MILP. Try. Repeat.

Computing solutions to the ITC 2021 instances by repeated massive parallel MILP computations

Timo Berthold · Thorsten Koch · Yuji Shinano

Abstract This article describes how we solved most instances of the ITC-2021 as mixed-integer linear optimization problems (MILP). Our goal was to contend in this competition without developing specialized algorithms, i.e., only using existing MILP solvers. This path was challenging but provided feasible solutions for 40 out of 45 instances. Four of these instances where the best solutions overall and two of them were even proven optimal.

First, we will present how we modeled the problems as MILPs, discussing a few variations. Next, we describe how we combined methods to compute better solutions increasingly by restarting different MILP solvers and running the distributed massively parallel ParaXpress solver on HPC computers. Additionally, we computed a particular objective function based on the analytic center and either used this directly or with a newly developed variant of the feasibility pump heuristic. In the end, we also added a simulating annealing heuristic from the literature.

Keywords ITC2021 · MILP · Timetabling · Sports Scheduling · Massive parallel · Analytic Center

Mathematics Subject Classification (2010) 90C09 · 90-08 · 68R05

Timo Berthold
Fair Isaac Germany GmbH
Stubenwald-Allee 19, 64625 Bensheim, Germany
E-mail: timo.berthold@fico.com, OrcID: 0000-0002-6320-8154

Thorsten Koch
TU Berlin, Chair of Software and Algorithms for Discrete Optimization
Str. des 17. Juni 135, 10623 Berlin, Germany
E-mail: koch@zib.de, OrcID: 0000-0002-1967-0077

Yuji Shinano
Applied Algorithmic Intelligence Methods department, Zuse Institute Berlin
Takustr.7, 14195 Berlin
E-mail: shinano@zib.de, OrcID: 0000-0002-2902-882X

1 Introduction

Our goal was to participate in the International Timetabling Competition on Sports Timetabling ITC-2021[1] without developing specialized algorithms, i.e., only by using existing software and mainly working on the modeling and combining existing MILP solving approaches. In the following, we will describe how we came up with a MILP-based approach that made us reach the finals without special-purpose algorithms.

The challenge in ITC-2021 is to construct time-constrained double round-robin tournament with 16 to 20 teams for 45 different scenarios. There are hard constraints, which have to be respected, and soft constraints, which might be violated, but whose violation will result in a penalty. The goal is to minimize the penalties, which makes it a classical optimization problem. For details, see [19].

Section 2 will present the MILP formulation we used and its variants. In Section 3 we will describe our basic approach of repeated restarts. Then, as some of the instances – expectedly – turned out to be difficult, we looked into existing approaches for finding feasible MILP solutions and combined them in a new manner: The analytic center objective and the feasibility pump heuristics, described in Section 4. One of our original ideas was to model the problem instances as MILPs and then solve them on a supercomputer. We sketch the setup and outcome in Section 5. Regarding the general state-of-the art in sports timetabling, we refer to [12,13].

2 The MILP model

We modelled the problem as a mixed-integer linear program (MILP) as follows: First we define the set of teams: $T := \{0, \dots, \text{teams} - 1\}$, the set of slots $S := \{0, \dots, \text{slots} - 1\}$, and $S_0 := S \setminus \{0\}$, and $S_9 := S \setminus \{\text{slots} - 1\}$. The sets of slots of the 1st and 2nd half of the season are represented as : $S_1 := \{0, \dots, \text{slots}/2 - 1\}$, $S_2 := S \setminus S_1$. Correspondingly, the sets of matches are called $M := \{(i, j) \mid i, j \in T, i \neq j\}$. We introduce binary variables x_{ijs} with $(i, j) \in M$ and $s \in S$. $x_{ijs} = 1$ indicating whether team i is playing home against team j away during slot s.

