The KTS High School Timetabling System

Jeffrey H. Kingston

School of Information Technologies
The University of Sydney, NSW 2006, Australia
http://www.it.usyd.edu.au/~jeff
jeff@it.usyd.edu.au

Abstract. KTS is a web-based software system for solving high school timetabling problems, freely accessible on the Internet. This paper describes KTS, including its data model, user interface, and solver. The solver uses operations research models in a polynomial-time heuristic framework to produce high quality solutions in a few seconds. Results are presented for six instances taken from Australian high schools.

1 Introduction

Research into automated timetabling has had many successes in recent years. Examination timetabling and university course timetabling have yielded to meta-heuristic methods, as the proceedings of recent PATAT conferences [1, 2] show.

High school timetabling has had much less success [3], probably because it is dominated by hard constraints, to which meta-heuristics seem less well suited [8]. There may also be non-technical reasons, such as fewer researchers in the field and less ready access to data.

KTS is a web server for high school timetabling created by the author. Its web interface puts the system on the desk of the timetable planner, and its polynomial time heuristic solver delivers a very good timetable in a few seconds. Together these features support non-traditional requirements such as rapid evaluation of alternative scenarios and incorporation of late changes, as well as the traditional one of solving a fixed instance to near-optimality. The system is fully operational and available continuously on the Internet [6].

This paper is a general overview of the KTS system. Section 2 presents a detailed specification of the high school timetabling problem as defined by KTS. Section 3 describes the user interface. Section 4 describes the solver, and Section 5 presents results for six instances taken from Australian high schools.

2 Data model

The KTS data model is object-oriented. It is described in this section, with a few minor omissions.

An *account object*, or just *account*, represents one user's account with the KTS system. Each account contains any number of *institutions*, representing

E. K. Burke, H. Rudová (Eds.): PATAT 2006, pp. 181–195. ISBN 80-210-3726-1.

educational institutions for which the user wishes to construct timetables. Each institution contains any number of *instances*, each representing that institution's timetabling problem for a particular year, or semester, etc.

Each instance contains a *time group* object, holding all information about time. KTS has a simple time model in which time is divided into individual *times* of equal duration, ordered chronologically, with each time optionally separated from the next by a break, which could be a meal break or the end of a day, etc. The full sequence of times is called the *cycle*.

A sequence of one or more times that follow each other chronologically and do not span a break is called a *time block*. Any set of times may be viewed as a set of time blocks, by grouping the times into blocks of maximal size. The sizes of these blocks, written as a sequence of integers, form the *block structure* of the set of times. For example, the set of times $\{Mon1, Mon2, Tue5, Tue6, Thu3\}$ presumably has block structure 2 2 1. The order in which the elements of a block structure are written does not matter; non-increasing order is used by convention. Meetings may specify that their times should have a particular block structure.

In addition to the instance's set of available times, the time group contains any number of *time subgroups*, which are subsets of the times, used when defining workload limits and *time conditions*. These latter place requirements on the sets of times assigned to meetings, and are either *limit conditions*, which limit the number of times from a given subgroup that a meeting may contain, for example limiting to 1 the number of undesirable times, or *spread conditions*, which require the time blocks assigned to a meeting to be spread evenly over a sequence of time subgroups, such as the days of the week.

An instance also contains any number of *resource group* objects, representing collections of *resources* (participants in meetings). Although not mandatory, there would typically be three resource groups, called *Student Groups*, *Teachers*, and *Rooms*. KTS is intended for high school timetabling problems, in which groups of students are timetabled, not individual students.

A resource group may contain *divisions*, representing administrative units such as faculties or departments (for teachers) and forms or years (for students). If a resource group has divisions, then each of its resources lies in exactly one of those divisions.

A resource group may also have *capabilities*, which are subsets of its set of resources. For example, an *English* capability would be the subset of teachers qualified to teach English; a *ScienceLab* capability would be the subset of rooms in which Science classes may be held. A resource may lie in any number of capabilities, and a capability may contain any number of resources. A division is usable as a capability, as is the resource group as a whole.

Each resource may have a set of times when it is unavailable to attend classes. It may also have *workload limits*, which might specify, for example, that the resource may attend meetings for at most 30 times over the cycle, and at most 7 times on each day. A limit may be placed on the number of occupied times in any subset of the times of the cycle, defined by a time subgroup. Each limit may

have a *hard component*, a number of times which must not be exceeded, and a *soft component*, for which violations are penalized but not prohibited.

One resource may *follow* another. For example, a room may follow a particular teacher, meaning that it is considered first when assigning room selections in meetings to which that teacher is assigned. Such a room is often called the *home room* of a teacher.

