Replication of a SAT Based Scheduler for Tournament Schedules

Martin Mariusz Lester[0000−1111−2222−3333]

University of Reading, United Kingdom m.lester@reading.ac.uk

Abstract. Automated sports timetabling and scheduling is an established field within operations research. The dominant technologies are local search and commercial Mixed Integer Programming (MIP) solvers, but Boolean Satisfiability (SAT) solvers are also viable.
In 2004, Zhang and others developed a SAT-based tool for generating single, double and partial double round robin tournament timetables satisfying a range of common constraints. The tool was available through a Web interface that ran the solver on a Web server, but this is currently offline. The original paper only gave an overview of the problem encoding. In this replication study, we construct a similar system, mostly following the original encoding, while filling in details as necessary.
The original system used the *Sato* SAT solver, which implemented an extension to the DIMACS CNF format for *true-literal* clauses. These have now been largely superseded by more general *psuedo-Boolean* (PB) clauses; our system encodes the problem using the standardised PB format and solves it with the PB solver *RoundingSat*.

Keywords: SAT solvers · pseudo-Boolean solvers · sports timetabling · replication.

1 Introduction

This replication study focuses on a sports timetabling tool described in the paper *A SAT Based Scheduler for Tournament Schedules* by Zhang and others from 2004 [13]. The tool generated single, double and partial double round robin tournaments timetables satisfying a range of common constraints. It used an encoding of the timetabling problem as a SAT instance, extended with *true-literal* clauses, which could be solved with Zhang's own SAT solver, *Sato* [12]. The tool was available for public use through a Web interface, but the website is no longer accessible.

Our reimplementation of the problem encoding is written in JavaScript[1]. We use the pseudo-Boolean solver RoundingSat to solve problem instances.

We give some background on sports timetabling in Section 2, then give full details of the constraints in our replication of the tool in Section 3. Section 4 presents some simple empirical results, which demonstrate that the encoding

[1] To appear here: https://doi.org/10.5281/zenodo.7790943

does indeed work; we also discuss some claims made in the original paper about the necessity of true-literal clauses. Finally, we reflect on how to make our implementation available for long-term public use in Section 5.

We attempt to verify and replicate the following claims in the original paper:

- *The encoding describes tournament timetables satisfying a range of constraints.* Replicated. We were able to generate timetables using the same encoding, once we had completed some omitted details.
- *Problem instances using the encoding can be solved quickly with a solver that supports true-literal clauses.* Replicated. The paper only gives rough timings for one instance; we solve it in less than a second with the PB solver RoundingSat.
- *Dedicated support for true-literal clauses is necessary for the encoding to be effective.* Partially disputed. With a good choice of encoding, the clauses can be translated into pure SAT and solved without dedicated support. We show this using MiniSat+.

2 Background

Sports timetabling is an area within automated timetabling and scheduling that considers how best to generate timetables for sports tournaments. Most research considers sports or games for two teams or players. One of the most common kinds of timetable considered is the *round robin*. In a single round robin, every team must play against every other team exactly once, while in a double round robin, every pair of teams must play twice, once at the first team's "home" stadium or venue and once at the second team's. Often, the timetable is *time-constrained*, meaning that it is arranged as a sequence of days, rounds or slots, with each team playing one game in each day.

While early research on sports timetabling focused mainly on mathematical design theory, automated timetabling currently uses two main techniques: Mixed Integer Programming (MIP) and local search with domain-specific heuristics. See Rasmussen and Trick [10] for a survey of the area.

In practice, timetables often have to satisfy a range of other constraints. Teams often perform worse if they playing many consecutive away games (an *away break*). Many other practical, economic and social issues, such as availability of stadiums, policing of fans and television schedules, may introduce further constraints. Recently, Van Bulck and others [1] observed that most research focuses on either abstract mathematical problems or a single real-world problem and introduced a standardised format called *RobinX* for specifying real-world problems and encouraging the development of general-purpose solvers [11].

