Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007)

Construction of Initial Neighborhoods for a Course Scheduling Problem Using Tiling

D. Moody, A. Bar-Noy and G. Kendall, *Member, IEEE*

***Abstract*—A recent competition course scheduling competition saw many solution approaches which constructed an initial solution, and then improved that solution using local search. The initial solution appears to be crucial for the local search to be effective and in this work we propose a tiling technique which can quickly construct a solution which we hope can be used as a good starting point for a local search procedure.**

I. Introduction

The Metaheuristics Network sponsored an International Timetabling Competition in 2003 [12], involving a course scheduling problem. Several competitors provided solution approaches, all of which relied upon the establishment of an initial neighborhood, followed by a swapping phase. The focus of those solutions is on the swapping phase, which used techniques such as simulated annealing, tabu search and the Great Deluge algorithm. The creation of the initial neighborhood was done randomly or employed a straightforward prioritizing mechanism. Our work centers on a construction algorithm, utilizing a tiling technique, which provides a feasible solution which can serve as a solid base for the swapping phase. Our approach creates the initial solution within a few seconds, enabling the overwhelming majority of processing time to be dedicated to the swapping phase.

II. Problem Definition

The competition problem is a sample university course timetabling problem. It consists of a set of events to be scheduled in 45 timeslots across five days and nine periods. Each event must take place in one of a set of *n* rooms provided for each instance. Each event is attended by a set of students. Rooms are constrained by room size and features, eliminating the possibility of certain events from being held in a given room. Other hard constraints of the problem include students only attending one event on a given day and time period, and only one event scheduled per day and period in a room. Meeting all hard constraints for each event constitutes a "feasible" solution.

The problem also contains soft constraints. Violating a soft constraint leads to a penalty and the quality of a solution is given by the summation of all penalities. Hence, solutions with the least penalties represent the best solutions. The soft constraints causing penalties are:

- a student has a class in the last slot of the day;
- a student has more than two classes consecutively;
- a student has a single class on a day

A review of the submitted solutions shows that most approaches were in at least two phases. Typically, the first phase was to build an initial feasible solution. The second and subsequent phases performed some form of search algorithm looking to swap previously scheduled events. Burke and Newall describe this evolutionary approach in [2].

The method of creating the initial solution varied as well as the search algorithms; at least in the top four solutions. Kostuch [10] had the best set of results. The approach taken was to place events into timeslots, and then attempt to assign rooms. If a feasible solution could not be obtained, then the algorithm randomly unscheduled a set of events and tried again. Twenty-five runs with different seeds were used to achieve the best feasible solution. Burke and Bykov[3] and Bykov[4] used a modified Brelaz (saturation degree) algorithm. Events were analyzed for the number of timeslots available. Events with the lowest number of timeslots were scheduled first. Shaerf [7] used a random placement of events for the initial feasible solution, while Courdreau [6] did not create a feasible solution before the swapping phase. This approach relies on the swapping phase to not only improve the solution but also develop a feasible solution.

III. Our Approach

Our approach is a constructive algorithm, which does not incorporate any steps depending upon random value input. The algorithm builds the schedule event by event in an ordered fashion. If a timeslot is not available, then a backtracking procedure is performed, unscheduling and re-scheduling events, until the event can be placed.

One aspect of this problem is an event's contribution toward the penalty count cannot always be fully determined until all the timeslots in the same day as the event are scheduled. There is always the possibility that another event placement will cause this event to be the only event of the day for a student, or be one of three or more consecutive classes. This characteristic of the problem reduces the

Manuscript received November 4, 2006.

Douglas L. Moody (email: dmoody@citytech.cuny.edu) and Amotz Bar-Noy (email: amotz@sci.brooklyn.cuny.edu) are with the City University of New York Graduate Center, Department of Computer Science. Graham Kendall is with the School of Computer Science & IT, University of Nottingham, UK (email:gxk@cs.nott.ac.uk).

effectiveness of the construction algorithm approach, since the partial solution value cannot be obtained adequately.

However, the constructive approach has relevance in producing the initial solution, which a local search algorithm can operate upon. We seek a good quality initial solution that can be found in a few seconds, and provides a good starting point for the search algorithm. Cormen discusses building upon initial solutionss in [5].