The following constraints ensure the basic requirements: each match gets assigned exactly one slot: $\sum_{s \in S} x_{ijs} = 1$ for all $i, j \in M$; in each slot the number of matches equals half the number of teams: $\sum_{i,j \in M} x_{ijs} = 1/2|T|$ for all $s \in S$; and each team only plays once in each slot: $\sum_{t,j \in M} x_{tjs} + \sum_{i,t \in M} x_{its} = 1$ for all $s \in S, t \in T$. We also added the (redundant) equation $\sum_{s \in S,(i,j) \in M} x_{ijs} = 1/2|S||T|$. Furthermore, in case we have a phased tournament, each pair of teams can only play once per half season: $\sum_{s \in S_1}(x_{ijs} + x_{jis}) = 1$ for all $(i, j) \in M, i < j$.

Now we introduce binary variables b^h_{ts}, and b^a_{ts}, $t \in T$, $s \in S_0$ indicating whether team t has a home or away break during slot s, respectively. Addition-

[1] `https://www.sportscheduling.ugent.be/ITC2021`

ally, we need the following constraints: $\sum_{a\in T\setminus\{t\}} x_{ta,s-1} + \sum_{a\in T\setminus\{t\}} x_{ta,s} \leq 1 + b^h_{ts}$ for all $t \in T, s \in S_0$ and similarly for b^a_{ts}.

Next, we introduce inequalities to enforce the different types of constraints that can appear in the instances. Please see [19] for the parameters of the particular constraints: `min, max, intp` $\in \mathbb{N}$, `teams` $\subseteq T$, `slots` $\subseteq S$, `meetings` $\subseteq T^2$. For CA2, CA3, CA4, and BR1 we only show the `mode="HA"` case, for the two cases the respective x-variables have to be removed accordingly. Note that the violation counter $v_* \geq 0$ has to be fixed to zero in case of a hard constraint. This gives rise to the following constraints:

- CA1(`max`): $\sum_{n\in T\setminus\{t\}} x_{tns} \leq \texttt{max} + v^{\text{ca1}}$ for all $t \in \texttt{teams}, s \in \texttt{slots}$ for `mode="H"`, and x_{nts} in case `mode="A"`.
- CA1(`max`): $\sum_{n\in T\setminus\{t\}} x_{tns} \leq \texttt{max} + v^{\text{ca1}}$ for all $t \in \texttt{teams}, s \in \texttt{slots}$ for `mode="H"`, and x_{nts} in case `mode="A"`.
- CA2(`max,teams`): $\sum_{s\in\texttt{slots}} \sum_{n\in\texttt{teams2}} (x_{tns} + x_{nts}) \leq \texttt{max} + v^{\text{ca2}}$ for all $t \in$ `teams`.
- CA3(`max,intp,teams`): $\sum_{n\in\texttt{teams2}} \sum_{r\in\{s-\texttt{intp}+1,\ldots,s\}} (x_{tnr} + x_{ntr}) \leq \texttt{max} + v^{\text{ca3}}_s$ for all $t \in \texttt{teams}, s \in S \setminus \{0,\ldots,\texttt{intp}-2\}$.
- CA4(`max,slots,teams1,teams2`): $\sum_{(i,j)\in\texttt{teams1}\times\texttt{teams2},i\neq j} (x_{ijs} + x_{jis}) \leq \texttt{max} + v^{\text{ca4}}_s$ for all $s \in \texttt{slots}$.
- GA1(`min,max,slots,meetings`): $\texttt{min} + v^{\text{ga1}}_n \leq \sum_{s\in\texttt{slots}} \sum_{(i,j)\in\texttt{meet}} x_{ijs} \leq \texttt{max} + v^{\text{ga1}}$.
- BR1(`max,slots,teams`): $\sum_{s\in\texttt{slots}} (b^h_{ts} + b^a_{ts}) \leq \texttt{max} + v^{\text{ba1}}$ for all $t \in \texttt{teams}$.
- BR2(`max,slots,teams`): $\sum_{s\in\texttt{slots}} \sum_{t\in\texttt{teams}} (b^h_{ts} + b^a_{ts}) \leq \texttt{max} + v^{\text{ba2}}$.
- FA2(`intp,slots,teams`): $\sum_{a,p\in T\setminus\{t\}} \sum_{p\in\{0,\ldots,s\}} x_{tap} = h_{ts}$ for all $t \in T$, $s \in S$; $h_{is} - h_{js} \leq \texttt{intp} + v^{\text{fa2}}_{ij}$ and $h_{js} - h_{is} \leq \texttt{intp} + v^{\text{fa2}}_{ij}$ for all $s \in \texttt{slots}, (i,j) \in \texttt{teams}^2, i < j$.
- SE1(`min,teams`): $|\sum_{s\in S}((s+1)x_{ijs} - (s+1)x_{jis})| \geq 1 + \texttt{min} + v^{\text{se1}}_{ij}$ for all $(i,j) \in \texttt{teams}^2, i < j$.