Compared with MIP and local search, SAT has received little attention in sports timetabling, with the work of Zhang and others [13] being a notable exception. Other work on SAT and timetabling includes that of Horbach and others [4] and our own [6, 7].

In this context, the paper we seek to replicate is interesting for two main reasons. Firstly, it was accompanied by an implementation of a general-purpose

solver available through a Web interface, which allowed a user to choose from a small range of practically-motivated constraints. Secondly, it considers the use of a SAT-based solver that supports an extended input format, which allows for *true-literal* clauses, which allow one to specify that a certain number of literals in a clause must be true. At the time, several researchers were investigating this, which led to the standardisation of the pseudo-Boolean input format (with pseudo-Boolean clauses generalising true-literal clauses) and the first pseudo-Boolean solver competition in 2005 [8].

3 Problem Encoding

We now present our replication of the problem encoding used by Zhang and others [13]. For a tournament involving n teams with games played over m days, the encoding uses Boolean decision variables $p_{t_1,t_2,d}$, where $t_1, t_2 \in [1, n]$ are teams and $d \in [1, m]$ is a day. (All quantification of t_1, t_2 and d is implicitly over these ranges, unless otherwise specified.) $p_{t_1,t_2,d}$ is true exactly when team t_1 plays against team t_2 on day d of the schedule. There are no auxiliary variables. The constraints we implement are as follows.

3.1 Round Robin Constraints

Each team plays exactly once per day:

$$\forall d, t_1. \sum_{t_2} (p_{t_1,t_2,d} + p_{t_2,t_1,d}) = 1$$

Note that the original paper makes the constraint at most once per day, which would suggest a time-relaxed timetable. We have changed this to exactly once per day, which seems more consistent with other parts of the paper, as discussed below.

Teams cannot play against themselves:

$$\forall d, t. p_{t,t,d} = 0$$

Each team can play each other team at most once at home and at most once away:

$$\forall t_1, t_2 \neq t_1. \sum_d p_{t_1,t_2,d} \leq 1$$

Each team must play each other team at least once and no more than twice:

$$\forall t_1, t_2 \neq t_1. \sum_d (p_{t_1,t_2,d} + p_{t_2,t_1,d}) \geq 1$$
$$\forall t_1, t_2 \neq t_1. \sum_d (p_{t_1,t_2,d} + p_{t_2,t_1,d}) \leq 2$$

There is some redundancy here, as we have not specified $t_1 < t_2$, so each fixture occurs in two constraints, but it does not seem to present a problem for the solvers we tried. Note that these constraints allow for either a single, double or partial double round robin, depending on the number of days in the timetable. $m = n-1$ gives a single round robin (with the ≤ 2 constraints being redundant), $m = 2(n-1)$ gives a double round robin and anything in between gives a partial double round robin.

3.2 Optional Constraints

The original tool offered a choice of optional constraints, but only give encodings for some of them, so we had to interpret them ourselves, as follows.

No three consecutive home games (encoding in original paper):

$$\forall t_1, d_0 \in [1, m-2]. \sum_{d \in [d_0, d_0+2]} \sum_{t_2} p_{t_1,t_2,d} \leq 2$$

No three consecutive away games (encoding in original paper):

$$\forall t_1, d_0 \in [1, m-2]. \sum_{d \in [d_0, d_0+2]} \sum_{t_2} p_{t_2,t_1,d} \leq 2$$

At least one home game in the first three game days:

$$\forall t_1. \sum_{d \in [1,3]} \sum_{t_2} p_{t_1,t_2,d} \geq 1$$

At least one home game in the last three game days:

$$\forall t_1. \sum_{d \in [m-2,m]} \sum_{t_2} p_{t_1,t_2,d} \geq 1$$

Home and away games are as balanced as possible for each team:

$$\forall t_1. \sum_{d} \sum_{t_2} p_{t_1,t_2,d} \geq \lfloor \tfrac{m}{2} \rfloor$$
$$\forall t_1. \sum_{d} \sum_{t_2} p_{t_2,t_1,d} \geq \lfloor \tfrac{m}{2} \rfloor$$