We use "tiling" as a means to create the initial neighborhood in a constructive manner. Bar-Noy and Moody [1] demonstrated that tiling was an effective approach for the traveling tournament problem to quickly develop a neighborhood within ten or less percentage points of the best known solutions. The tiles for that problem were a set of games that were to be scheduled sequentially within a team's overall schedule. Tiles were placed in a team schedule, as long as hard constraints were not violated with tiles in other teams' schedules. Kingston [8,9] also used tiles for set of classes in a high school scheduling environment.

For our problem, a tile represents a set of events that will be scheduled in the same day and in period x and period x+1, where x the number of periods available without penalty (8 in the competition). The events in the tile relating to the same period, must not share any students, since this would violate the hard constraint of a student attending two events in one timeslot. Additionally, the tile provides a "break" in the student's schedule. Since no student attends more than one event in a tile, the student has a break between period x and period x+1. After the tiles are created and placed, remaining events are placed and backtracking employed if necessary.

The tiling approach can also be viewed as creating the "macro event" defined by McCollum [11]. The macro event is a collection of events that can be associated together. This association may be because a student would take them as a block, or in our case, that the events are independent of each other. The ability to recognize relationships between events (beyond the obvious constraints) is key for tile or macro event usage.

IV. PREPROCESSING STEPS

The Metaheuristics Network competition problem is presented by a series of files that indicate the following relationships:

- Rooms to Features
- Features to Events
- Students to Events

By transitivity, rooms to events can easily be calculated. This information can be analyzed to determine the key relationship in the problem – event to event conflict due to students. The event-event relationship is helpful in two ways. We give a value to the relationship equal to the number of students required to attend both events. For hard constraints, the relationship specifies which events can share timeslots that have the same day and period. All event-event relationship values between all events in the same day and period timeslot must be zero. For soft constraints, the relationship provides input on the potential violation of the three or more consecutive classes for a student. If two events have an event-event relationship that is positive and the events are scheduled in consecutive timeslots, these events may lead to a soft constraint violation, given the events scheduled around the pair.

The final step of the preprocessing phase is to calculate the degrees of each event. The degree of an event is the number of room, period combinations that are possible for the event. Initially, this value is the number of time periods (45) multiplied by the number of allowable rooms for the event, since each event can be placed in any timeslot, if no other events are scheduled. In the next phase as events become scheduled, the degrees for events will decrease.

V. CREATING THE TILES

Prior to placing any events, a set of "tiles" is created from the information of the instance. A tile is a set of up to $2n$ (n is the number of rooms) events in the instance. In the Metaheuristics Network competition, this value was usually ten. Considering the events of a tile in two sets of size n, the first set was simply a set of events that could co-exist in the same day and time period. Searching by lowest event degree, the first n events that had an event-event relationship of zero were chosen for the tile. Also the event cannot have already been assigned to another tile. After the first set of events for the tile was chosen, a second set was selected. Again, considering remaining events in lowest degree order, events were chosen that had zero event-event relationship with all events in the first set, and the events already assigned to the second set. The second requirement of an event in the second set is similar to the first set's requirements. However the cross-set event-event relationship number guarantees a student taking an event in the first set of the tile, will not have a class in the second set. Hence the student taking any event in the tile has a "break" in his or her daily schedule. This break will help reduce the possibility of receiving penalties for the consecutive classes for a student constraint. Figure 1 illustrates this point.

This information can be analyzed to determine the key relationship in the problem – event to event conflict due to students. The event-event relationship is helpful in two ways. We give a value to the relationship equal to the number of students required to attend both events. For hard constraints, the relationship specifies which events can share timeslots that have the same day and period. All event-event relationship values between all events in the same day and period timeslot must be zero. For soft constraints, the relationship provides input on the violation of the three or more consecutive classes for a student. If two events have an

event-event relationship that is positive and the events are scheduled in consecutive timeslots, these events may lead to a soft constraint violation, given the events scheduled around the pair.

