Towards a deterministic algorithm for the International Timetabling Competition

Oscar Chávez Bosquez[1], Pilar Pozos Parra[1], and Florian Lengyel[2]

[1] Department of Informatics and Systems
University of Tabasco
Carretera Cunduacán - Jalpa Km. 1, Tabasco, Mexico
{oscar.chavez,pilar.pozos}@dais.ujat.mx

[2] Department of Computer Science
The Graduate Center, CUNY
365 Fifth Ave., New York, USA
flengyel@gc.cuny.edu

Abstract

The Course Timetabling Problem consists of the weekly scheduling of lectures of a collection of university courses, subject to certain constraints. The International Timetabling Competitions, ITC-2002 and ITC-2007, have been organized with the aim of creating a common formulation for comparison of solution proposals. This paper discusses the design and implementation of an extendable family of sorting-based mechanisms, called Sort Then Fix (STF) algorithms. Only ITC-2002 problem instances were used in this study. The STF approach is deterministic, and does not require swapping or backtracking. Almost all solutions run in less than 10% of the ITC-2002 benchmark time.

1 Introduction

The Course Timetabling Problem consists of assigning a sequence of events (lectures) to a collection of university courses that meet within a number of rooms and time periods, usually weekly, satisfying some constraints. Course timetabling problems vary from university to university, and many researchers have shown that there is no single best solution for all situations.

Standards for comparing algorithms for solving the Course Timetabling Problem have emerged in recent years. The International Timetabling Competitions, ITC-2002 and ITC-2007, have been organized with the aim of creating a common formulation for comparing solutions, of which many have been proposed [15, 7].

While the competition is open to stochastic and deterministic approaches, virtually all the proposed solutions appearing in the competition web pages are stochastic algorithms [5]; as far as we know there is no record of a competitive and effective deterministic approach used in the contest.

The motivation of this work was to develop a deterministic algorithm that solves the hard constraints of the 20 instances of the ITC-2002 in a

Proceedings of the 17th International RCRA workshop (RCRA 2010):
Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion
Bologna, Italy, June 10–11, 2010

timely manner, so we introduce a family of deterministic algorithms, Sort Then Fix (STF) algorithms, for solving timetabling problems. The algorithms were tested with the official instances of the ITC-2002.

The rest of the paper is organized as follows. We formalize the course timetabling problem in the second section; the third section describes the family of deterministic STF algorithms; the fourth section provides our results; and we conclude with some possible research directions.

2 Target Problem

We consider the problem of weekly scheduling a set of single events (or lectures). The problem has been discussed in [1] and it was the topic of ITC-2002 [2], where twenty artificial instances were proposed. The instances are available from the ITC-2002 web page.

The problem consists of finding an optimal timetable within the following framework: there is a set of events $E = \{E_1, E_2, ..., E_{n_E}\}$ to be scheduled in a set of rooms $R = \{R_1, R_2, ..., R_{n_R}\}$, where each room has 45 available timeslots, nine for each day in a five day week. There is a set of students $S = \{S_1, S_2, ..., S_{n_S}\}$ who attend the events, and a set of features $F = \{F_1, F_2, ..., F_{n_F}\}$ satisfied by rooms and required by events. Each event is attended by a number of students, and each room has a given size, which is the maximum number of students the room can accommodate. A feasible timetable is one in which all events have been assigned a timeslot and a room so that the following hard constraints are satisfied:

(1) no student attends more than one event at the same time;

(2) the room is big enough for all the attending students and satisfies all the features required by the event; and

(3) only one event is scheduled in each room at any timeslot.

In contest instance files there were typically 10-11 rooms, hence there are 450-495 available places. There were typically 350-400 events, 5-10 features and 200-300 students.

The problem will penalize a timetable for each occurrence of some soft constraint violation, which are the following:

(1) a student has to attend an event in the last timeslot on a day;

(2) a student has more than two classes in a row; and

(3) a student has to attend solely an event in a day.

