

# Using Heuristics in Constraint-based Sports Tournament Timetabling<sup>1</sup>

Hon Wai CHUN  
Department of Computer Science  
City University of Hong Kong  
Tat Chee Avenue, Kowloon  
Hong Kong SAR  
[andy.chun@cityu.edu.hk](mailto:andy.chun@cityu.edu.hk)

Ngai Ming LAM  
Department of Electronic Engineering  
City University of Hong Kong  
Tat Chee Avenue, Kowloon  
Hong Kong SAR  
[50135941@student.cityu.edu.hk](mailto:50135941@student.cityu.edu.hk)

## ABSTRACT

Round robin is a popular format for many types of sports tournament. Different approaches have been proposed in the past to solve round robin sports tournament timetabling problems. One approach is to model this problem as a constraint-satisfaction problem (CSP). Because of the computational complexity of tournament timetabling, this problem is usually broken down into sub-problems and solved separately in steps. Although it is well known that a single-step CSP model does not scale well with increasing number of teams, this paper investigates how far we can push the performance limit of a single-step model through the use of different search heuristics. This paper focuses on solving a specific type of round robin tournaments – temporally dense single round robin tournaments (TDSRR). We experimented with different types of heuristics, which we classified as variable-based heuristics, constraint-based heuristics, structure-based heuristics and model-based heuristics. The performances of different heuristics within these classifications are compared in this paper.

**Keywords:** Sports Tournament Timetabling, Round Robin Scheduling, Constraint Satisfaction Problem, Search Heuristics.

## 1. INTRODUCTION

Round robin is a popular format for many types of sports tournament. In this type of sports tournament,  $n$  teams play against each other  $r$  times. Usually,  $r$  is either 1 or 2. The tournament is called

*single round robin* if  $r$  is 1, and *double round robin* if  $r$  is 2. The matches are usually held in either of the opponents' facilities. A match at a team's own facility is called a *home* match, otherwise it is called an *away* match.

A *temporally dense* round robin tournament is one where  $r n (n-1)/2$  matches can be completed within  $d$  dates, assuming each team plays at most one match per date. For even number of teams,  $d$  is equal to  $r(n-1)$ . For odd number of teams,  $d$  is equal to  $rn$  dates where  $n-1$  teams play while one does not. That team is said to have a *bye* on that date.

Most double round robin tournaments can be reduced to a single round robin problem using *mirroring*, where each date has a mirror date when the same match is played again but at the other opponent's site. Therefore, scheduling the first match automatically schedules the mirror match.

This paper focuses on the temporally dense single round robin (TDSRR) tournament problem. The following (Table 1) is a sample tournament schedule for  $n = 9$  teams,  $r = 1$  (single round robin), and  $d = (r n) = 9$  dates.

The notation used in this paper is similar to that of Henz [4], Schreuder [12] and Russell [10]. Each matrix row contains all the matches that a particular team will play at different dates. The number in the matrix is the opponent team's ID. "b" represents a *bye*, "+" is a *home* match, and "-" is an *away* match. Each column, on the other hand, contains all the matches that are to be held in one particular date. In other words, row  $i$  and column  $j$  shows the match

---

<sup>1</sup> "This work was supported in part by a Hong Kong RGC Earmarked Grant and a Strategic Research Grant provided by the City University of Hong Kong."

that team  $i$  will play in date  $j$ . Numbers in each column are unique as each team only plays one match per date. Numbers in each row are also unique as each team plays against another team only once in a single round robin.

Date	1	2	3	4	5	6	7	8	9
Team 1	b	-9	-5	+7	-2	+3	+6	-4	+8
Team 2	-9	b	+8	-6	+1	-4	-3	+5	+7
Team 3	-8	+5	b	-9	+7	-1	+2	-6	+4
Team 4	-6	-7	+9	b	+8	+2	-5	+1	-3
Team 5	+7	-3	+1	-8	b	-9	+4	-2	+6
Team 6	+4	-8	-7	+2	+9	b	-1	+3	-5
Team 7	-5	+4	+6	-1	-3	+8	b	+9	-2
Team 8	+3	+6	-2	+5	-4	-7	+9	b	-1
Team 9	+2	+1	-4	+3	-6	+5	-8	-7	b