An instance also contains *meetings*, which specify that certain resources are to meet together at certain times.

A meeting's times are specified by a single *time selection*, which requests that a particular number of times be assigned. It may request that the times conform to a given block structure, and include preassigned times. All time conditions defined in the time group apply to all time selections, as far as the time selection's block structure and preassigned times allow.

A meeting's resources are specified by any number of *resource selections*. For example, a meeting in which class *7A* studies Science might contain a *Student Groups* resource selection requesting student group *7A*, a *Teachers* resource selection requesting one teacher with the *Science* capability, and a *Rooms* selection requesting one *ScienceLab*. A resource selection may include preassigned resources.

An instance may contain any number of *solve profiles*, which are named collections of options for controlling the solver. The solver may be invoked with this set of options by a single click on the appropriate link. An instance may also contain any number of *display profiles*, which are named collections of options describing a timetable display or print: whether to use HTML, PDF, or PostScript; whether to display large planning timetables or individual resources' timetables; whether to display the whole timetable, or just one division or resource; and so on. Again, one click produces a display using these options.

An instance may also contain a current solution. This consists of assignments of particular times and resources to some (hopefully all) of its time and resource selections. A resource assignment may be a *split assignment*, in which one qualified resource is assigned for some of the times of the meeting and a different one to the remaining times; or it may be a *partial assignment*, in which a particular resource is assigned for some of the times of a meeting but there is no assignment for the remaining times.

KTS objects are persistent: they exist permanently on disk, but can be updated in memory while the system is running. They are stored externally in UTF-8 text files, updated by a two-phase algorithm which protects against accidental corruption. Each account and its institutions occupies one file, and each instance occupies one file, including all the instance's objects (typically 10 to 20 kilobytes of data). Most operations concern a single instance, and they begin by reading this file and end by writing it. Instances are represented using a simple specification language, also called KTS, which is a descendant of the well-known TTL language [4]. The user may upload and download KTS instance files, although there is no strong motive for doing so.

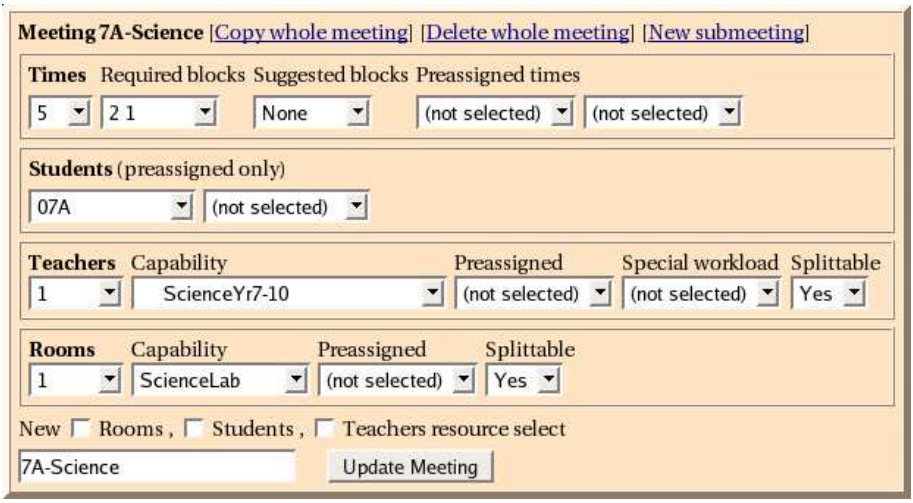

Fig. 1. Screen shot of the user interface to one small meeting. A page header and navigation links precede this box and are not shown here. After the header line, the first inner box holds the time selection, here requesting 5 times including block structure 2 1. The next box holds a Student Groups resource selection, requesting student group resource 07A. This box accepts preassignments only, in accordance with an option set on the Student Groups resource group page. The following boxes request one ScienceYr7-10 teacher and one ScienceLab room. Split assignments are usually allowed; the Splittable boxes let the user disallow them for individual resource selections. Teachers have workload limits, so the Teachers selection offers a Special Workload box which allows the workload associated with this selection to be reduced (e.g. to 0 for staff meetings).

3 User interface

The KTS system is not distributed to users for installation on their own systems. Instead, there is a unique copy running on a server at the author's institution, publicly accessible via the web, using HTML and CGI for its user interface. This has several advantages: it makes KTS available instantly on any computer connected to the Internet; the software may be upgraded centrally at any time; and the data is held on the server where it may be captured for research purposes, in accordance with an agreement that users enter into when they create their accounts.