The problem may be precisely formulated as:

- let $F = \{F_1, F_2, ..., F_{n_F}\}$ be a set of symbols representing the features;
- $R_i = \{F'_1, F'_2, ..., F'_{n_{R_i}}\}$ where $F'_j \in F$ for $j = 1, ..., n_{R_i}$ and n_{R_i} is the number of features satisfied by room R_i;
- $N = \{N_1, ..., N_{n_R}\}$ be a set of integer numbers indicating the maximum of students each room can accommodate;
- $E_i = \{F''_1, F''_2, ..., F''_{n_{E_i}}\}$ where $F''_j \in F$ for $j = 1, ..., n_{E_i}$ and n_{E_i} is the number of features required by event E_i;
- $S_i = \{E'_1, E'_2, ..., E'_{n_{S_i}}\}$ where $E'_j \in E$ for $j = 1, ..., n_{S_i}$ and n_{S_i} is the number of events student S_i attends; and
- $T = \{T_1, ..., T_{45}\}$ be a set of timeslots.

Find a feasible solution, i.e. a set of pairs $\{(T'_1, R'_1), ..., (T'_{n_E}, R'_{n_E})\}$ such that:

- $T'_i \in T$ and $R'_i \in R$
- $(T'_i \neq T'_j)$ if $E_i \in S_k$ and $E_j \in S_k$ and $(i \neq j)$
- $E_i \subseteq R'_i$ and $|\{S_j | j = 1, ..., n_S \text{ and } E_i \in S_j\}| \leq N_k$ where $R'_i = R_k$;
- $\forall i \forall j \ (i = j \vee T'_i \neq T'_j \vee R'_i \neq R'_j)$.

The competition adds a constraint over execution time, i.e. given the information about rooms, events, features, and students, find the best possible feasible solution within a given time limit. The time limit is given by a benchmark tool provided by the organizer.

3 *STF* Algorithms

STF is a family of algorithms to solve timetabling problems, based on events and rooms sorting. Its distinguishing property is that the k-th iterate yields a schedule for the k most constrained events, where the notion of the most constrained event depends on the particular STF algorithm.

The first step of an STF algorithm is to find a binary matrix that represents the available rooms for every event. The following actions are performed:

- the number of students for each event is calculated and stored,
  $n_i = |\{S_j | j = 1, ..., n_S \text{ and } E_i \in S_j\}|$
- a list of available rooms is created for each event,
  $e_i = \{R_j | E_i \subseteq R_j \text{ and } n_i \leq N_j\}$

This first step allows us to reduce the problem by eliminating the information concerning features and room capacity, defining a new event set $\{e_1, ..., e_{n_E}\}$, that includes the eliminated information. The new event set will be used for defining the most constrained event. The core strategy of the algorithms is to assign the most constrained event to the least constrained timeslot found for this event. In order to avoid strategies of movement that require backtracking to recover from mistakes, our aim is not to make mistakes of selection of placement and event. The intuitive idea is as follows: if there is at least one optimal solution where all soft and hard restrictions are satisfied, then it is possible to sort the events and placements in such a way that all the events are fixed satisfying all the constraints. The general structure is as follows:

1. sort events from the most constrained to the least constrained one,
2. choose $e_i \in [e_1, \ldots, e_{n_E}]$, the next most constrained event,
3. sort the rooms of e_i from the least to the most constrained one,
4. choose $r \in e_i$, the next least constrained room for event e_i,
5. search $s \in [1 \ldots 45]$, such that:
   $\forall j\ (i = j \vee s \neq T'_j \vee r \neq R'_j)$ and if $E_i, E_j \in S_k$ and $i \neq j$, $s \neq T'_j$,
6. if $(s \neq null)$ then $R'_i := r$ and $T'_i := s$ else go to step 4,
7. update data for the next iteration,
8. go to step 2 (or go to step 1, depending on the STF algorithm).

The different versions have particular criterions that are described in Table 1. Multiple criteria take precedence, from highest to lowest, in order of occurrence. For example, algorithm 3 has two criterions for ordering the events, first by the number of rooms that can allocated to the event if there are two or more events with the same number of rooms; second, by the number of students attending the event.