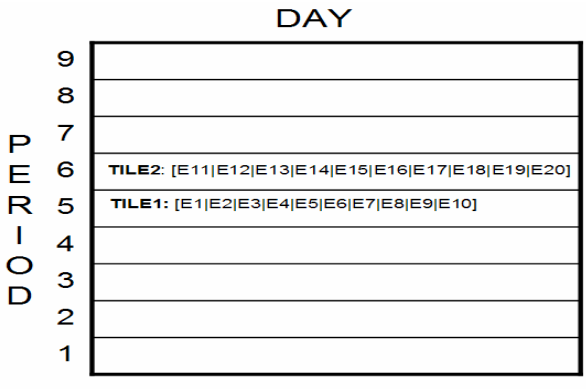

Fig. 1. Tiles slotted in periods 5 and 6

Tiles placed in periods 2, 4 and 6 in each day would provide a solution with zero soft constraint violations for consecutive student classes during the first 8 periods.

The requirement for the creation for a tile is a set of events that cover all rooms for two time periods ($2n$), each of which has a zero value for the event-event relationship. With 200 students, 400 events and an approximate average of 17 events per student for most instances, it was difficult to create a sufficient number of tiles for most instances. The requirement was relaxed to allow an overlap of up to two students between an event in the first set of the tile and the second set. This does not necessarily guarantee a soft constraint violation, as the preceding event to the tile, or the subsequent event, may not match the students appearing in events within both sets of the tile. For all instances we were able to create at least ten tiles. Some tiles were created with fewer than n (number of rooms) events in each set due to event-event relationships.

Once all tiles are created, they are placed in specific timeslots within the schedule. Unlike the traveling tournament problem addressed by Bar-Noy and Moody[1], the placement of the tiles are independent of each other. In the traveling tournament problem, tiles consisted of set of games. Placement of a tile could conflict with previously scheduled games. Bar-Noy and Moody used a backtracking mechanism to move the tile in this situation. Within our problem, the placement of tiles in unique timeslots can not cause a hard constraint violation.

For each instance our approach generated ten tiles. The tiles were then placed in the schedule as shown in figure 2.

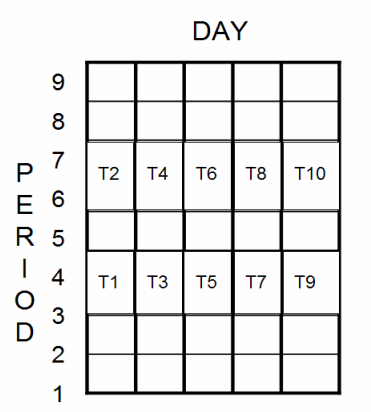

Fig. 2. Tile placement within the schedule

We selected the above tile placement to minimize the number of students violating the "taking more than two classes consecutively" constraint. A tile placed at period x represents a break in all student schedules for that day between period x and x+1. Placing the tile at period 3 assures us that only students with classes in the first 3 periods could be assessed a penalty point, as the tile creates a break between period 3 and 4. Placing the two tiles in periods 3 and 6 ensures that a break occurs every three periods (including the ninth period if needed). We did not explore the concept of moving tiles to different periods. Placing the tiles closer together would provide a longer time period block where absolutely no breaks could occur, while leaving a time period block (possibly up to 5 periods), where consecutive classes could be scheduled. In the latter scenario, students could have more than 3 classes consecutively, while in our approach, we have limited the number of consecutive classes for a student to 3.

T1 represents the first tile created, which would contain events with the lowest event degrees. As tiles are scheduled, the degrees of the remaining events are changed. The tile is placed first, and then the event pool is analyzed for next set of events for the subsequent tile creation.

VI. Placing the Remaining Events

After the ten tiles are placed, the remaining unscheduled events, must be placed in the schedule. The construction and backtracking method is used to place these events with the algorithm shown in figure 3.

```
While (events remain)
  set schedule_flag to false
  Select event with lowest event degree
    set event timeslot to last timeslot of this event
    While (event day <= 5 and event period <= 8)
    Advance event timeslot to next day or period
        Check event conflicts
      If no event conflict then
        Make temp event placement in schedule
```

```
            Analyze  event degree of remaining events
               If event degree of any event is negative
                  set schedule_flag to false
                 else
                    set schedule_flag to true
                    make permanent event placement
                   push event onto schedule_stack
          if schedule_flag is false
             Advance event timeslot to next day/period
       end while
       If schedule_flag is false repeat for period =9
    If schedule_flag is false then
       pop event from schedule_stack
        Advance event timeslot to next day or period
      end select
end while
```

Fig 3. Event Placement with Backtracking

The algorithm attempts to schedule an event in the period 1 to 8. If this is not successful, then the event is considered for period 9. If this also unsuccessful, the stack of scheduled events (including those scheduled through tiling) is popped. The popped event is advanced (adding one to the period of timeslot consideration) and re-scheduled.