We wrote a python program to convert the XML description of the ITC-2021 instances into the above model formulated in the modeling language Zimpl [9], which then can generate an LP or MPS file. Zimpl automatically reformulates the absolute value needed for SE1 into an integer linear formulation.

As an objective, we minimize the sum over all violation counters multiplied with their respective penalty values. This objective function is precisely the one computed by the validator of the ITC-2021.

Variations: We experimented (to a minimal extent) with some variations of the above model. We added an objective cutoff with the objective value of the best known solution so far to improve the location of the analytic center as described in Section 4. As far as we can tell, it didn't hurt.

We also experimented with adding parts or even all of the odd-set constraints to push up the lower bound or increase feasibility. Apart from the

LP solves getting slow due to the huge number of constraints, there was no observable effect. Hence we discarded this approach quickly.

Another modeling variant we explored was to remove all soft constraints to get a feasibility instance. We assumed that this might be useful to find initial solutions, as due to the reduced number of constraints, we expected the problems to solve faster. Unfortunately, the effect was not as pronounced as we hoped for. We were only able to generate few initial solutions this way.

Furthermore, we also tried relaxing feasibility by changing the basic constraints from *equal to one* to *less equal to one* and penalized a non-sufficient number of variables set to one in the objective. Also CPLEX [8] and Xpress [5] offer such a procedure out-of-the-box, it is available via the `feasopt` and `repairinfeas` command, respectively. This approach did not provide additional solutions.

We believe the last two ideas had little effect for the following reasons: Most heuristics in a MILP solver are guided by the current LP solution. The LP solution depends on the search tree, and the search tree is built depending on the objective function. We observed that it was most effective to restart the solution process frequently. Apparently, the objective was not guiding the tree search towards better feasible solutions. Removing the objective as in the two experiments above left the solver completely clueless in what direction to go. Since the instances have comparatively few feasible solutions, hitting one by chance proved to be unlikely.

Additionally, we observed that increasing the running time of an instance only had a limited effect, i.e., only minor local improvements were happening, and then the primal solution was stuck in a local optimum. A substantial amount of change in the solution would be required for a major improvement beyond this local optimum, i.e., a distant part of the search tree needed to be explored. At the same time, MILP tree search algorithms tend to explore the vicinity of the current search area first and avoid big jumps. Hence, restarting worked reasonably well to overcome this behavior.

3 Repeated restarts and simulated annealing

In an initial step, we attempted to solve the problem formulation as given in Section 2 using standard MILP solvers. We employed various solvers and solver settings. This is for two reasons: Firstly, different solvers have different strengths, so while some models might solve better with one solver, another solver might be superior in another instance. Secondly, we aim at exploiting the effect of performance variability [10].

Using our cluster facility, we ran each instance up to 25 times, using either CPLEX, Gurobi [7], SCIP [1,6] or Xpress as a solver, using a time limit of up to 12 hours. We varied the parameters between the runs, usually increasing the number of heuristic runs, setting an emphasis for feasible solutions, reducing the amount of cutting plane generation, etc. Further, we altered the LP solver used for the initial relaxation. When available, we used a solution polishing mode after a certain time. We also changed the random seed parameter between runs. As an example, a typical setting for Gurobi would be `Seed=131313 TimeLimit=30000 ImproveStartTime=15000 Cuts=0 Method=3 Presolve=2 Heuristics=0.5`

This way, we could gradually improve the best-known solution. Finally, there were a few notoriously hard problems for which we needed better heuristic methods to find a first feasible solution. We discuss this in the next section.

Note that we never used any specific settings for any of the instances. In principle, they all went through the same loop of repeated solving by various different solvers. The amount of loops depended on the solution quality. Similarly, whether we applied heuristics methods to ignite the search depended on whether our initial MILP searches came up with a feasible solution, but not on any problem characteristics.