We retain the idea of the early techniques for solving the timetabling problem; our proposal is based on a simulation of the human way of solving the problem with a successive augmentation. In [14], these techniques are called direct heuristics. We start with an empty timetable that is extended event by event, until all the events have been scheduled. The idea is to schedule the most constrained event first, and then the second most constrained event, and so on until all the events are scheduled. Our approach differs from direct heuristics, which usually fill up the complete timetable with one event at a time as far as no conflicts arise, and until they begin swapping to accommodate any remaining events.

Table 1: Details of STF

Algorithm	Event sorting criterion	Room sorting criterion	Go to step
1	(a)	(1)	2
2	(b)	(1)	2
3	(a) (b)	(1)	2
4	(b) (a)	(1)	2
5	(a)	(1) (2)	2
6	(b)	(1) (2)	2
7	(a) (b)	(1) (2)	2
8	(b) (a)	(1) (2)	2
9	(c)	(1)	1
10	(c) (d)	(1)	1
11	(c)	(1) (2)	1
12	(c) (d)	(1) (2)	1
13	(c) (b)	(1)	1
14	(b) (c)	(1)	1
15	(c) (b)	(1) (2)	1
16	(b) (c)	(1) (2)	1
17	(c) (b) (d)	(1)	1
18	(b) (c) (d)	(1)	1
19	(c) (b) (d)	(1) (2)	1
20	(b) (c) (d)	(1) (2)	1
21	(c) (b) (f)	(1)	1
22	(b) (c) (f)	(1)	1
23	(c) (b) (f)	(1) (3)	1
24	(b) (c) (f)	(1) (3)	1
25	(b) (a) (e)	(1) (3)	1
26	(e) (b) (a) (f)	(1) (3)	1
27	(e) (b) (a)	(4)	1
28	(g) (b) (a)	(4)	1
29	(g) (b) (a)	(4) (3)	1

The criteria of Table 1 are as follows:

(a) Number of rooms, in ascending order, where the event can be allocated.

(b) Number of students, in descending order, that attend the event.

(c) Number of free slots of all the event rooms in ascending order, where a free slot is one which has no assigned event.

(d) Number of events, in ascending order, that can be allocated to the event rooms, where the number of events includes assigned and unassigned events.

(e) Number of free slots of all the event rooms in ascending order, where a free slot is one which has no associated student attending another event assigned to the same slot and the slot has no assigned event in at at least one event room.

(f) Number of unassigned events, in ascending order, that can be allocated to the event rooms.

(g) Number of free slots of all the event rooms in ascending order, where a free slot is one which has no assigned event and no attending student has another event placed at the same slot.

(1) Number of free slots of the room in descending order, where a free slot is a slot that has no assigned event.

(2) Number of events that can be allocated in the room in ascending order, where the number of events includes assigned and unassigned events.

(3) Number of unassigned events that can be allocated to the room in ascending order.

(4) Number of slots that are free for the room in descending order, where a free slot has no assigned event and no attending student has another event placed at the same slot.

Our approach avoids swapping; if an event cannot be scheduled then the next most constrained event is scheduled, leaving the event in question without a room and timeslot. Thus we careful to formulate the definition of the "most constrained event" in the STF algorithms to avoid unscheduled events.

4 Results

The STF algorithms were developed in GNU Octave 3.0 on a Toshiba Satellite A105-S433®laptop computer, with a 1.60 GHz Centrino Duo proccesor T5200, Mobile Intel®945GM Express Chipset, 2 GB in RAM, 160 GB in HDD and the Ubuntu Linux 9.10 operating system.

All tests were done on this computer, within the time allowed by the benchmark tool provided by the organizers of the ITC-2002. This benchmark tool shows the time limit (in seconds) in which the algorithm must be executed in a particular computer, so that everybody can participate in a fair competition. In our box, the benchmark program allowed a maximum execution time of 558 seconds, time in wich the STF algorithms must be executed in order to find feasible solutions.

Tables 2 and 3 show the results obtained by the STF algorithms on the 20 problem instances of the ITC-2002. The table displays the hard constraints violated by the algorithms running under GNU Octave 3.0 and under Matlab ®R2009b.

