Brno University of Technology
Faculty of Information Technology

Simulation Tools and Techniques
Project – Traveling Umpire Problem

Dominik Harmim (xharmi00)
xharmi00@stud.fit.vutbr.cz
28th April 2020

1 Introduction

In this project, a selected *operational research* problem is described according to the certain published research paper. This paper, as well as other resources, are stated in Chapter 1.1. A given problem is defined in Chapter 1.2. Approaches of a solution to the problem from the paper are re-implemented and experimental results are finally compared with the results in the paper.

The rest of the project documentation is organised as follows. In Chapter 2 there is presented an approach of the solution of the problem, and there are also described appropriate methods. Re-implementation of the methods is then depicted in Chapter 3. Experimental evaluation is discussed in Chapter 4. Finally, Chapter 5 concludes the documentation and project results.

1.1 Resources

The project is based on the paper [3]. Almost the same paper with slight differences was previously published in [2]. Further description of the problem together with other related publications, detailed results, data sets, implemented solvers and validators, and other resources can be found in [1].

1.2 Traveling Umpire Problem

By [3, 2], the *Traveling Umpire Problem* is defined as follows. It is a *sports scheduling problem* inspired by the real-life needs of the officials of a sports league. It is all about scheduling the *umpires* for tournaments. Each game in a tournament schedule requires an umpire (more precisely the whole umpire crew but this fact can be neglected in this problem). The problem of assigning the umpires to games in the tournament is subject of several *restrictive constraints*. Given these constraints, the primary objective is to *minimise the travel* of the umpires among single venues.

Formally, it is given a *double round-robin tournament* where every team plays against all other teams twice. For n umpires, there are $2n$ teams and $4n - 2$ game slots (rounds), each with n games. So, it is needed to assign one of the n umpires to each of these games. And the total travel of all the umpires should be as low as possible. Furthermore, the following constraints are needed to be satisfied:

1. Every game gets an umpire.
2. Every umpire works exactly one game per slot.
3. Every umpire sees every team at least once at the home venue of that team.
4. No umpire is in the same venue more than once in any $n - d_1$ consecutive slots.
5. No umpire sees a team more than once in any $\frac{n}{2} - d_2$ consecutive slots.

Here, d_1 and d_2 are parameters that represent the level of constraint required. Setting these to 0 leads to the most constrained system. While setting these to n and $\frac{n}{2}$, respectively "turns off" the corresponding constraint.

Finding a *feasible* solution to this problem may be challenging. Thus, it makes it difficult to find a good solution in a reasonable amount of time. The problem is *NP-Hard* and exact solution approaches are inefficient in solving large instances. In [3, 2] there are presented some exact solution approaches. In particular, there is formulated the problem as a *mixed-integer linear program* and as a *constraint program*. For both cases, it is possible to find the *optimum solutions* for small instances. But for larger instances, it takes more than dozens of hours to find feasible solutions. In the following chapters, there are shown approaches and heuristics that help to find good solutions quickly.

2 Greedy Matching Heuristic with Benders' Cuts Guided Neighbourhood Search

In [3, 2] there is considered a hybrid use of *Bender's cuts*, *large neighbourhood search*, and a *greedy matching based heuristic* to find good solutions to the Traveling Umpire Problem. By combining these seemingly unrelated approaches, it can be found good solutions quickly. In the following chapters, there is explained how to generate the Benders' cuts during the execution of a simple greedy heuristic. These cuts enforce feasibility requirements that allow the greedy heuristic to avoid infeasibilities. These Benders' cuts are embedded in a *very large neighbourhood search* method and by searching this neighbourhood, guided by the Benders' cuts, it is obtained a good initial feasible solution which is then further improved by *local search*. Combining this approach with some other heuristic described later leads to very good solutions.

A high level algorithm of a greedy matching heuristic with the Benders' cuts guided neighbourhood search is presented in Algorithm 1. At each time slot, a greedy matching heuristic is used to construct a schedule. When an infeasibility is recognised first a single step backtracking is tried to resolve the infeasibility. If unsuccessful, the Benders' cuts are generated to guide a large neighbourhood search to ensure feasibility and to improve the solution. Single steps of the algorithm are explained in the chapters below.

Algorithm 1: Greedy matching heuristic with Benders' cuts guided neighbourhood search

```
1  Arbitrarily assign umpires to games in 1. slot;
2  for s = 1 to 4n − 2 do
3      Construct a bipartite graph G = (V, E) where
         the partitions are the umpires and the games
         in s;
4      Solve a perfect matching problem on G with
         respect to constraints;
5      if there is no feasible perfect matching then
6          Backtrack to slot s − 1 and pick the
             second best matching;
7          Construct a bipartite graph G = (V, E)
             where the partitions are the umpires and
             the games in s;
8          Solve a perfect matching problem on G
             with respect to constraints;
9      end
10     if there is no feasible perfect matching then
11         Find all the Benders' cuts;
12         Objective ←
             (number of violated cuts) ∗ cost +
             (number of violated constraints) ∗
             cost + (total umpire travel);
13         while at least one of the cuts is violated ||
             at least one of the constraints is violated
             do
14             Do neighbourhood search using
                 2-ump neighbourhood;
15             Calculates a new objective and update
                 a current solution if the objective is
                 better than the previous one;
16         end
17     end
18 end
19 Do neighbourhood search using 2-ump
     neighbourhood until all constraints are satisfied
     and the time limit is reached;
```

2.1 Greedy Matching Heuristic

A simple greedy matching heuristic seamlessly handles constraints 1 and 2 but it runs into difficulties enforcing constraints 4 and 5. Because of that, this approach is then extended to use the Benders' cuts and a neighbourhood search.