The user interface has one page for each object, beginning with a header and some navigation links, and continuing with updatable displays of the object's attributes. Most pages contain paragraphs of text describing their fields, so are self-documenting. The exception is the page which displays a meeting (Figure 1), where there is too much detail to document on the spot. Instead, a set of examples of meetings of increasing complexity is offered, which shows step-by-step how each meeting is built up. There is also an overview document explaining the capabilities of the system, and a glossary.

Bankstown Girls High School, 1998	Satisfactory		Unsatisfactory		Total	
	No.	%	No.	%	No.	%
Assignments to time selections						
Required blocks	147	96.7	5	3.3	152	100.0
Suggested blocks	152	100.0	0	0.0	152	100.0
At most 1 Undesirable time	152	100.0	0	0.0	152	100.0
Even spread through Day 1 .. Day 5	120	78.9	32	21.1	152	100.0
Assignments to resource selections						
Rooms	229	97.9	5	2.1	234	100.0
Students	316	100.0	0	0.0	316	100.0
Teachers	408	92.3	34	7.7	442	100.0
Soft workload limits						
Teachers	266	95.0	14	5.0	280	100.0

Fig. 2. Screen shot of the summary table from the evaluation page. Each underlined number is a link leading to a detailed list of defects. Below this table are other tables giving an intermediate level of detail, such as the number of time conditions defects affecting each student form, the number of soft workload overloads per teacher, etc.

When there is a solution, KTS offers an evaluation page summarizing its defects (Figure 2), with links to more detailed evaluations. The most interesting of these detects sets of resource slots that cannot all be assigned to, owing to a shortage of resources (Figure 3).

Entry of a complete instance takes some hours. Short-cut operations for creating a time group and the usual three resource groups help somewhat, as do operations for copying resources and meetings. There is also an operation for copying a complete instance, which saves time when moving to a new year or semester.

4 The solver

The KTS solver aims to produce a very good and comprehensible timetable in ten seconds or less. It has five stages: *column layout*, *tile construction*, *time assignment*, *time adjustment*, and *resource assignment*. The basic approach appeared in an earlier paper by the author [5], but the present work describes a completely rewritten solver, with more and better results.

The following five subsections describe the five stages. Some details have been omitted, since a full description would be too lengthy for this paper, which aims to present a balanced view of the whole system.

Hall Set 2

The resource selections in this Hall set are:

Meeting	Submeeting	Capability or resource	Required time(s)	Tixels
7CKO-D&T12-Art34	7CKO3-2	VisualArtsYr7-10	Mon5	1
7CKO-D&T12-Art34	7CKO4-2	VisualArtsYr7-10	Mon5	1
11-2/12-1	12-1-VisualArts	VisualArtsYr11-12	Mon5	1
Total tixels demanded				3

To satisfy this demand, the following tixels of supply are available:

Resource	Time(s)	Tixels
Diamond	Mon5	1
Leeon	Mon5	1
Total tixels supplied		2

Subtracting supply from demand gives 1 unassignable tixel in this Hall set.

Fig. 3. Screen shot of a detailed evaluation, showing that a set of three simultaneous Art classes cannot all be assigned teachers, because there are only two Art teachers. The analysis is based on finding the Hall sets of a bipartite matching between all the tixels demanded by the instance and all the tixels supplied (a *tixel* is one resource at one time). Two versions of this analysis are carried out, one before time assignment and one after. Hall sets can be much more complex than this very simple example; they might reveal that the supply of English and History teachers, taken together, is insufficient to cover all the English and History classes even before time assignment, and so on. KTS merely prints the Hall sets; the user must find the explanations.

4.1 Column layout

As far as possible, the meetings in a high school timetable should overlap exactly in time, or not at all. This makes the timetable comprehensible, and simplifies resource assignment.

KTS's method of achieving such regularity begins by dividing the cycle into *columns*: sets of times which make good choices for assigning to meetings, and which meetings are encouraged to use wherever possible. The reader may be familiar with this approach from its use in North American universities, where the columns Mon-Wed-Fri 9-10am, Mon-Wed-Fri 10-11am, and so on, are frequently used. A traditional column plan in Australian high schools divides a cycle of 40 times into six columns each with six times, and one column with four times preassigned those times when the whole school attends Sport and optional religious instruction.

There is no requirement that meetings fit exactly into columns. In the senior years they usually do, but in the junior years the school offers many small subjects, often with little resemblance to any column plan.