Tables 4 and 5 display the soft constraints violated by the algorithms running on GNU Octave 3.0 and Matlab®R2009b.

In Fig. 1, we can see markedly different patterns of hard constraints violation. Some of them nearly find feasible solutions for all the instances. For example, algorithm 28 and 29 violated only one hard constraint of one instance. Some of them find feasible solutions only for a reduced number of instances, such is the case of algorithm 1, which only finds 3 of the 20 feasible solutions. It is worth noting that more refined sorting criterions will find a greater number of feasible solutions for all the instances. As we can see in Fig. 2, the graphs of the number of soft constraints violated by the 29 STF algorithms share a common pattern.

The Fig. 3 displays the time in seconds taken by the algorithms running on GNU Octave 3.0, and Fig. 4 shows the time taken by the algorithms running Matlab®R2009b. As we can see the both implementations of algorithms terminate within the 558 second allowed benchmark time and almost all the algorithms running on Matlab take less of the 10% of the allowed benchmark time, but the run over Octave takes much more time that Matlab. However, the purpose of Figs. 3 & 4 is to show the pattern that follows the STF algorithms in matter of time, and to prove that STF algorithms over different environments can compete in the ITC-2002 following the time rule.

Table 2: Hard constraints violated by the STF algorithms

*	Version of the Sort-Then-Fix algorithm														
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
01	4	0	1	0	3	0	1	0	0	1	0	1	0	0	0
02	0	7	0	0	0	5	0	0	1	3	0	4	1	0	0
03	2	0	0	0	1	0	0	0	1	0	3	4	0	0	0
04	15	3	7	7	11	3	8	3	15	7	16	12	7	4	8
05	10	1	2	3	10	2	2	2	8	7	8	6	7	3	7
06	4	0	6	0	6	2	5	1	8	10	6	9	10	0	5
07	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
08	4	0	3	0	4	0	2	0	1	1	0	3	1	0	0
09	3	0	2	0	4	0	3	0	2	3	2	1	1	0	1
10	3	2	0	0	3	0	0	0	4	4	4	4	0	0	0
11	2	9	2	4	2	9	2	5	2	2	4	1	2	4	2
12	4	0	5	2	4	0	5	1	6	3	6	3	3	2	3
13	6	0	3	0	5	0	4	0	4	3	5	3	4	0	7
14	0	0	0	0	2	0	1	0	4	0	3	1	0	0	0
15	3	0	0	0	4	0	1	0	3	0	3	0	0	0	0
16	1	8	0	3	2	8	0	3	1	0	1	0	1	3	1
17	10	1	1	1	10	2	1	1	13	10	13	10	10	3	10
18	0	2	1	0	0	1	0	0	0	1	1	0	0	0	0
19	3	0	3	0	5	0	3	0	2	2	2	2	0	0	1
20	1	0	2	0	1	0	0	0	1	0	1	1	0	0	0

***** Problem instance

Table 3: Hard constraints violated by the STF algorithms

*	Version of the Sort-Then-Fix algorithm													
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**
01	0	1	0	0	0	1	0	0	0	0	0	2	0	0
02	0	0	0	0	0	0	0	0	0	0	0	0	0	0
03	0	0	0	0	0	0	0	0	0	0	0	2	0	0
04	4	7	3	8	2	7	3	8	0	0	0	6	1	0
05	6	5	1	5	2	5	1	5	2	0	0	2	0	0
06	0	12	0	9	0	12	0	9	0	0	0	0	0	0
07	0	0	0	0	0	0	0	0	0	0	0	0	0	0
08	0	2	0	1	0	2	0	1	0	0	0	0	0	0
09	0	2	0	1	0	2	0	1	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	4	1	5	1	4	1	5	0	4	4	2	3	0	0
12	3	4	1	4	2	4	1	4	2	0	0	0	0	0
13	1	4	1	3	0	4	1	4	0	0	0	0	0	1
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	1	0	0	0	0	0	0
16	6	2	3	1	6	2	3	2	5	3	0	1	0	0
17	3	4	3	4	4	4	3	4	5	0	0	4	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	1	0	0	0	2	0	0	0	0	0	0
20	0	0	0	1	0	0	0	2	0	0	0	0	0	0