This heuristic builds the umpire schedules starting from the first slot and ending at the slot $4n - 2$. For every slot s, the heuristic assigns umpires to games such that all constraints, except constraint 3, are satisfied and the best possible assignment is made to minimise the total umpire travel at slot s. The heuristic solves a perfect matching problem on a bipartite graph in every slot s. The partitions in this graph are the umpires and the games in the slot s. An edge is placed between an umpire u and a game (i, j) (a game is represented as a pair (i, j) where i is the home team and j is the away team) if constraints 4 and 5 are not violated by assigning u to game (i, j) in slot s, given the assignments from slot 1 to $s - 1$. The cost of edge $(u, (i, j)) = Distance(k, i)$. In this cost function, k is the venue that u is assigned in slot $s - 1$ and $Distance(k, i)$ is the distance between venues k and i.

This heuristic often gets stuck at some time slot because there may be no feasible perfect matching. In this case, a set of previous that are causing this lack of perfect matching is identified. At least one of those previous assignments must be change in order to create a perfect matching. This leads to a set of Benders' cuts used to guide a large neighbourhood search heuristic. Using this search, a solution that satisfies all the Benders' cuts is found and the perfect matching problem for slot s can be solved again. It there is no feasible perfect matching, it is required to repeat this process. The Benders' cuts and the large neighbourhood search are described further.

2.1.1 Arbitrarily Assignment in the First Slot

It was found out that that by arbitrarily assigning the games to umpires in the first slot, it is broken the symmetry of the problem and the solution quality is then improved and the solution times are reduced very significantly. For this reason, this arbitrarily assignment was also implemented within this project.

2.1.2 Incentive to Assign Unvisited Venues

The only constraint that the greedy matching heuristic is unable to explicitly handle is constraint 3, because it spans the whole schedule and its feasibility can not be ensured within such a constructive heuristic. Thus, this constraint is handled as follows. While forming the matching problem at a slot s, the cost of an assignment (u, i) is discounted by a defined discount if umpire u has never visited venue i in the previous slots. This reduction guides the heuristic towards assigning the umpires to venues that they have not visited yet. So, the possibility of having a solution that violates constraint 3 at the end of the execution of the heuristic is reduced. If the resulting solution violates constraint 3, then the infeasibility is penalised in the objective function during the course of the large neighbourhood algorithm executed afterwards.

2.1.3 Backtracking

As shows Algorithm 1, when there is no feasible matching at a slot s during the course of the greedy matching heuristic, before the Benders' cuts are generated and the neighbourhood search is executed, it is first attempt to change the last game-umpire assignment to obtain a feasible matching in slot s. For that, it is backtracked to the previous slot $s-1$ and it is picked the second best matching, instead of the best one, and the perfect matching is tried for slot s again. This backtracking is made at most once at each slot.

2.2 Very Large Neighbourhood Search with Benders' Cuts

An algorithm that starts at some initial solution and iteratively moves to solutions in the neighborhood of the current solution is called a neighborhood search algorithm or a local search algorithm. This algorithm is guided by Bender's cuts described below.

2.2.1 Benders' Cuts

Benders' cuts are generated when there is no feasible matching at slot s during the course of the greedy matching heuristic. The Benders' Cut states that at least one of the edges between the the umpires and the games that can not be matched because of constraints has to be in the graph. Further information about these cuts can be found in [3].

3 Implementation

The described algorithm has been implemented in Python programming language. At least, it is required Python version 3.6. Python libraries NumPy[1] and SciPy[2] was used for mathematics computation. Required versions of these libraries are specified in file `requirements.txt`. The implementation consists of several modules described in the paragraphs below.

The module `main` severs an entry point of the application. It reads an input file with the specification of the problem and it executes the greedy matching heuristic.

Modules `inp` and `out` contains functions for reading and parsing input files with the specification of the problem, and functions for writing solutions to output files in the appropriate format.

A class `Tup` in module `tup` represents the Traveling Umpire Problem. The class encapsulates the specification of the problem as well as its input parameters. Moreover, it contains operations such as computation of the total umpire travel or computation of violations of single constraints. All of the methods are properly commented in the source code.

The most interesting is module `gmh` which contains five main functions. Function `gmh` the greedy matching heuristic algorithm. It tries to find a perfect matching in every slot. In the case that there is no feasible perfect matching, the Benders' cuts are used to guide the large neighbourhood search heuristic. Function `benders_cuts` computes all the Benders' cuts in a given slot. Function `benders_violations` calculates the number of violated Benders' cuts in a current solution. The very large neighbourhood search algorithm uses the Benders' cuts is implemented in function `neigh_search`. Finally, function `neigh_search_objective` computes the value of the objective function which is used during the large neighbourhood search.

