data for own projects.

The TimeFinder-implementation was applied to all data sets of the International Timetabling Competition (track 2). The hard-conflicts for all data sets can be removed in under 20 seconds with a Intel Core 2 Duo CPU P8600 2.40GHz (except file 10), where only one CPU is used according to the requirements of the competition. For a comparison of TimeFinder to the constraint based solver of UniTime after an optimization of 345 seconds for the instances of the International Timetabling Competition (track 2) see table 1.

UniTime's solver clearly shows its strengths in soft-conflict reduction, where TimeFinder's solver outperforms UniTime in decreasing only the hard-conflicts of an order of magnitude. It can be concluded that the TimeFinder-implementation of the No-Collision-Principle is a powerful tool to optimize timetables very fast especially for "complicated" timetables, where the hard-conflicts are not easy to reduce.

Experiments with approximative assignment algorithm have been made to find an algorithm which has a better complexity than $O(n^3)$. The "path growing algorithm" [Drake

File	A) soft-conflicts (a.u.) UniTime	TimeFinder	B) time (sec) UniTime	TimeFinder
1	1517-∅	1903	144	**9**
2	∅	**1848**	450	**17**
3	**377-859**	1817	12	<**0.1**
4	**750-932**	1627	15	<**0.1**
5	**1-140**	1103	9	**1**
6	**110-268**	1112	7	<**0.1**
7	**80-496**	828	5	<**0.1**
8	**33-299**	699	4	<**0.1**
9	∅	**2230**	377	**16**
10	∅	**2093**	910	**53**
11	**648-845**	1690	14	<**0.1**
12	962-∅	1659	32	**0.1**
13	**143-579**	1181	15	**1**
14	**144-886**	1134	20	**0.4**
15	**283-324**	785	3	<**0.1**
16	**158-221**	813	2	<**0.1**

Table 1 A) The soft-conflicts are listed for UniTime's and TimeTabler's solvers. Three seeds (9486273, 12312, 321456) give a range of soft-conflicts. The ∅-sign marks solutions where not all hard-conflicts could be removed. **B)** The required time t in seconds is listed to get a solution with no hard-conflicts. For TimeTabler $t_{\mathrm{RMS}} < 1\mathrm{s}\,\forall$ files. For UniTime's solver the best seed was selected.

file	assignment algorithm exact	path growing	simple
1	**9**	13	18
2	**17**	199	42.9
3	<0.1	<0.1	<0.1
4	<0.1	<0.1	<0.1
5	1	0.4	**0.1**
6	<**0.1**	0.3	0.8
7	<0.1	<0.1	<0.1
8	<0.1	<0.1	<0.1
9	**16**	200	79
10	**53**	156	>345
11	<0.1	<0.1	<0.1
12	0.1	<**0.1**	0.1
13	1	3	**0.7**
14	0.4	1	**0.2**
15	<0.1	<0.1	<0.1
16	<0.1	<0.1	<0.1

Table 2 Comparison of different assignment algorithms. The required time in seconds is listed to get a solution for the No-Collision-Principle with no hard-conflicts. The simple approximative algorithm has a better constant in the complexity, but a worse accuracy than the "path growing algorithm".

and Hougardy(2003)] with $O(n^2)$ and another very simple approximative algorithm with $O(n^2)$ [Karich(2008)] have shown that it is more important to have a high accuracy than a fast calculation of the result especially for complicate timetables. See table 2 for a detailed comparison. I.e. the results of a better assignment algorithm in *Step 3* should be as much as possible close to the exact assignment algorithm but faster than this. In section "Outlook" the candidates for an improvement compared to the Hungarian Algorithm are presented.

The data from University of Bayreuth (curriculum based course timetabling problem) can be used to demonstrate that the algorithm is able to solve large problems too. The data set contains more than 1000 events, over 600 persons and over 100 locations. Here again no soft-constraints are taken into account. The hard-conflicts of this data set was reduced to 6 within 2 seconds with the same hardware configuration.