An event with a negative event degree indicates that there are no room / timeslot assignments that satisfy the event. In this case, backtracking is performed.

The events are always selected by the lowest event degree. These events have the least flexibility of timeslot and room assignment, at this point in the scheduling process.

Only two instances required more than 200 backtracks. Hence the algorithm produced a feasible solution with tiles in a few seconds.

VII. Results

We present our results in figure 4. We look at three major criteria – how did the solution improve with tiling, how close is our neighborhood to the best solutions, and does our approach support the next search phase.

Instance	Tiles	Initial Neighorhood	Simple Swapping	Number of Swaps	%tile improve	%swap improve	Kostuch	%diff
20	0	956	915	30		4.29%		
	10	692	645	18	27.62%	6.79%		
19	0	988	906	43		8.30%		
	9	904	864	16	8.50%	4.42%		
18	0	801	601	99		24.97%		
	10	570	497	43	28.84%	12.81%		
17	0	1023	932	99		8.90%		
	10	927	846	80	9.38%	8.74%		
16	0	764	706	37		7.59%		
	10	606	570	59	20.68%	5.94%		
15	0	979	849	15		13.28%		
	10	821	746	36	16.14%	9.14%		
14	0	1113	1070	21		3.86%		
	10	937	864	21	15.81%	7.79%		
13	0	842	795	49		5.58%		
	10	822	753	95	2.38%	8.39%		
12	0	818	734	44		10.27%		
	7	700	614	59	14.43%	12.29%		
11	0	702	649	43		7.55%		
	10	615	582	38	12.39%	5.37%		
10	0	737	630	113		14.52%	516	5.04%
	10	542	513	43	26.46%	5.35%		
9	0	704	656	29		6.82%	502	18.73%
	10	596	578	27	15.34%	3.02%		
8	0	788	692	85		12.18%	646	6.66%
	10	689	605	54	12.56%	12.19%		
7	0	1164	1116	38		4.12%	913	3.83%
	8	948	873	38	18.56%	7.91%		
6	0	1044	1027	16		1.63%	771	-2.08%
	10	755	748	17	27.68%	0.93%		
5	0	1282	1027	16		19.89%	900	-2.22%
	9	880	740	17	31.36%	15.91%		
4	0	1090	1078	17		1.10%	734	24.11%
	10	911	890	16	16.42%	2.31%		
3	0	738	677	21		8.27%	491	14.26%
	10	561	535	23	23.98%	4.63%		
2	0	641	584	38		8.89%	444	9.91%
	10	488	443	22	23.87%	9.22%		
1	0	711	661	42		7.03%	515	14.17%
	9	588	534	42	17.30%	9.18%		

Fig. 4. Results

The table in figure 4 shows the solution value for the initial neighborhood created using no tiling (zero tiles) for the set of tiles as described above. The "Simple Swapping" column represents the solution value after a rudimentary algorithm is employed. The simple swapping algorithm identifies the most expensive event in terms of penalty. For example, the third consecutive event for a student would be assessed the penalty point. The most expensive event is compared with the next most expensive to see if a swap would improve the solution. The comparison is done 10,000 times and the number of resulting swaps is shown in the fifth column.

The first improvement column shows the benefit of performing tiling over a straight forward construction algorithm. The second column shows the impact of the simple swapping phase. This column is provided to show the relationship between the initial neighborhood phase and the swapping phase. Similar improvement percentages indicate that the tiling does not inhibit success during the swapping phase.

The last two columns provide information from Kostuch, who had the best results in the competition. This is the only competitor who provided information on the value of the initial neighborhood. Other submissions did not mention this value, or did not achieve a feasible (and hence scoreable) solution in the initial neighborhood. Kostuch's initial neighborhood value, achieved by taking the best of 50 seeds

is shown in column 8 and is compared to our tiling approach in column 9 .

VIII. CONCLUSIONS

The key result of the tiling approach is the positive impact of tiling on the solution score. The percentage improvement from adding the tiles averages 18.48%, with some instances reaching nearly a 1/3 improvement. This improvement is gained with negligible processing time. Our results involve one run versus the "best" of a series of runs.