***** Problem instance

Table 4: Soft constraints violated by the STF algorithms

*	Version of the Sort-Then-Fix algorithm													
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
1	912	982	984	977	936	882	980	855	894	959	904	970	902	941
2	908	923	854	949	885	915	853	945	839	845	855	840	821	920
3	876	985	944	940	872	981	955	932	892	889	867	859	927	912
4	1155	1296	1234	1280	1219	1321	1192	1286	1154	1162	1154	1108	1254	1293
5	1310	1461	1314	1432	1326	1434	1336	1430	1299	1381	1277	1354	1364	1234
6	1357	1318	1325	1355	1284	1331	1287	1453	1256	1266	1202	1310	1405	1493
7	1687	1944	1871	1927	1754	1944	1871	1919	1608	1668	1756	1727	1864	1932
8	1050	1260	1077	1197	1057	1179	1102	1207	1054	1124	1065	1084	1123	1186
9	936	1035	910	1031	931	1088	906	1088	985	858	954	864	935	1055
10	874	947	905	910	886	923	905	888	863	863	863	863	997	898
11	940	1055	996	1056	923	1013	996	1065	997	941	960	929	958	1049
12	892	947	871	886	893	952	871	891	853	853	853	856	907	944
13	1067	1226	1073	1154	1051	1206	1061	1221	993	1030	996	999	1144	1174
14	1538	1814	1704	1727	1537	1632	1696	1855	1553	1611	1576	1679	1682	1752
15	1324	1612	1495	1527	1299	1556	1512	1565	1411	1415	1411	1416	1536	1538
16	914	1100	1035	1035	923	993	1037	1065	963	946	903	923	1009	991
17	1317	1374	1418	1395	1317	1376	1418	1395	1308	1286	1308	1278	1429	1418
18	918	999	876	960	915	930	872	990	854	902	838	849	906	978
19	1252	1375	1262	1382	1303	1347	1257	1387	1309	1311	1291	1291	1286	1329
20	1259	1403	1362	1412	1234	1361	1401	1399	1318	1351	1310	1305	1346	1389

***** Problem instance

Table 5: Soft constraints violated by the STF algorithms

*	Version of the Sort-Then-Fix algorithm														
	15	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**
1	915	973	911	958	936	982	911	958	950	989	985	1098	931	904	945
2	826	943	884	960	871	958	884	943	839	940	978	1053	864	938	948
3	913	908	952	935	933	979	952	947	952	998	956	1147	917	920	960
4	1221	1354	1245	1306	1228	1327	1245	1306	1228	1321	1273	1367	1217	1238	1263
5	1368	1300	1423	1344	1407	1297	1423	1344	1423	1291	1379	1460	1310	1402	1376
6	1426	1393	1268	1404	1241	1445	1268	1404	1254	1390	1528	1536	1313	1407	1377
7	1850	1902	1864	1932	1850	1902	1864	1952	1851	1949	1903	1680	1497	1806	1761
8	1112	1220	1108	1186	1108	1219	1108	1186	1111	1219	1219	1399	1049	1092	1128
9	955	1074	945	946	942	992	945	946	916	1002	1023	1114	925	956	973
10	997	909	997	898	997	909	997	898	997	909	964	954	934	968	968
11	956	1008	975	1005	1010	1009	975	1005	1022	1021	1049	1144	929	1014	1008
12	909	930	933	975	933	976	993	975	933	976	932	954	838	920	923
13	1136	1255	1141	1173	1130	1211	1141	1173	1141	1215	1116	1208	1131	1102	1065
14	1794	1782	1680	1769	1681	1774	1680	1769	1667	1795	1744	1796	1580	1673	1704
15	1518	1512	1464	1631	1478	1620	1464	1631	1446	1620	1605	1495	1259	1529	1501
16	997	991	965	1065	936	991	965	1065	967	1102	1045	1325	1016	971	1051
17	1429	1433	1431	1507	1405	1500	1436	1507	1436	1496	1445	1565	1283	1450	1456
18	903	1018	925	933	905	988	925	933	888	967	993	1094	838	908	924
19	1266	1288	1286	1352	1266	1322	1286	1329	1273	1363	1325	1518	1354	1295	1312
20	1375	1316	1396	1452	1410	1404	1396	1452	1407	1426	1332	1452	1227	1372	1401