**Table 1.** One solution to TDSRR with  $n = 9$  teams,  $r = 1$  (single round robin).

The notation used in this paper is similar to that of Henz [4], Schreuder [12] and Russell [10]. Each matrix row contains all the matches that a particular team will play at different dates. The number in the matrix is the opponent team's ID. "b" represents a *bye*, "+" is a *home* match, and "-" is an *away* match. Each column, on the other hand, contains all the matches that are to be held in one particular date. In other words, row  $i$  and column  $j$  shows the match that team  $i$  will play in date  $j$ . Numbers in each column are unique as each team only plays one match per date. Numbers in each row are also unique as each team plays against another team only once in a single round robin.

Despite the simplicity of the problem statement, the problem is highly computationally intensive for large values of  $n$ . Using a single-step CSP model, which we will describe later, and a common search heuristic of selecting the first unbound variable and assigning it the minimum value from its domain, the search stalls even for a very small value of  $n=6$  and  $r=1$ .

In our experiments, for cases where  $n \geq 9$ , we further limit the solutions with the following constraints, borrowed from Nemhauser [9]:

- **No Two Final Aways** – No team can play *away* on both the last two dates.
- **Home/Away/Bye Pattern Criteria** – No team may have more than two away matches in a row. No team may have more than two home matches in a row. No team may have more than

three away matches or byes in a row. No team may have more than four home matches or byes in a row.

## 1I. PAST APPROACHES

Nemhauser and Trick [9] presented an approach to solving the 1997/1998 Atlantic Coast Conference (ACC) – a major basketball tournament for 9 universities in the southeastern United States ( $n=9$  and  $r=2$ ). They used a combination of integer programming with enumeration and divided the problem into 3 steps:

- **Step 1:** Find patterns of home (+), away (-), and bye (b) with length equal to  $d$  dates. The following is one such pattern for  $n=9$  and  $r=2$ :

Date	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Pattern	+	-	+	+	-	+	-	-	+	b	+	-	-	+	-	+	b	-

A set of  $n$  patterns that satisfies the tournament constraints is called a *pattern set*.

- **Step 2:** Assign games to pattern sets to generate timetables.
- **Step 3:** Assign an actual team to each game to produce a tournament schedule.

For [9], Step 1 is solved by enumeration combined with integer programming. Step 2 is then solved by integer programming. Finally, step 3 is solved by enumeration. This approach was able to enumerate all solutions in roughly 24 hours on a Sun SPARC-20 workstation. Russell and Leung also used similar 3 step approaches [10] but with combinatorial design theory with enumeration.

Henz [3, 4, 5, 6, 7] solved the same problem using only finite-domain constraint programming. However, the problem was still divided into 3 steps with a separate constraint model for each step. Using this approach, Henz was able to generate all solutions within one minute on a PC.

Other researchers used combinations of Genetic Algorithm and Tabu search for similar types of timetabling problems [1, 2].

## 1II. OUR SINGLE-STEP MODEL

Previous approaches used 3 separate steps since a single-step model was too computationally intensive. Obviously, not all constraint-based problems can be sub-divided into smaller problems. With the objective of finding general solutions to this type of timetabling problems, we instead use a

single-step constraint-based model to test the limits of this approach. Our problem is a generalization of the problem solved in Nemhauser and Trick [9] and Henz [5], which was specific to only the 1997/1998 ACC problem with  $n=9$ . Our single-step model on the other hand works for any value of  $n$ . The model itself is used mainly as a test bed to investigate the effects of search heuristics on performance in timetabling problems. McAloon [8] also used a similar single-step model. They remarked that their code's performance was very delicate and at best can find a solution to  $n=14$  within 45 minutes on an UltraSparc with C++. Our best heuristics was able to find  $n=18$  solutions with an average of 2.3 seconds each on a PC with Java.