The second point is whether the construction of our neighborhood prohibits the next phase of swapping from being effective. A comparison of the %swap improvement column shows nearly identical improvement rates before and after tiling. Only two instances (18,10) favor the initial neighborhood before tiling by more than 3%. Hence the tiling provides a solid base for other swapping algorithms.

The final comparison is between the solution value of our approach to that of Kostuch. In two instances our result beat that approach's initial neighborhood, which is the best of 50 runs using different seeds. The average difference across all instances was 8.53%. While we would have liked to provide a better solution in all instances, our approach yields a relatively close solution, without the processing time involved in Kostuch's approach. This processing time appears to average 55 seconds per instance. This would need to be multiplied by 50 for the different seeds yielding almost 45 minutes processing time to achieve an initial neighborhood approximately 8% better than our approach. This 8% differential could be resolved within the swapping phase, given the extra processing time saved in our approach.

IX. FUTURE WORK

An obvious extension of our approach is to investigate search algorithms and incorporate them into the methodology. This would further justify our approach in creating the initial neighborhood.

Also, we can investigate the creation of the tile process to look for methods to include more events in the tiles, and possibly utilize more than ten tiles. The problem could support using 3 tiles in a day, placing the tiles in periods 2,3; 4,5; and 6,7. Also, the selection of the events to be the starters for the tile could be analyzed more fully to see if a more intelligent approach would yield better tiles.

Finally, the research will continue to focus on various problems and identifying when and how the tiling approach can add to the solution process. The tiling approach holds the potential to aid in the construction of timetabling solutions efficiently, with the ability to handle large instances.

REFERENCES

[1] Bar-Noy, A. and Moody, D. "A Tiling Approach for Fast Implementation of the Traveling Tournament Problem", Practice and Theory of Automated Timetabling (PATAT06, Brno, August 2006), Conference Proceedings, pp. 351-358.

[2] Burke, E.K. and Newall, J.P. "A Multi-Stage Evolutionary Algorithm for the Timetable Problem", the IEEE Transactions on Evolutionary Computation, Vol. 3.1. pp.63-74

[3] Burke, E.K. and Bykov, Y. "Solving Exam Timetabling Problems with the Flex-Deluge Algorithm", Practice and and Theory of Automated Timetabling (PATAT06, Brno, August 2006), Conference Proceedings, pp. 351-358.

[4] Bykov, Y. "The Description of the Algorithm for the International Timetabling Competition", International Timetabling Competition Results. 31. March 2003. http://www.idsia.ch/Files/ttcomp2002/bykov.pdf.

[5] Cormen , T. et al. , Introduction to Algorithms, Second Edition. Boston: McGraw-Hill, 2001. pp. 1022,1054

[6] Courdeau, J.F. et al., "Efficient Timetabling Solution with Tabu Search", International Timetabling Competition Results. 31. March 2003. http://www.idsia.ch/Files/ttcomp2002/jaumard.pdf.

[7] Di Gaspero, L. and Schaerf, A. "A Multineighbourhood Local Search Solver for the Timetabling Competition TTCOMP 2002, Practice and and Theory of Automated Timetabling (PATAT04, Pittsburgh, August 2004), Conference Proceedings, pp. 475-478

[8] Kingston, J. "Hierarchical Timetable Construction", Practice and Theory of Automated Timetabling (PATAT06, Brno, August 2006), Conference Proceedings, pp. 196-208.

[9] Kingston, J. "A Tiling Algorithm for High School Timetabling", Practice and Theory of Automated Timetabling (PATAT04, Pittsburgh, August 2004), Conference Proceedings, pp. 233-250.

[10] Kostuch, P. "The University Course Timetabling Problem with a 3-Phase Approach", Practice and and Theory of Automated Timetabling (PATAT04, Pittsburgh, August 2004), Conference Proceedings, pp. 251-266.

[11] McCollum, B. "University Timetabling: Bridging the Gap between Research and Practice", Practice and Theory of Automated Timetabling (PATAT06, Brno, August 2006), Conference Proceedings, pp. 15-33.

[12] Paechter, B. "International Timetabling Competition". Metaheuristics Network. 31 March 2003.