***** Problem instance

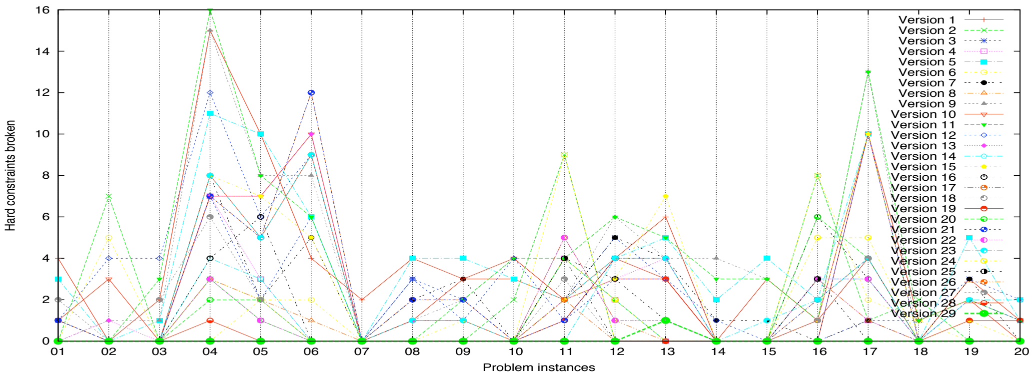

Figure 1: Hard Constraints Broken

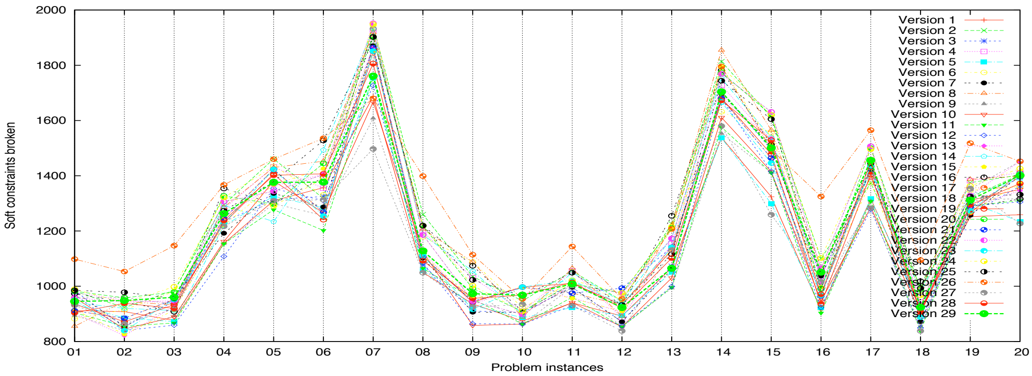

Figure 2: Soft Constraints Broken

The criterions included in the algorithms deal only with hard constraints, however it is possible to extend the algorithms to handle soft constraints. Table 6 shows the results of modifying algorithm 29. In algorithm 29, the search of the slots is in ascending order from 1 to 45; in the modified version algorithm 29', the search of slots is in ascending order from 1 to 8 then from 10 to 17 then from 19 to 26 then from 28 to 35 then 37 to 44 and finally in slots 9, 18, 27, 36 and 45. It is worth noticing that if a student takes lectures at the end of the day, i.e. in slots 9, 18, 27, 36 or 45, a soft constraint is violated. The number of violated hard constraints is the same in both versions, however the number of soft constraint violations is clearly reduced in the new version for all the instances.

In order to verify the results, the participants of the ITC-2002 provide the executable file of their implementations.[1]

[1]The STF implementations in source code are available by request via e-mail.