There were five instances where we failed to find any feasible solution despite all efforts. After the contest, the ITC team provided us a sample solution to one of these problems. With the knowledge of this initial solution, we were immediately able to improve it.

Towards the end, we implemented a basic ad-hoc version of the simulating annealing heuristics as described in [2]. Again, we used the best-known solution as a start. The heuristic was able to improve this solution in several cases. We used the resulting solution as starting solution for further MILP solver runs.

4 Analytic Center Objective and Feasibility Pump

Some of the strongest primal heuristics implemented inside MILP solvers are improvement heuristics and depend on the knowledge of an initial solution for the problem at hand. To a certain extent, this also holds for the main branch-and-bound search.

In many cases, getting any feasible solution for a MILP problem is not too hard; the complexity of the problem primarily lies in finding the optimal solution and, even more, proving its optimality. However, the ITC-2021 instances, as typical for sport scheduling problems, are such that even getting an initial feasible solution can be very challenging.

Therefore, we decided to employ two special, expensive start heuristic procedures. The first one is making use of an Analytic Center objective as described in [4]. It replaces the objective function of a MILP by coefficients that correspond to the analytic center of the polyhedron associated to the MILP. Furthermore, we searched for initial solutions using a Feasibility Pump algorithm, see, e.g. [3]. The variant we employed uses multiple integer reference vectors in the projection step and is described in the thesis of Mexi [11]. Note that both methods are available out-of-the-box for general MILPs. The Feasibility Pump is available in source code on GitHub, while Analytic Center Search can be run via a special setting of Xpress.

Using our MILP solver portfolio and the two described dedicated start heuristics, we could find solutions for almost all ITC instances. The few without a feasible solution and others for which we only saw a low-quality solution were forward to the next step in our problem-solving scheme: massively parallel MILP solving.

5 Massively parallel computations

As a third step, next to running a MILP solver portfolio and employing MILP start heuristics, we utilized massive parallel MILP solving on a supercomputer. Therefore, we used the massively parallel MILP solver *ParaXpress* [17] which has been developed by using *Ubiquity Generator(UG) framework* [14]. It builds on the FICO-Xpress Optimizer. From a massive parallelization point of view, the latest version of ParaXpress has almost the same features as that of ParaSCIP [16].

We ran ParaXpress on two HLRN IV supercomputers, Lisa and Emmy. In HLRN IV, each compute node employs two sockets of an Intel Xeon Platinum Processor 9242 and 362 GB of Memory, and each job uses 128 or 256 nodes (12288 or 24576 cores, respectively). ParaXpress runs a single controller process called LoadController and ten of thousands of solver processes that solve (sub-)MILPs, each running on a single core. We executed each job with a time limit of 12 hours. Jobs could be interrupted. Altogether, we spent more than seven million core hours.

We show the principal procedure in Algorithm 1. Note that initially, the set $\mathcal{I}$ contained selected instances that we thought were most suitable for a supercomputing approach.

More precisely, we conducted a large scale racing [18,15], in which all Solvers in ParaXpress run Xpress with different parameter settings independently, but incumbent solutions found are shared to cut off search trees.

When there is a new incumbent solution for an instance, a new job to restart the solving process with the incumbent solution is created. We submitted almost all jobs with 12,287 Solvers; only a few ran with 24,576 Solvers. We also tried to solve instances that could not find improved solutions with different ways of running ParaXpress, such as a general parallel Branch-and-bound method provided by UG, rather than a racing search. Note that ParaXpress could solve some of the instances to proven optimality.

6 Results and Conclusion

Table 1 shows our final results. For two instances, we were able to prove optimality of the solution. We could show that 36 of the 45 instances need to violate some of the soft constraints, given that the MILP solver proved a positive lower bound for them. Our results are not exhaustive. We mainly stopped due to the deadline. However, the progress noticeably slowed down.