	Day 1	Day 2	Day 3	Day 4	Day 5
Time 1	Column 1	Column 6	Column 2	Column 2	Column 5
Time 2	Column 1	Column 6	Column 2	Column 2	Column 5
Time 3	Column 6	Column 3	Column 4	Column 3	Column 3
Time 4	Column 6	Column 3	Column 4	Column 3	Column 6
Time 5	Column 5	Column 2	Column 1	Column 4	Column 4
Time 6	Column 5	Column 5	Column 1	Column 4	Column 7
Time 7	Column 4	Column 1	Column 5	Column 6	Column 7
Time 8	Column 2	Column 7	Column 3	Column 1	Column 7

Fig. 4. A typical layout of a week of 40 times into six columns of width 6 plus one of width 4. Breaks are not shown, but occur after the fourth and sixth times each day except Friday, when they occur after the third and fifth. This diagram was generated in PostScript by KTS.

Although a column plan could easily be inferred from the time selections of the meetings, it is such a basic part of the timetable planner's thinking that it seems better to have the user enter it, including a number of times, block structure, and optional preassigned times for each column. Given this plan, the solver's first task is to assign specific times to each column, aiming to ensure that each column satisfies the time conditions, so that meetings assigned to them will do so. An example of such a *column layout* appears in Figure 4. Producing it is quite easy in practice. The solver does it in two steps.

First, the time blocks naturally present in the cycle (between one break and the next) are partitioned into smaller blocks whose sizes exactly match the complete set of block sizes of the columns. KTS does this heuristically, checking after each break that the columns' block sizes can be packed into the current cycle breakdown, and with an eye to the time conditions defined by the user: if meetings should be spread evenly over five days, then the solver aims to have the same number of time blocks on each day, and so on. Blocks of preassigned times already present in meetings are used wherever possible.

Second, the time blocks created by breaking down the cycle's blocks are assigned to columns. After an initial round-robin assignment, a simple hill climber swaps pairs of equal-width time blocks between columns until no swap exists that reduces the badness of the columns as measured against the time conditions.

4.2 Tile construction

KTS continues its efforts to build a regular timetable by first timetabling small sets of meetings together into larger entities called *tiles*.

Figure 5 contains two examples of tiles. The students are grouped by ability for Mathematics, so the five Mathematics classes must run simultaneously

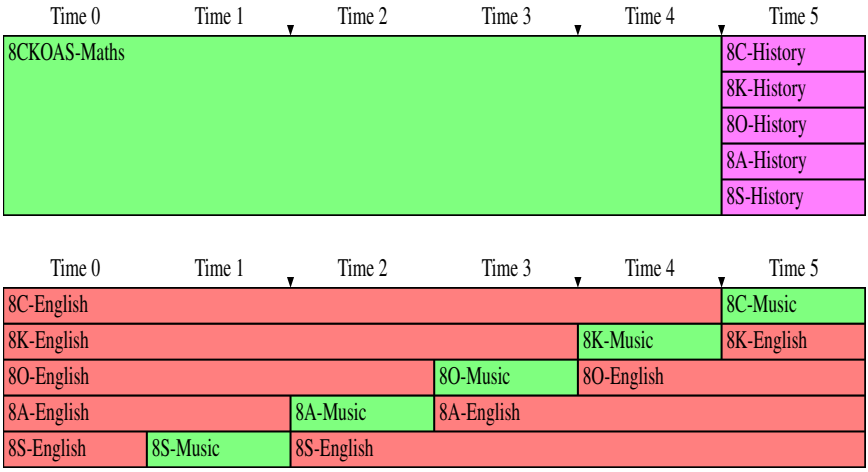

Fig. 5. Two examples of tiles from the *bghs98* instance. Each row is the timetable of one student group resource; each column is one time. The wedges indicate block structure.

and are combined into one large meeting in the input data. The adjacent History meetings do not have to run simultaneously, but fitting them neatly alongside Mathematics forces them to. The second tile illustrates a construction, well known to manual timetablers, called the *runaround*. There are only two Music teachers and two Music rooms, so the five Music classes cannot run simultaneously. By interleaving them among other meetings as shown, the tile demands only one of each at any one time.

Tiles are built in three steps. First, the meetings of each student form are grouped into *buckets*. Any meeting containing all the form's student group resources goes into a bucket by itself; meetings which are identical except for their student group resources share a bucket; any meetings which cannot be analysed in a similar manner go into a leftovers bucket.

Second, a series of decisions is taken to merge certain sets of buckets. These decisions are made by a sequential heuristic which produces one merged bucket per iteration. Buckets that cannot be timetabled effectively because of a lack of resources are merged with other buckets. For example, the bucket holding the Music classes from Figure 5 is not viable alone and must be merged. Other relevant factors include preassigned times, the presence of student group resources from several forms, and a preference for tiles whose width (number of times) is a multiple of the usual column width, for regularity.

Finally, the meetings within each bucket are timetabled with respect to each other, producing tiles. This is a general time assignment problem, on a small scale, and the time assignment algorithm described in the next subsection is used to solve it. This step is interleaved with the previous one: if the bucket's timetable turns out to be more defective than its meetings individually, the bucket merging heuristic tries alternative bucket mergings.