Home weekday games, home weekend games, away weekday games and away weekend games are as balanced as possible for each team:

$$\forall t_1. \sum_{d \in M} \sum_{t_2} p_{t_1,t_2,d} \geq \lfloor \tfrac{|M|}{2} \rfloor$$
$$\forall t_1. \sum_{d \in M} \sum_{t_2} p_{t_2,t_1,d} \geq \lfloor \tfrac{|M|}{2} \rfloor$$
$$\forall t_1. \sum_{d \in E} \sum_{t_2} p_{t_1,t_2,d} \geq \lfloor \tfrac{|E|}{2} \rfloor$$
$$\forall t_1. \sum_{d \in E} \sum_{t_2} p_{t_2,t_1,d} \geq \lfloor \tfrac{|E|}{2} \rfloor$$

where M is the set of weekdays and $E = \{e_1, e_2, \ldots\}$ is the set of weekend days. This encoding was given in the original paper, but all constraints were of the form $\ldots = k$, without specifying how to calculate k, which seems incorrect if $|M| \neq |E|$ or either $|M|$ or $|E|$ is odd.

No more than three away games in the first five weekends:

$$\forall t_1. \sum_{d \in \{e_1, \ldots, e_5\}} \sum_{t_2} p_{t_2,t_1,gw} \leq 3$$

No two final away games:

$$\forall t_1. \sum_{d \in [m-1,m]} \sum_{t_2} p_{t_2,t_1,d} \leq 1$$

3.3 Discussion of Constraints

Time-constrained timetables. The original paper does not explicitly state whether the implementation is for time-constrained or time-relaxed timetables. In a time-constrained timetable, every team must play in every game day, and there are only as many game days as necessary. A time-relaxed timetable has more game days, but not every team plays in every day. The interface to the original implementation has no way of specifying how many game days there are. It does allow format to be specified (single, double or partial double round robin), as well as number of games for a partial double round robin. This is sufficient to determine the number of game days for a time-constrained timetable only, so we conclude that that the implementation only handles time-constrained timetables.

Redundant variables. The original formulation includes variables for a team playing against itself, with constraints to force these always to be false. This is clearly redundant, but we replicate it in our implementation. In practice, a solver may be able to eliminate it immediately.

Terminology of home and away games. The original paper refers to a games played at the opponent's stadium as *road games*. We refer to these as *away games*, as the phrase *road game* could also refer to a game played when a team is on a *road trip*, meaning that it has a sequence of away games.

Handling of byes. The original paper does not specify whether an odd number of teams is permitted. The webpage invites numbers of teams between 6 and 32, but does not specify whether odd numbers are allowed. In a time-constrained tournament, as each game needs 2 teams, an odd number of teams leads to one team not playing in each round. This is called having a *bye*. A simple way of handling this, which we adopt, is to add a dummy team into the round robin schedule. When another team plays against this team, it has a bye. It is not clear whether a bye should be counted as a home or away game for constraints on home and away games. A constraint like "at least one home game in the first three days" might literally mean that a team must play at home, so as to attract an audience at the start of the season, or it might mean that a team should not have to play three consecutive away games, as this would be a disadvantageous "away break". In the former case, playing two away games and one bye would not be acceptable, but in the latter case, it might.

Games per week and weekdays. The original implementation includes an option for the number of games per week, which is shown with value 2. The number of game days within a week is significant for some of the constraints, which distinguish between weekday games and weekend games. The encoding in the paper suggests that odd-numbered days should be weekdays, while even-numbered days should be weekends. This is fine for 2 games in a week, but does not really make sense for higher numbers, as it interleaves weekdays with weekend days, and may lead to more than 2 weekend days. So instead, we make the decision to have one weekend day per week, which is always the last day of the week.