Many types of scheduling problems can be formulated as a constraint-satisfaction problem (CSP) [13, 14, 15], which involves the assignment of values to variables subjected to a set of constraints. CSP can be defined as consisting of a finite set of  $m$  variables  $v_1, v_2, \dots, v_m$ , a set of domains  $d_1, d_2, \dots, d_m$ , and a set of constraint relations  $c_1, c_2, \dots, c_k$ . Each  $d_i$  defines a finite set of values (or solutions) that variable  $v_i$  may be assigned. A constraint  $c_j$  specifies the consistent or inconsistent choices among variables and is defined as a subset of the Cartesian product:  $c_j \subseteq d_1 \times d_2 \times \dots \times d_m$ . The goal of a CSP algorithm is to find one tuple from  $d_1 \times d_2 \times \dots \times d_m$  such that  $m$  assignments of values to variables satisfy all constraints simultaneously.

In our single-step model, each match  $M_{i,j}$  of row  $i$  and column  $j$  of the timetable is modelled with 2 constrained variables –  $loc_{i,j}$  and  $team_{i,j}$ :

- $loc_{i,j}$  is a constrained variable representing the location of the match for team  $i$  on date  $j$  with a domain of 3 possible values - [home, away, bye]
- $team_{i,j}$  is a constrained variable representing the opponent team that team  $i$  will play against on date  $j$  with domain consisting of all the possible teams except  $i$ . If  $loc_{i,j} = \text{bye}$ ,  $team_{i,j}$  will be assigned a dummy value as there will be no opponent.

The basic constraints in our model are:

- For each date  $j$ ,  $\Sigma(loc_{i,j}=\text{bye})$  is  $b=0$  if  $n$  is even, and  $b=1$  if  $n$  is odd
- For each date  $j$ ,  $\Sigma(loc_{i,j}=\text{home})$  is  $(n-b)/2$
- For each date  $j$ ,  $\Sigma(loc_{i,j}=\text{away})$  is  $(n-b)/2$
- For each team  $i$ ,  $\Sigma(loc_{i,j}=\text{bye})$  is  $b=0$  if  $n$  is even, and  $b=1$  if  $n$  is odd
- For each team  $i$ ,  $\Sigma(loc_{i,j}=\text{home})$  is  $r(n-b)/2$

- For each team  $i$ ,  $\Sigma(loc_{i,j}=\text{away})$  is  $r(n-b)/2$
- For each  $M_{i,j}$ , if  $team_{i,j}=x$ , then  $team_{x,j}=i$ , unless  $loc_{i,j}=\text{bye}$
- For each  $M_{i,j}$ , if  $loc_{i,j}=\text{home/away}$ , then  $loc_{x,j}=\text{away/home}$  where  $x$  is the opponent
- For each team  $i$ , each opponent  $x$ ,  $\Sigma(team_{i,j}=x)$  is  $r$
- For each date  $j$ , value of  $team_{i,j}$  is all different

In addition, there are the previously mentioned constraints:

- **No Two Final Aways**
- **Home/Away/Bye Pattern Criteria**

The CSP search algorithm finds a solution by assigning a value to each variable. The value is taken from the variable's current domain, which gets reduced through constraint propagation as the search tree gets expanded. This *domain reduction* effect helps "narrow" down the search (trim down the search tree) and hence reduces search time. Therefore, the "shape" of the search tree as it gets expanded has a major impact on the performance of any CSP algorithm. If the search tree is not expanded correctly, the CSP search might not be any better than simple exhaustive search.

## 1V. SEARCH HEURISTICS

The "shape" of the CSP search tree is determined greatly by the search heuristics that guide the search. In general, CSP search heuristics consists of two components – heuristics to determine the order that constrained variables should be selected for instantiation, and heuristics on the order that values should be retrieved from the variable domain to be used as the instantiated variable value. We call these *choose variable heuristics* and *select value heuristics* respectively.

These heuristics may be chained together in any order, in ways similar to how Java streams may be chained or piped together. For example, we can chain a "<size" with a "<value" heuristic to select the variable with the minimum domain size (<size) and minimum domain value (<value). We represent this heuristics as "<size-<value".

In this paper, we investigate the effects of different types of heuristics on the performance of the TDSRR timetabling problem. In our experiments, we further divide the *choose variable heuristics* into different categories:

- **Variable-based Heuristics** – these are heuristics that order constrained variables based purely

on information about the variables, such as position in the list of uninstantiated variables, the domain size, etc.