Outlook

At the moment the main disadvantage of the algorithm is that the soft constraint evaluation has not the necessary influence on the "Spread Events"- or "Compression"-step, so that soft-conflicts won't necessary be avoided in the next iteration. This will be addressed in a later implementation.

Another improvement could be made with the incremental assignment algorithm, which would not recalculate the assignment every time an event is added to the assignment-manager - it will use the existing augmenting tree from previous calculations [Toroslua(2007)].

Other approximative algorithms should be tried, too. For example the approximative algorithm of Pettie et al. [Pettie and Sanders(2004)] with a higher performance ratio should be studied then.

It would be interesting to apply Hungarian algorithm to the assignment of events to time-slots. Which is similar to the approach of Bahurmoz [Bahurmoz(2004)], but could use the dynamic assignment algorithm developed by Mills-Tettey et al. [Mills-Tettey et al(2007)Mills-Tettey, Stentz, and Dias].

Acknowledgements I would like to thank my family - TimeFinder wouldn't be possible without them. Thanks to Mr. Flügge from Pannous for the valuable discussion. Further thanks goes to Prof. Jablonski for the possibility to apply TimeFinder's algorithm on the data of the University of Bayreuth.

References

[Bahurmoz(2004)] Bahurmoz AMA (2004) A hungarian based algorithm for the academic scheduling problem. In: Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling

[Clark and Holton(1994)] Clark J, Holton D (1994) Graphentheorie - Grundlagen und Anwendungen. Spektrum - Akademischer Verlag

[Drake and Hougardy(2003)] Drake, Hougardy (2003) A simple approximation algorithm for the weighted matching problem. Information Processing Letters 85(4):211–213

[Edmonds(1965)] Edmonds J (1965) Maximum matching and a polyhedron with $0,1$ vertices. Journal of Research of the National Bureau of Standards 69 B:125–130

[Edmonds and Karp(1972)] Edmonds J, Karp RM (1972) Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the Association for Computing Machinery 19(2):248–264

[Gabow(1990)] Gabow HN (1990) Data structures for weighted matching and nearest common ancestors with linking. In: SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp 434–443

[Gabow et al(1989)Gabow, Galil, and Spencer] Gabow HN, Galil Z, Spencer TH (1989) Efficient implementation of graph algorithms using contraction. Journal of the Association for Computing Machinery 36(3):540–572, DOI http://doi.acm.org/10.1145/65950.65954

[Goldfarb(1985)] Goldfarb D (1985) Efficient dual simplex algorithms for the assignment problem. Mathematical Programming 33(2):187–203

[Karich(2008)] Karich P (2008) TimeFinder.de project. URL http://timefinder.sourceforge.net/

[Kuhn(1955)] Kuhn HW (1955) The hungarian method for the assignment problem. Naval Research Logistics Quarterly 2:83–97

[McCollum(2007)] McCollum B (2007) The second international timetabling competition 2007/2008

[Mills-Tettey et al(2007)Mills-Tettey, Stentz, and Dias] Mills-Tettey GA, Stentz AT, Dias MB (2007) The dynamic hungarian algorithm for the assignment problem with changing costs. Tech. Rep. CMU-RI-TR-07-27, Robotics Institute, Pittsburgh, PA

[Müller(2005)] Müller T (2005) Constraint-based timetabling. PhD thesis, Charles University

[Müller(2007)] Müller T (2007) UniTime.org project. URL http://www.unitime.org/

[Munkres(1957)] Munkres J (1957) Algorithms for the assignment and transportation problems. Journal of the Society of Industrial and Applied Mathematics 5(1):32–38

[Pettie and Sanders(2004)] Pettie S, Sanders P (2004) A simpler linear time 2/3 ϵ approximation for maximum weight matching. Information Processing Letters 91(6):271–276

[Stützle et al(2003)Stützle, Lorenco, and Martin] Stützle T, Lorenco HR, Martin O (2003) Handbook of metaheuristics, International Series in Operations Research & Management Science, vol 57, Springer New York, chap Iterated Local Search, pp 320–353

[Toroslua(2007)] Toroslua I (2007) Incremental assignment algorithm. Information Sciences 177(6):1523–1529