Unclear constraint. The paper mentions the optional constraint that "weekday vs. weekend games are as balanced as possible for each team". We were unable to work out what this meant, so we did not implement it. In a time-

relaxed timetable, where teams might not play on every day, the constraint is meaningful, but in a time-constrained timetable it is not, and as discussed earlier, we concluded that the original implementation was for a time-constrained timetable. Our best consistent guess is that, in a time-constrained partial double round robin with an odd number of teams, one might wish to avoid both byes being at the weekend.

Repeated second half of timetable. The screenshot in the original paper shows a double round robin timetable where, in the second half of the timetable, teams meet in the same order as in the first half. It is unlikely this would have occurred by chance. We suspect that the original tool had extra constraints to enforce this, but there is no mention of it in the text, so we have chosen not to implement it.

4 Results

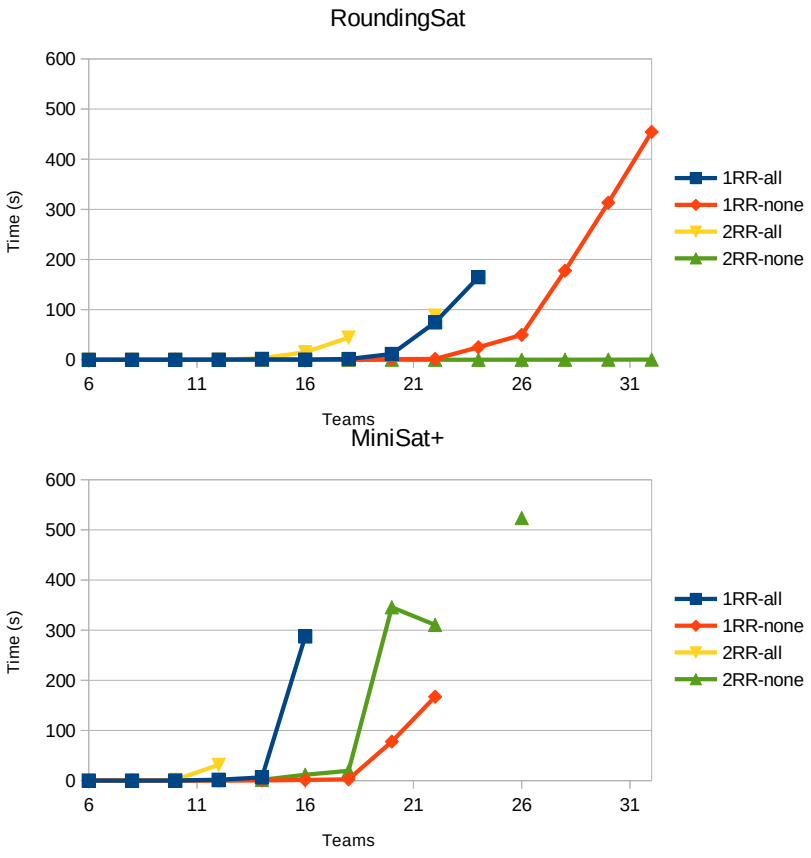

Fig. 1. Solver CPU times for different numbers of teams and encodings. Broken line indicates timeout. All solved instances are satisfiable except 1RR-all for 6 teams.

Following the suggested numbers of teams in the original tool, we tested our implementation with all numbers of teams from 6 to 32. We tested instances with all optional constraints turned off (*none*), and with all optional constraints turned on (*all*), with 2 games per week, for both single (*1RR*) and double (*2RR*) round robins.

Figure 1 shows our results. All times are for a single core of an Intel i5-7500 CPU @ 3.40 GHz, with a timeout of 600 s, on Debian Linux 12. We used RoundingSat commit b5de84d (February 2022) [3] and MiniSat+ 1.1 [2] as solvers. RoundingSat represents the current state-of-the-art in PB solving, while an earlier version of MiniSat+ ranked highly in the first PB Evaluation in 2005 [8], so is roughly contemporary with the original paper.