<i>Feature</i>	<i>Heuristic</i>	<i>Description</i>
order	<order	first unbound var first
	>order	last unbound var first
	random	random order
size	<size	var with min domain size first
	>size	var with max domain size first
value	<value	var with min value in domain first
	>value	var with max value in domain first

- **Constraint-based Heuristics** – these are heuristics that order constrained variables based on information about the constraint-based environment, such as number of associated constraints, number of associated variables, etc.

<i>Feature</i>	<i>Heuristic</i>	<i>Description</i>
cnst	<cnst	var with min number of constraints first
	>cnst	var with max number of constraints first
pcnst	<pcnst	var with min number of propagatable constraints first
	>pcnst	var with max number of propagatable constraints first
avar	<avar	var with min number of associated constrained variables first
	>avar	var with max number of associated constrained variables first
uavar	<uavar	var with min number of unbound associated constrained variables first
	>uavar	var with max number of unbound associated constrained variables first
bavar	<bavar	var with min number of bound associated constrained variables first
	>bavar	var with max number of bound associated constrained variables first

- **Structure-based Heuristics** – these are heuristics that order constrained variables based on the structure of the model, such as order of the row in the timetable matrix a variable belongs to.

<i>Feature</i>	<i>Heuristic</i>	<i>Description</i>
row	<row	var with min row index first
	>row	var with max row index first
col	<col	var with min column index first
	>col	var with max column index first
xrow	<xrow	var in both row extremes first
	>xrow	var in middle rows first
xcol	<xcol	var in both column extremes first
	>xcol	var in middle columns first
uni	<uni	vars uniformly spread out
	>uni	vars clustered to other instantiated vars

- **Model-based Heuristics** – these are heuristics that order constrained variables based on aspects of the model, such as selecting *loc* variables first over *team* variables for the timetabling problem.

<i>Feature</i>	<i>Heuristic</i>	<i>Description</i>
match	match	assign one match at a time
team	team	assign team variables first
loc	loc	assign loc (home/away/bye) variables first

Similarly, we divide the *select value heuristics* into the following categories:

- **Local Heuristics** – these are heuristics that order values in the domain based on local information, such as the order or value of the domain values.

<i>Feature</i>	<i>Heuristic</i>	<i>Description</i>
order	<order	first value in domain first
	>order	last value in domain first
value	<value	min value in domain first
	>value	max value in domain first

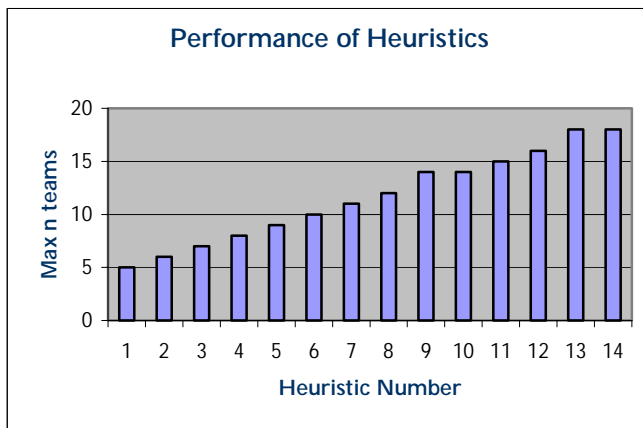
- **Global Heuristics** – these are heuristics that order values in the domain based on global information, such as the frequency of use and availability of different values.

Feature	Heuristic	Description
freq	<freq	least frequent value used in all bound vars first
	>freq	most frequent value used in all bound vars first
Avail	<avail	least frequent available value in all domains first
	>avail	most frequent available value in all domains first

## V. EXPERIMENTS

We implemented our single-step CSP model using a Java constraint-programming class library and ran the experiments on a Pentium III 750 MHz PC. The constrained variables of the TSDRR problem were created and stored in a list. The initial order of the list consists of all the *loc* variables followed by all the *team* variables. The variables are stored in row-major order. For a problem with  $n$  teams and  $d$  dates, there are  $2(nd)$  variables total in the list. We then search for a solution to the TSDRR problem by assigning a value to each variable. Variables and values were selected in the order determined dynamically by our heuristics chain.