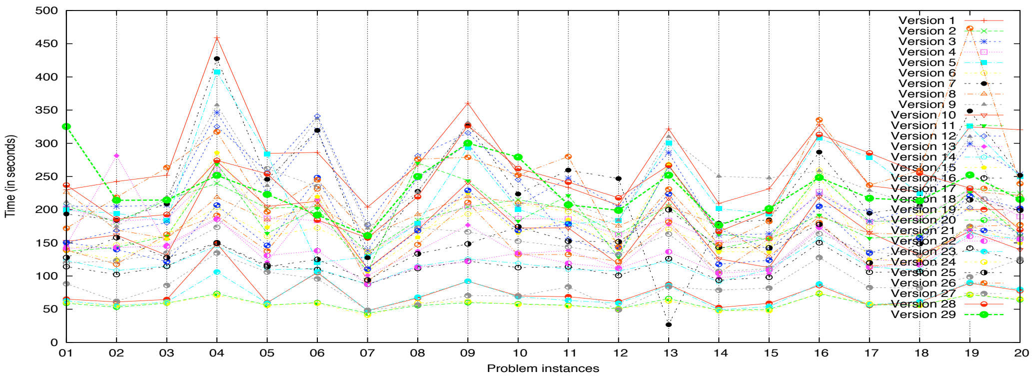

Figure 3: Time using Octave

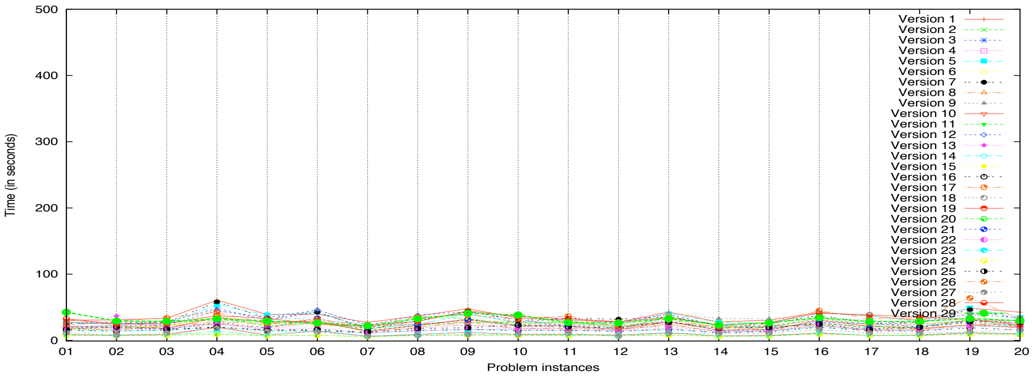

Figure 4: Time using Matlab

Table 6: Soft Restrictions Broken for algorithm 29' and 29

Instance	**Algorithm 29'**	**Algorithm 29**
1	700	945
2	633	948
3	628	960
4	1067	1263
5	1104	1376
6	949	1377
7	1167	1761
8	783	1128
9	715	973
10	603	968
11	696	1008
12	663	923
13	923	1065
14	1039	1704
15	911	1501
16	690	1051
17	1132	1456
18	611	924
19	1020	1312
20	908	1401

5 Conclusion

The family of STF algorithms, which appear to solve timetabling problems in a natural way, has been proposed. STF algorithms define a notion of the most constrained event, which is used to sort events before scheduling them. Unlike other approaches to the ITC-2002 contest, STF is deterministic and avoids swapping and backtraking.

We are confident that we will find an STF algorithm that will find feasible solutions for the 20 instances respecting the benchmark time. In this paper we focus on satisfying hard constraints only; soft constraints may be left unsatisfied. Nevertheless, as we showed in algorithm 29', with a slight modification we can considerably reduce the number of violated soft constraints in each one of the problem instances. We intend to find an STF algorithm that finds feasible solutions satisfying both hard and soft constraints for the 20 ITC-2002 instances.