3.1 Created Tool Usage

The created tool is capable of read an instance of the problem and a solution. Format of an input instance is described in Section 4.1. Experimental problem instances are in directory `input/`. Output of the tool is a single line indicating the umpire assigned for each game as follows:

$$A_{11}, A_{12}, \ldots, A_{1g}, A_{21}, A_{22}, \ldots, A_{2g}, \ldots, \ldots,$$

$$A_{r1}, A_{r2}, \ldots, A_{rg}$$

where $A_{rg} \in \{1, 2, \ldots, U\}$ denotes the umpire assigned to game $g \in \{1, 2, \ldots, U\}$ of round $r \in \{1, 2, \ldots, R\}$. U is the number of umpires and R is the number of rounds. The solution will be printed in the appropriate file to directory `output/`.

There is a prepared `Makefile` for running the program. It can be run using the following command: `make run INST=inst D1=d1 D2=d2 LIMIT=limit`, where

[1] Python library for mathematics computation NumPy – https://numpy.org.

[2] Python library for mathematics computation SciPy – https://www.scipy.org.

`inst` is a name of an instance of the problem, `d1` is the parameter d_1 of 4. constraint, `d2` is the parameter d_2 of 5. constraint, and `limit` is the time limit of the execution. E.g., `make run INST=ump4 D1=0 D2=0 LIMIT=30` executes the program with an instance with 4 teams stored in input directory with $d_1 = d_2 = 0$ and with the time limit 30 minutes.

4 Experimental Evaluation

This chapter describes used data sets, discusses executed experiments, and compares the results with the paper.

4.1 Data Sets Description

Data sets are available in [1]. It also can be found in directory `input/`. It was used the same problem instances as the instances used in the paper, so it can be easily compared. A format of the instances is described in file `input/input-format.pdf`.

4.2 Experiments Results

Table 1 show results of execution of the program on smaller problem instances from 4 teams to 10 teams. The instances followed by a letter A, B, or C are the same as the original with the distances matrix is changed. In the Table 2 there can be seen experiments with larger instances.

Experiments show that results of the implemented algorithm within this project are comparable with the results in the paper. These algorithm has been run for several minutes. It is likely that longer run would lead to more optimal results.

Instance	Optimal Dist.	Paper	**Harmim**
4	5,176	5,176	**5,176**
6	14,077	14,077	**14,077**
6A	15,457	15,457	**15,735**
6B	16,716	16,716	**16,716**
6C	14,396	14,396	**14,396**
8	34,311	34,311	**34,311**
8A	31,490	31,490	**31,490**
8B	32,731	32,731	**32,731**
8C	29,879	29,879	**29,879**
10	48,942	49,528	**53,898**
10A	46,551	46,822	**49,259**
10B	45,609	45,609	**51,920**
10C	43,149	43,533	**45,396**

Table 1: Comparison of experimental results of smaller instances executed in the paper and in this project (Harmim)

Instance	$n - d_1$	$\frac{n}{2} - d_2$	Paper	**Harmim**
14	6	3	176,290	**178,697**
14	5	3	167,146	**181,675**
14A	6	3	172,764	**176,445**
14A	5	3	163,978	**170,624**
14B	6	3	179,278	**175,161**
14B	5	3	160,717	**172,492**
14C	6	3	172,243	**183,794**
14C	5	3	168,970	**173,963**
16	7	2	166,274	**181,548**
16A	7	2	177,857	**189,470**
16B	7	2	184,923	**197,952**
16C	7	2	181,013	**202,693**

Table 2: Comparison of experimental results of larger instances with less restricted constraints executed in the paper and in this project (Harmim)

5 Conclusion

Within this project, algorithm from [3] for solving the Traveling Umpire Problem has been successfully implemented and experimental evaluated.

Bibliography

[1] Toffolo, T. A. M.; Wauters, T.; Trick, M. A.: An automated benchmark website for the Traveling Umpire Problem [online]. June 2015 [cit. 2020-04-27]. Retrieved from: http://gent.cs.kuleuven.be/tup

[2] Trick, M. A.; Yildiz, H.: Bender's Cuts Guided Large Neighborhood Search for the Traveling Umpire Problem. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 4th International Conference, CPAIOR 2007, Brussels, Belgium, May 23–26, 2007. Proceedings*, *Lecture Notes in Computer Science*, vol. 4510, edited by P. Van Hentenryck; L. Wolsey. Berlin, Heidelberg: Springer Berlin Heidelberg. 2007. ISBN 978-3-540-72397-4. pp. 332 – 345. doi:10.1007/978-3-540-72397-4_24.

[3] Trick, M. A.; Yildiz, H.: Benders' cuts guided large neighborhood search for the traveling umpire problem. *Naval Research Logistics (NRL)*. vol. 58, no. 8. October 2011: pp. 771 – 781. ISSN 0894-069X. doi: 10.1002/nav.20482.