The original paper does not give detailed information on timings or feasibility, but does say that, for 12 teams in a double round robin with all constraints turned on, generating a schedule takes only a few seconds. For this instance, on our hardware, RoundingSat took 0.1 s, while MiniSat+ took 31.4 s. Our replication is therefore roughly consistent with this specific result. More generally, our results replicate the main claimed result, that the encoding is effective and instances can be solved quickly with a solver that supports true-literal clauses. We also observe that double round robin instances seem to be easier than single round robin instances, and that instances become significantly harder to solve above 18 teams.

One might intuitively assume that turning on all optional constraints leads to the hardest instances, but in practice, extra constraints can sometimes make an instance easier, for example when trying to minimise the total number of breaks [4]. We did not investigate whether this was the case here.

Zhang and others discuss how a SAT solver with support for true-literal clauses was necessary for their implementation to be effective. However, there are two underlying decisions in their encoding, and their claim may not hold if either is changed. Firstly, central to their claim is the observation that the straightforward encoding of a true-literal clause in pure SAT may need exponentially many clauses. But this is actually a decision about a rejected design; we now know several more efficient encodings [9]. Indeed, MiniSat+ works by encoding a PB instance as a SAT instance and solving it with MiniSat.

Secondly, they do not use any auxiliary variables to track whether a team plays home or away. Many of the optional constraints involve the number and timing of a team's home/away games. Using auxiliary variables, the number of literals needed to check for a team playing a home or away game is reduced from around n to 1. Consequently, even with the exponential encoding of true-literal clauses, all the optional constraints would be relatively tractable except for those involving balancing numbers of home and away games.

5 Deployment

One simple way of making a tool available for easy use and evaluation is to write a CGI script that reads input from an environment variable, runs the tool on

the input with a suitable output, reads the output from the tool, then generates an HTML page with the output dumped it into a `textarea` element. The CGI script can be called as the handler for a static HTML form. This is fine in the short term, provided the number of users is small, so the Web server hosting the tool is not overloaded.

However, it is likely that the server hosting the tool eventually be retired, as has happened to some of our older work [5]. Or perhaps a library upgrade will stop the tool from working without maintenance and recompilation. Whatever the reason, tools deployed in this way through a Web interface often become unusable after a few years. This makes the approach unsuitable for long-term accessibility and reproducibility of research.

Many Computer Science conferences avoid this problem through an artifact submission and evaluation process, which requires that an artifact consisting of the source and/or object code for a tool be archived on a platform such as Zenodo. The artifact usually has an accompanying link to a virtual machine or Docker image, on which the tool can be run. This solves the problems of accessibility and reproducibility, but is relatively heavyweight. A fair amount of time and effort is needed to gain access to and test the tool, compared with the simple Web interface.

We suggest that a longer-lasting Web interface may be built and deployed using modern client-side Web technologies, such as JavaScript and WebAssembly. To this end, our implementation will be made available through a webpage, with the problem encoding written in human-readable JavaScript and the solver compiled to WebAssembly. Although Web standards do change, sometimes in ways that break backwards compatibility, we expect that browsers capable of viewing the page and running the solver will be readily available for the foreseeable future.

6 Conclusion

Our replication of the SAT-based timetabling tool was largely successful and confirms that the original approach works. However, one important timetabling constraint not considered in the original paper, which we are aware can present a problem for SAT-based approaches, is minimisation of the number of breaks (consecutive home games or consecutive away games) [4].

It took a little effort to complete the omitted details from the original paper in a consistent and meaningful way, but as we had already worked on application of pseudo-Boolean solvers to sports timetabling [7], it was not overly burdensome.

Drawing on our knowledge of more recent research [9], we can see that native support for true-literal or pseudo-Boolean clauses is not strictly necessary in the solver, but that these extended clauses are certainly useful in modelling and encoding the problem. We can also see how the encoding might be improved by use of auxiliary variables.