4.3 Time assignment

After tiles are built, the next stage is to timetable them into the times of the cycle, producing a complete time assignment for all meetings.

The time assignment software module is called from three places within the KTS solver: to timetable submeetings into their meetings, meetings into their tiles, and tiles into the cycle. These problems are all essentially the same, differing only in scale. This description will speak of timetabling meetings into the cycle, rather than introducing unilluminating general terminology.

The meetings to be timetabled are first grouped into *layers*: sets of meetings required to be disjoint in time, typically because they contain the same preassigned student resources. The layers are sorted so that the most difficult ones (those requiring the most resources) come first, and timetabled one by one with no backtracking. A meeting may lie in more than one layer, in which case it is timetabled along with its first layer. In the time assignment stage of the solver there is one layer per student form, plus one layer for each staff meeting.

Within each layer, each meeting is timetabled in turn, widest first, if possible into a single column. A few assignments are tried for each meeting, but without backtracking; instead, forward checks, involving two kinds of bipartite matchings that monitor the availability of resources, keep the solver on track. These checks are described in detail in a companion paper [7]. A timetable created by this algorithm, plus time adjustment, appears in Figure 6.

4.4 Time adjustment

After a complete time assignment is obtained, *time adjustment* attempts to improve it by hill climbing: swapping time blocks around while this produces an improvement. Hill climbing is very effective here, since it corrects simple problems resulting from the lack of backtracking during time assignment, in time proportional to the number of improvements it makes.

Although no resources have yet been assigned to meetings, there are nevertheless two useful evaluations that can be made at this point: checking the sets of times assigned to meetings for their conformance to time conditions, and checking that resources are sufficient at each time to cover the resource demands made by meetings assigned that time (using a bipartite matching at each time between resource demands and resources). A neighbour is accepted if it reduces problems with resources, or improves time conditions without increasing problems with resources.

There are several promising neighbourhoods that could be tried. The current implementation explores two, repeating until neither gives any improvement.

The first neighbourhood takes each pair of time blocks of equal size assigned to columns, such that none of the times involved is preassigned to any meeting or column, and tries swapping these time blocks globally through every meeting. This might reduce resource problems as well as time condition problems, because resources' unavailable times stay fixed, and a swap might move resource demands away from the unavailable times of the resources they need.

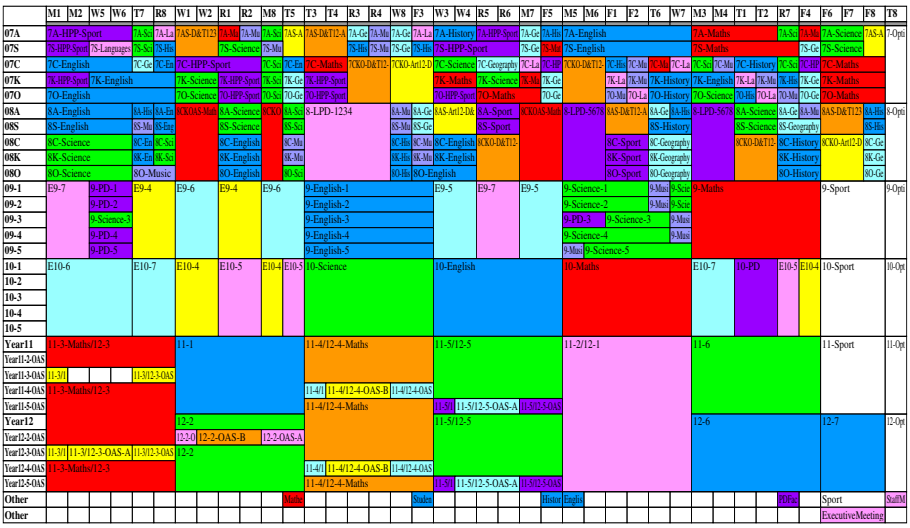

Fig. 6. A planning timetable for the *bghs98* instance. Each row except the last two represents the timetable of one student group resource. The columns represent times, permuted to bring the times of the columns (in the column layout sense: six of width 6 and one of width 4) together, making them and the tiles within them clearly visible. An example of a time adjustment, swapping Science with Personal Development, appears in the row of student group 09-3. This diagram was generated in PostScript by KTS.

The second neighbourhood takes pairs of meetings that contain the same preassigned resources (typically student group resources) and swaps blocks of their times of equal width. Since this can disrupt the regularity of a timetable, these swaps are only accepted if they reduce problems with resources, and indeed are only tried at times where there are such problems.