We tested these search heuristics by running experiments with  $n=4$  and upwards. The first 10 solutions for each value of  $n$  were retrieved. Each run has a time limit of 3 minutes, after which it will time out and continue with the next experiment. The following Chart 1 shows some of our results.



**Chart 1.** The maximum number of teams solved by different heuristics.

The heuristics tested were:

Heuristics	Variable	Value	Max n
1	<size >value	>value	5
2	<uni	<value	6
3	<cnst <size	<value	7
4	<xrow	<value	8
5	<row >size >value	>value	9
6	<col >size >value <cnst	>value	10
7	<row >size >value <cnst	>value	11
8	>size >value	<value	12
9	>size >value <cnst	>value	14
10	>size <value	>value	14
11	>order	<value	15
12	loc >size >value	>value	16
13	>size >value	>value	18
14	>size >value team	>value	18

## V1. CONCLUSION

Surprisingly, for the TSDRR timetabling problem, more complex heuristics, such as constraint-based heuristics did not contribute to better performances. Instead, a more generic “>size >value” *choose variable heuristic* combined with “>value” *select value heuristic* produced the best performance, solving TSDRR tournaments with  $n=18$  teams with the first solution in 18.7 seconds and then an average of 0.5 seconds for subsequent solutions. This search heuristic basically selected constrained variables that were least constrained, i.e., had more choices, first. This leaves highly constrained variables to be pushed to the bottom of the search tree, probably allowing more efficient backtracking and hence better performance for the TSDRR timetabling problem.

## 12. REFERENCES

- [1] Costa, Daniel, “An evolutionary tabu search algorithm and the NHL scheduling problem,” ORWP 92/11, Swiss Federal Institute of Technology, Department of Mathematics, 1992.
- [2] J.P. Hamiez and J.K. Hao, “Solving the sports league scheduling problem with Tabu search,” *Lecture Notes in Artificial Intelligence* 2148: 24-36, Springer-Verlag. (Preliminary version)

presented at ECAI00 Workshop on Local Search for Planning and Scheduling, Berlin, August, 2000).

P. Van Hentenryck, *Constraint Satisfaction in Logic Programming*, MIT Press, 1989.

- [3] Henz, Martin, and Jörg Würtz, "Constraint-based Time Tabling - A Case Study," *Applied Artificial Intelligence*, 10:439-453, 1996.
- [4] Henz, Martin, "Constraint-based Round Robin Tournament Planning," In the *Proceedings of the 1999 International Conference on Logic Programming*, pp.545-557, 1999.
- [5] Henz, Martin, "Scheduling a major college basketball conference - revisited," *Operations Research*, 49(1), Jan/Feb, 2001.
- [6] Henz, Martin, Tobias Müller, Sven Thiel and Marleen van Brandenburg, "Global Constraints for Round Robin Tournament Scheduling," *EJORS Special Issue on Timetabling*, 30 September 2001.
- [7] Henz, Martin, Tobias Müller, Sven Thiel and Marleen van Brandenburg, "Benchmark Results for Constraint-based Round Robin Tournament Scheduling," to be published.
- [8] McAloon, Ken, Carol Tretkoff, and Gerhard Wetzl, "Sports league scheduling," In *Proceedings of the 1997 ILOG Optimization Suite International Users' Conference*, Paris, July, 1997.
- [9] Nemhauser, G. and M. Trick: 1998, "Scheduling a major college basketball conference," *Operations Research* 46(1), 1-8.
- [10] Russell, Robert A. and Janny M.Y. Leung, "Devising a cost effective schedule for a baseball league," *Operations Research*, 42(4):614-625, 1994.
- [11] Schaerf, Andrea, "Scheduling sport tournaments using constraint logic programming," In *Proceedings of the European Conference on Artificial Intelligence*, pages 634-639, Budapest, Hungary, 1996.
- [12] Schreuder, Jan A. M., "Combinatorial aspects of construction of competition dutch professional football leagues," *Discrete Applied Mathematics*, 35:301-312, 1992.
- [13] Steele, G.L. Jr., *The Definition and Implementation of a Computer Programming Language Based on Constraints*, Ph.D. Thesis, MIT, 1980.
- [14] Tsang, E., *Foundations of Constraint Satisfaction*, Academic Press, 1993.