The catch of our approach is that it did not use any special purpose scheduling algorithm. All methodology is general MILP technology, available indepen-

Algorithm 1: Massive parallel computation procedure

```
Data: Let $\mathcal{I}$ be the set of pairs of instances and its incumbent solution $(I, \mathbf{x})$.
Result: Updated $\mathcal{I}$.
1  $NoImprovedSol := \emptyset$. // indicate instances with no improved solutions
2  $SubmittedJob := \emptyset$. // indicate instances for submitted jobs
3  while The due date has not reached do
4      while $\mathcal{I} \neq \emptyset$ do
5          Select $(I, \mathbf{x})$ and $\mathcal{I} := \mathcal{I} \setminus (I, \mathbf{x})$.
6          Submit a job to solve $I$ by ParaXpress with MIP start $\mathbf{x}$
7                  (Method: large scale racing).
8          $SubmittedJob := SubmittedJob \cup (I, \mathbf{x})$.
9      while terminated job exists do
10         Remove $(I, \mathbf{x})$ from $SubmittedJob := SubmittedJob \setminus (I, \mathbf{x})$ .
11         if Improved soluiton $x^*$ was found for $(I, \mathbf{x})$ then
12             $\mathcal{I} := \mathcal{I} \cup (I, \mathbf{x}^*)$
13         else
14             $NoImprovedSol := NoImprovedSol \cup (I, \mathbf{x})$
15     while Enough computing resources are available and $NoImprovedSol \neq \emptyset$ do
16         Select $(I, \mathbf{x})$ and $NoImprovedSol := NoImprovedSol \setminus (I, \mathbf{x})$.
17         Submit a job to solve $I$ by ParaXpress with MIP start $\mathbf{x}$
18                 (Method: large scale parallel Branch-and-Bound).
19         $SubmittedJob := SubmittedJob \cup (I, \mathbf{x})$.
```

Table 1 Submitted results. Bold indicates the objective value is proven to be optimal. Italic indicates a zero lower bound. For all other instances a positive lower bound was proven.

Instance no.	Early	Mid	Late
01	804	—	2068
02	*402*	—	5525
03	1246	9700	3069
04	*764*	**7**	**0**
05	—	413	—
06	5506	2270	1212
07	6881	2991	2525
08	1409	174	1454
09	*122*	*810*	790
10	—	1813	2544
11	6843	3367	*305*
12	*1025*	1538	5669
13	360	*1051*	3877
14	25	*1679*	1433
15	4616	1614	*40*

dent of this work. Our main contribution was modeling the ITC-2021 problems and the way to combine the different MILP-solving methods.

It became apparent that a dedicated scheduling heuristic to find initial solutions would have been a helpful extension since finding an initial solution was causing severe troubles for some of the models. Once we knew an initial solution, the MILP technology could improve it to a satisfactory level in most cases. We conclude that our methodology could probably be well combined with more problem-specific approaches. It would be interesting to see how well such an integrated approach performed in practice.

For our research, the competition showed direct impact, as it was a great test case for a new version of the feasibility pump heuristic [11], which could compute feasible solutions for 33 of the 45 instances. Furthermore, given that the number of cores available on single machines is constantly increasing, it might be worthwhile to incorporate a restart mechanism similar to the one we scripted directly into the solvers. There are certain classes of problems where this seems to work well. In a sense, this is similar to the idea of racing ramp-up, which we employed for the massive parallel computations.

We will make the code to generate the MILP model instances publicly available through the ITC organizers.

Acknowledgements The work for this article has been conducted within the Research Campus MODAL funded by the German Federal Ministry of Education and Research (fund number 05M14ZAM). The work was supported by the National High Performance Computing Center at the Zuse Institute Berlin (NHR@ZIB). We are grateful to the supercomputer staff, especially Matthias Läuter and Tobias Watermann. Thank to the organizers of the ITC 2021 for providing such nice instances and doing a great job.