4.5 Resource assignment

Resource assignment is the assignment of particular resources to the resource slots of meetings. The solver does this after times are all assigned.

Each resource group may be assigned independently of the others, apart from a slight connection caused by 'follows' requirements. For each resource group in turn, in an order influenced by the presence of 'follows' requirements, preassignments are first converted to assignments, then assignments arising from 'follows' requirements are made, then all remaining unassigned slots are assigned. Some preassignments may fail to convert owing to resource unavailabilities and workload limits; their slots remain unassigned and become defects in the solution.

The resource assignment problem comes in two versions, depending on how acceptable split assignments are. Typically, split assignments are undesirable when assigning teachers, but acceptable when assigning rooms, provided classes do not have to change rooms part-way through a time block.

	Avail	M1	M2	W5	W6	T7	R8	W1	W2	R1	R2	M8	T5	T3	T4	R3	R4	W8	F3	W3	W4	R5	R6	M7	F5
Gibbons	0	8C-Science				*7A-Scie*	8C-Scie	*7K-Science*		8A-Science		*7A-Scie*	8A-Scie						Studen	12-5-Chemistry					
Kassab	0							7O-Science		8S-Science		7O-Scie	8S-Scie	10-Science2						11-5-Biology					
Kidd	0	12-3-Physics										7C-Scie		10-Science1						7C-Science					
Prasad	0	8K-Science					8K-Scie	12-2-GeneralScience-1						10-Science3											
Saule	0	*8O-Science*						12-2-Biology						10-Science5						12-5-Biology					
Smith	1			9-Science-3		7S-Scie				7S-Science			*8O-Scie*	10-Science4						11-5-Physics					
Unassigned												7K-Scie										7K-Science			

	M5	M6	F1	F2	T6	W7	M3	M4	T1	T2	R7	F4	F6	F7	F8	T8
Gibbons	9-Science-1					9-Scienc			8A-Science		*7A-Scie*		Sport			StaffMe
Kassab	11-2-GeneralScience						7O-Science		8S-Science							
Kidd		9-Science-5					7C-Scie				7C-Scie		ExecutiveMeeting			
Prasad	9-Science-4						11-6-Chemistry-1						Sport			
Saule							11-6-Chemistry-2									
Smith			9-Science-3				11-6-Biology						7S-Science			
Unassigned																

Fig. 7. Planning timetable showing the teacher assignment for the Science faculty of the *bghs98* instance. (The resource assignment algorithm assigns all faculties simultaneously, but it is convenient to analyse its results faculty by faculty.) The second column gives the remaining unused workload of each teacher. Split and partial assignments are shown in italic font. There are three unassigned tixels. This diagram was generated in PostScript by KTS.

Room assignment is not difficult. The solver assigns each time block of each meeting, largest blocks first, choosing a qualified resource whose use does not increase the number of resource problems at any of the block's times (the usual bipartite matching checks this condition), and preferring a resource which has already been assigned to another block of the meeting. If a block of two or more times is encountered for which this is not possible, it is split into blocks of width 1; if a block of width 1 cannot be assigned, it is passed over and becomes a defect in the solution.

The teacher assignment algorithm tries much harder to avoid split assignments (Figure 7). It is based on the alternating path method familiar from bipartite matching and similar problems, used as a heuristic, since the optimality guarantees that usually accompany it are absent.

Choose a currently unassigned teacher slot of maximum width. If there is a qualified teacher able to fill this slot (i.e. without causing clashes or exceeding workload limits), assign that teacher and move to the next widest slot. Otherwise, see if there is a teacher who could fill the slot if only some one of the assignments currently given to that teacher were deassigned and given to some other teacher able to fill it. If so, make the indicated chain of two assignments and one deassignment, and move on. If not, look for a longer chain, and so on. At each moment when there are no workload overloads or clashes, compare the whole set of assignments with the best so far, and replace it if it is better.

Table 1. The six instances tested, showing the number of times, resources, and meetings in each.

Instance	*Times*	*Student Groups*	*Teachers*	*Rooms*	*Meetings*
bghs93	40	23	53	46	155
bghs95	40	27	52	48	147
bghs98	40	30	56	45	152
tes98	30	11	33	20	95
tes99	30	13	37	26	86
sahs96	60	20	43	36	131

Two methods of controlling the size of the search are used. One is the traditional one of marking each possible assignment and deassignment *visited* when it is first considered, and refusing to reconsider it during the course of the search (it becomes available again when we move to the next slot). The other method is to allow revisiting but to strictly limit the depth of the search, to the empirically determined value of 5 (three assignments and two deassignments). The searches are repeated until there is no improvement.