As a main contribution, the STF algorithms can be used as a preprocessing step for other optimization algorithms. For example, it can be combined with a non-deterministic approach, such as a metaheuristic, to build a hybrid algorithm that can be more robust and can find a solution faster than a metaheuristic by itself. In fact, as far as we know all the references showing a solution to the problem only describe the metaheuristic used, but not the process to reach the initial feasible solution.

The metaheuristics used by the other participants of the contest do a pre-processing of the problem data before starting the search process, so all of the algorithms start with a different initial solution. To tackle this issue, the timetables generated by the STF algorithms can be used as an initial base solution to all of these metaheuristics, in order to measure the real effectiveness of each metaheuristic with respect to the others.

The STF algorithm finds a timetable in much less than the benchmark time available. We intend future versions of the algorithm to satisfy both hard and soft constraints to find complete solutions to the 20 ITC-2002 contest problem instances, within the time set by the benchmark program. Also, the STF algorithms will be extended and tested on the earlier instances of the International Timetabling Competition 2007, in order to manage the new constraints proposed in these instances.

Acknowledgments

We thank the Mexican Academy of Sciences (AMC) and the United States-Mexico Foundation for Science (FUMEC) for the support granted in 2009 under the Stays Summer Program.

References

[1] Rossi-Doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardello, L.M., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B., Paquette, L., Stutzle, T.: A comparison of the performance of dfferent metaheuristics on the timetabling problem. PATAT 2002, LNCS, **2740** (2003) 329–351

[2] Paechter, B., Gambardella, L.M., Rossi-Doria, O.: International Timetabling Competition Webpage (2003) (viewed January, 2010) (updated July 10, 2003), http://www.idsia.ch/Files/ttcomp2002/

[3] McCollumn, B.: International Timetabling Competition Webpage (2007) (viewed January, 2010) (updated October 1, 2008), http://www.idsia.ch/Files/ttcomp2002/

[4] Barry McCollum, Andrea Schaerf, Ben Paechter, Paul McMullan, Rhyd Lewis, Andrew J. Parkes, Luca Di Gaspero, Rong Qu, and Edmund K. Burke Setting the Research Agenda in Automated Timetabling: The Second International Timetabling Competition INFORMS JOURNAL ON COMPUTING 2010 22: 120-130, DOI: 10.1287/ijoc.1090.0320.

[5] di Gaspero, L., Schaerf, A.: Timetabling Competition TTComp 2002: Solver Description, (viewed January, 2010) (updated July 10, 2003), http://www.idsia.ch/Files/ttcomp2002/schaerf.pdf

[6] Socha, K., Knowles, J., Sampels, M.: A $\mathcal{MAX}-\mathcal{MIN}$ Ant System for the University Timetabling Problem. Algorithmic Number Theory. LNCS, **2369** (2002)

[7] Kostuch, P: The university course timetabling problem with a three-phase approach. PATAT 2004, LNCS, **3616** (2005)

[8] Frausto-Solis, J.F., Alonso-Pecina, F., Larre, M. Gonzalez-Segura, C., Gomez-Ramos, J.L.: Solving the timetabling problem with three heuristics. WSEAS Transactions on Computers. **11-5** (2006)

[9] de Werra, D.: An introduction to timetabling, European Journal of Operational Research, (1985)

[10] Gotlieb, H.: The construction of class-teacher timetables, IFIP Congress, (1963)

[11] Kostuch, P. Socha, K.:, Hardness prediction for the University Course Timetabling problem, Proceedings of the Evolutionary Computation in Combinatorial Optimization EvoCOP, (2004)

[12] Rossi-Doria, O., Blum, C., Knowles, J., Sampels, M., Socha, K., Paechter, B.:, A local search for the timetabling problem, PATAT (2002)

[13] Socha, K.: Metaheuristics for the Timetabling Problem. DEA Thesis, Universite Libre de Bruxelles, (2003)

[14] Andrea Schaerf. A survey of automated timetabling. Artificial Intelligence Review, 13(2):87-127, (1999)

[15] Chiarandini, M., Birattari, M., Socha, K., Rossi-Doria, O.: An effective hybrid approach for the university course timetabling problem. Journal of Scheduling, **9** (2006) 403–432