References

1. Achterberg, T., Berthold, T., Koch, T., Wolter, K.: Constraint integer programming: A new approach to integrate CP and MIP. In: L. Perron, M.A. Trick (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, pp. 6–20. Springer (2008). DOI 10.1007/978-3-540-68155-7_4
2. Anagnostopoulos, A., Michel, L., Hentenryck, P.V., Vergados, Y.: A simulated annealing approach to the traveling tournament problem. Journal of Scheduling **9**, 177–193 (2006). DOI 10.1007/s10951-006-7187-8
3. Berthold, T., Lodi, A., Salvagnin, D.: Ten years of feasibility pump, and counting. EURO Journal on Computational Optimization **7**(1), 1–14 (2019). DOI 10.1007/s13675-018-0109-7
4. Berthold, T., Perregaard, M., Mészáros, C.: Four good reasons to use an interior point solver within a MIP solver. In: N. Kliewer, J.F. Ehmke, R. Borndörfer (eds.) Operations Research Proceedings 2017, pp. 159–164. Springer (2018). DOI 10.1007/978-3-319-89920-6_22
5. FICO: Xpress Optimization. URL `https://www.fico.com/fico-xpress-optimization/docs/latest/overview.html`
6. Gamrath, G., Anderson, D., Bestuzheva, K., Chen, W.K., Eifler, L., Gasse, M., Gemander, P., Gleixner, A., Gottwald, L., Halbig, K., Hendel, G., Hojny, C., Koch, T., Le Bodic, P., Maher, S.J., Matter, F., Miltenberger, M., Mühmer, E., Müller, B., Pfetsch, M.E., Schlösser, F., Serrano, F., Shinano, Y., Tawfik, C., Vigerske, S., Wegscheider, F., Weninger, D., Witzig, J.: The SCIP Optimization Suite 7.0. ZIB-Report 20-10, Zuse Institute Berlin (2020). URL `http://nbn-resolving.de/urn:nbn:de:0297-zib-78023`
7. Gurobi: Optimizer version 9.1. URL `https://www.gurobi.com`
8. IBM: IBM ILOG CPLEX optimization studio 12.1. URL `https://www.ibm.com/products/ilog-cplex-optimization-studio`
9. Koch, T.: Rapid mathematical programming. Ph.D. thesis, Technische Universität Berlin (2004). URL `https://zimpl.zib.de`
10. Lodi, A., Tramontani, A.: Performance variability in mixed-integer programming. In: Theory Driven by Influential Applications, pp. 1–12. INFORMS (2013). DOI 10.1287/educ.2013.0112

11. Mexi, G.: New ideas for the feasibility pump: Using multiple reference vectors. Master's thesis, Technische Universität Berlin (2021). To appear
12. Rasmussen, R.V., Trick, M.A.: Round robin scheduling - a survey. Tech. rep., Carnegie Mellon University (2006). DOI 10.1184/R1/6707867.v1
13. Ribeiro, C.C.: Sports scheduling: Problems and applications. International Transactions in Operational Research **19**(1-2), 201–226 (2012). DOI 10.1111/j.1475-3995.2011.00819.x
14. Shinano, Y.: The ubiquity generator framework: 7 years of progress in parallelizing branch-and-bound. In: N. Kliewer, J.F. Ehmke, R. Borndörfer (eds.) Operations Research Proceedings 2017, pp. 143–149. Springer (2018)
15. Shinano, Y., Achterberg, T., Berthold, T., Heinz, S., Koch, T., Winkler, M.: Solving hard MIPLIB2003 problems with ParaSCIP on supercomputers: An update. In: Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International, pp. 1552–1561 (2014). DOI 10.1109/IPDPSW.2014.174
16. Shinano, Y., Achterberg, T., Berthold, T., Heinz, S., Koch, T., Winkler, M.: Solving open MIP instances with ParaSCIP on supercomputers using up to 80,000 cores. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 770–779. IEEE Computer Society (2016)
17. Shinano, Y., Berthold, T., Heinz, S.: ParaXpress: an experimental extension of the FICO Xpress-Optimizer to solve hard MIPs on supercomputers. Optimization Methods and Software **33**(3), 530–539 (2018). DOI 10.1080/10556788.2018.1428602
18. Shinano, Y., Heinz, S., Vigerske, S., Winkler, M.: FiberSCIP—a shared memory parallelization of SCIP. INFORMS Journal on Computing **30**(1), 11–30 (2018). DOI 10.1287/ijoc.2017.0762
19. Van Bulck, David and Dries Goossens and Jeroen Beliën and Morteza Davari: The fifth international timetabling competition (ITC 2021): Sports timetabling. Proceedings of MathSport International 2021 Conference, MathSport pp. 117–122 (2021)