Finally, we encourage researchers in a range of fields to think about how best to make the tools they develop available for easy use beyond the life of

their research groups. We suggest that WebAssembly may be part of a practical solution to this problem.

References

1. Bulck, D.V., Goossens, D.R., Schönberger, J., Guajardo, M.: Robinx: A three-field classification and unified data format for round-robin sports timetabling. Eur. J. Oper. Res. **280**(2), 568–580 (2020). https://doi.org/10.1016/j.ejor.2019.07.023, https://doi.org/10.1016/j.ejor.2019.07.023
2. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into SAT. J. Satisf. Boolean Model. Comput. **2**(1-4), 1–26 (2006). https://doi.org/10.3233/sat190014, https://doi.org/10.3233/sat190014
3. Elffers, J., Nordström, J.: Divide and conquer: Towards faster pseudo-boolean solving. In: Lang, J. (ed.) Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden. pp. 1291–1299. ijcai.org (2018). https://doi.org/10.24963/ijcai.2018/180, https://doi.org/10.24963/ijcai.2018/180
4. Horbach, A., Bartsch, T., Briskorn, D.: Using a SAT-solver to schedule sports leagues. J. Sched. **15**(1), 117–125 (2012). https://doi.org/10.1007/s10951-010-0194-9, https://doi.org/10.1007/s10951-010-0194-9
5. Lester, M., Neatherway, R.P., Ong, C.L., Ramsay, S.J.: Verifying liveness properties of ML programs. CoRR **abs/2012.13333** (2020), https://arxiv.org/abs/2012.13333
6. Lester, M.M.: Scheduling reach mahjong tournaments using pseudoboolean constraints. In: Li, C., Manyà, F. (eds.) Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12831, pp. 349–358. Springer (2021). https://doi.org/10.1007/978-3-030-80223-3_24, https://doi.org/10.1007/978-3-030-80223-3_24
7. Lester, M.M.: Pseudo-boolean optimisation for robinx sports timetabling. J. Sched. **25**(3), 287–299 (2022). https://doi.org/10.1007/s10951-022-00737-7, https://doi.org/10.1007/s10951-022-00737-7
8. Manquinho, V.M., Roussel, O.: The first evaluation of pseudo-boolean solvers (pb'05). J. Satisf. Boolean Model. Comput. **2**(1-4), 103–143 (2006). https://doi.org/10.3233/sat190018, https://doi.org/10.3233/sat190018
9. Philipp, T., Steinke, P.: Pblib - A library for encoding pseudo-boolean constraints into CNF. In: Heule, M., Weaver, S.A. (eds.) Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9340, pp. 9–16. Springer (2015). https://doi.org/10.1007/978-3-319-24318-4_2, https://doi.org/10.1007/978-3-319-24318-4_2
10. Rasmussen, R.V., Trick, M.A.: Round robin scheduling - a survey. Eur. J. Oper. Res. **188**(3), 617–636 (2008). https://doi.org/10.1016/j.ejor.2007.05.046, https://doi.org/10.1016/j.ejor.2007.05.046
11. Van Bulck, D., Goossens, D.: The international timetabling competition on sports timetabling (itc2021). European Journal of Operational Research (2022). https://doi.org/https://doi.org/10.1016/j.ejor.2022.11.046, https://www.sciencedirect.com/science/article/pii/S0377221722009201

12. Zhang, H.: SATO: an efficient propositional prover. In: McCune, W. (ed.) Automated Deduction - CADE-14, 14th International Conference on Automated Deduction, Townsville, North Queensland, Australia, July 13-17, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1249, pp. 272–275. Springer (1997). https://doi.org/10.1007/3-540-63104-6_28, https://doi.org/10.1007/3-540-63104-6_28
13. Zhang, H., Li, D., Shen, H.: A SAT based scheduler for tournament schedules. In: SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings (2004), http://www.satisfiability.org/SAT04/programme/74.pdf