At each slot, in addition to searching for ordinary assignments, the solver finds a qualified resource which is available for as many times as possible, and generates all split assignments which have that resource and those times as the first branch, and one other qualified resource with the remaining times as the second branch. The alternating path search continues down the second branch. A single partial assignment is also generated, holding the first branch as before but omitting the second.

5 Results

This section analyses the performance of the solver on six instances taken from three high schools in Sydney, Australia. Statistical descriptions of these instances appear in Table 1, run times are given in Table 2, and the quality of the solutions is summarized in Tables 3, 4, and 5. The solver always assigns the correct number of times to each meeting, never introduces student group clashes, and prefers to leave teacher and room slots unassigned rather than introducing teacher and room clashes and workload overloads. So the possible defects are time assignment problems (wrong block structure, meeting spread over too few days, etc.) and unsatisfactory room and teacher assignments (split, partial, and missing).

The *sahs96* instance has a two-week cycle, and all its teacher slots are preassigned. These two factors make time assignment very slow. It is encouraging that only 3.1% of these preassigned teacher tixels could not be assigned (Table 5), given that the solver is not optimized to handle instances that are highly constrained in this way. However, the solver's desperate attempt to satisfy all these preassignments leads to a quite irregular timetable.

The other instances are more typical of the solver's intended domain of application. Run times are under ten seconds. Block structure defects are somewhat

Table 2. Run times in seconds for the major stages and in total. The tests used a 3.2GHz Pentium machine running Linux. Run times are as reported by the Linux *time* command, which is accurate to one second. Column layout time was always 0.0 seconds so has been omitted. Time assignment includes time adjustment by hill climbing, never more than one second. The times given for resource assignment essentially measure teacher assignment only, since room assignment is very fast. Total times were checked against wristwatch time.

Instance	*Tile construction*	*Time assignment*	*Resource assignment*	*Total*
bghs93	0.0	3.0	3.0	6.0
bghs95	0.0	1.0	7.0	8.0
bghs98	0.0	1.0	6.0	7.0
tes98	1.0	1.0	0.0	2.0
tes99	0.0	1.0	0.0	1.0
sahs96	1.0	31.0	0.0	32.0

Table 3. Evaluation of time assignments, showing the absolute number of meetings with defective block structure, uneven spread through the cycle, and more than one undesirable time, plus this number as a percentage of the total number of meetings.

Instance	*Block structure*	*Spread*	*Undesirable times*
bghs93	48 (31.0%)	51 (31.2%)	-
bghs95	20 (13.6%)	44 (29.9%)	7 (4.8%)
bghs98	5 (3.3%)	31 (21.1%)	0 (0.0%)
tes98	36 (37.9%)	22 (23.2%)	2 (2.1%)
tes99	37 (43.0%)	27 (31.4%)	-
sahs96	2 (1.5%)	74 (56.5%)	18 (13.7%)

high (Table 3). This problem awaits analysis but should be correctable. Time conditions defects are probably acceptable now, given their relative unimportance, although there is room for improvement.

Resource assignment can be evaluated either in terms of the number of defective assignments (split, partial, or missing), or the number of unassigned individual tixels (a *tixel* is one resource at one time, either supplied or demanded). Some tixels are inevitably unassignable given a particular time assignment – for example, if the time assignment requires five Science laboratories to be available at some time, but the school has only four. These are shown in the fourth column of Tables 4 and 5, while the number of unassigned tixels after resource assignment is shown in the fifth column.

Room assignment (Table 4) is virtually perfect. The room assignment algorithm always assigns every room tixel that time assignment permits, because it breaks time blocks up into individual times if necessary, and, using a bipartite matching between room demands and rooms at each time, it never allows the number of unassignable rooms at any time to increase. This is why the fourth and fifth columns of Table 4 are equal. The fact that only two split assignments were ever introduced shows how easy this problem is in practice.

Table 4. Evaluation of room assignments, showing the absolute number of split assignments, partial and missing assignments, unassignable tixels after time assignment (1), and unassigned tixels after resource assignment (2), plus this number as a percentage of the number of room assignments or tixels demanded. In this table, a split assignment is one in which a class has to change rooms part-way through a time block.

Instance	*Split*	*Partial/missing*	*Tixels (1)*	*Tixels (2)*
bghs93	0 (0.0%)	7 (3.1%)	15 (1.2%)	15 (1.2%)
bghs95	0 (0.0%)	6 (2.9%)	9 (0.7%)	9 (0.7%)
bghs98	0 (0.0%)	5 (2.1%)	7 (0.5%)	7 (0.5%)
tes98	2 (2.2%)	5 (5.5%)	7 (1.5%)	7 (1.5%)
tes99	0 (0.0%)	5 (3.7%)	7 (1.3%)	7 (1.3%)
sahs96	0 (0.0%)	27 (11.2%)	15 (1.0%)	15 (1.0%)

Table 5. Evaluation of teacher assignments, showing the absolute number of split assignments, partial and missing assignments, unassignable tixels after time assignment (1), and unassigned tixels after resource assignment (2), plus this number as a percentage of the number of teacher assignments or tixels demanded, as appropriate. In this table, a split assignment is one in which a class is taught by two teachers.

Instance	*Split*	*Partial/missing*	*Tixels (1)*	*Tixels (2)*
bghs93	3 (0.7%)	7 (1.5%)	5 (0.3%)	9 (0.6%)
bghs95	17 (3.7%)	15 (3.3%)	7 (0.5%)	27 (2.0%)
bghs98	24 (5.4%)	10 (2.3%)	8 (0.5%)	17 (1.2%)
tes98	7 (3.8%)	13 (7.1%)	14 (3.0%)	14 (3.0%)
tes99	2 (1.1%)	9 (5.1%)	9 (1.7%)	9 (1.7%)
sahs96	0 (0.0%)	27 (11.2%)	47 (3.1%)	47 (3.1%)

Unassigned room tixels typically request specialized laboratories whose demand is very tight. This problem is quite common in high schools and is not of major concern, since, given its low relative frequency, it is not difficult to ensure that no class meets in an inappropriate room for more than one of its times, and the teacher would organize the classroom material accordingly. An option to assign inappropriate rooms where necessary, spreading them fairly among the classes affected, could easily be added.

Split teacher assignments and unassigned teacher tixels (Table 5) are the main areas of concern. How acceptable these results are it is hard to say. Hand-generated timetables also have these problems. Split assignments are quite routine. Unassigned tixels are handled in various ways: by excusing a teacher from a faculty meeting, having an available but unqualified teacher supervise a class, and so on. Unlike other defects, every unassigned teacher tixel is a real problem requiring the attention of the timetable planner.

One unassignable tixel in a teacher slot spoils the assignment of the entire slot. This suggests that finding time assignments with fewer unassignable teacher tixels would be more helpful than improving the teacher assignment algorithm.

6 Conclusion

This paper has presented KTS, a freely accessible web-based system for high school timetabling which produces good timetables in a few seconds.

The fast response time makes KTS well suited to exploring alternative scenarios and incorporating late changes to requirements. However, KTS does not yet address the problem of making minimal changes to a published solution in response to changes in requirements.

The data model is mature, except perhaps in its treatment of time, and the overall structure of the solver is quite successful. It seems likely that future work will focus on improving the existing solver components, rather than radically redesigning the solver. The time assignment stage is the obvious next target for improvement. In fact, since this paper was written, the author has designed and implemented a more flexible approach to time assignment and adjustment which should allow the algorithms described here to be varied and generalized in several interesting ways [7].

In parallel with these efforts, the KTS system will be promoted to Australian high schools. At the time of writing, 60 accounts have been created, but only a few are active. More users will bring a larger and more diverse set of test instances, which should lead to further progress.

References

1. Edmund Burke and Wilhelm Erben (eds.): Practice and Theory of Automated Timetabling III (PATAT2000, Konstanz, Germany, August 2000, Selected Papers). Springer Lecture Notes in Computer Science 2079, 2001
2. Edmund Burke and Patrick de Causmaecker (eds.): Practice and Theory of Automated Timetabling IV (PATAT2002, Gent, Belgium, August 2002, Selected Papers). Springer Lecture Notes in Computer Science 2740, 2003
3. M. W. Carter and Gilbert Laporte: Recent developments in practical course timetabling. Practice and Theory of Automated Timetabling II (Second International Conference, PATAT'97, University of Toronto, August 1997, Selected Papers), Springer Lecture Notes in Computer Science 1408, pages 3-19, 1008
4. Tim B. Cooper and Jeffrey H. Kingston: The solution of real instances of the timetabling problem. The Computer Journal **36**, pages 645–653, 1993
5. Jeffrey H. Kingston: A tiling algorithm for high school timetabling. In Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, Pittsburgh, PA, pages 233-249, August 2004
6. Jeffrey H. Kingston: The KTS high school timetabling web site (Version 1.3), October 2005. http://www.it.usyd.edu.au/~jeff
7. Jeffrey H. Kingston: Hierarchical timetable construction. To appear in Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling, Brno, Czech Republic, August 2006
8. Peter Ross, Emma Hart, and Dave Corne: Some observations about GA-based exam timetabling. Practice and Theory of Automated Timetabling II (Second International Conference, PATAT'97, University of Toronto, August 1997, Selected Papers), pages 115-129, 1998