The Development of SYLLABUS—An Interactive, Constraint-Based Scheduler for Schools and Colleges

Rick Evertsz

In most schools and colleges, producing a timetable is a task that is dreaded by those charged with annually creating it. In schools in particular, it can be difficult to produce any solution at all because teaching resources are usually stretched to the limit. This situation is made more difficult by the fact that the person making up the timetable must not only produce a valid solution but one that is of good quality.

A large number of judgmental factors are involved in producing a good-quality timetable. Some of these are educational (for example, it is undesirable for a group of pupils to have a Spanish lesson immediately following a German one), others pertain to the work conditions of the staff (for example, the teaching load should be distributed evenly through the day so that no teacher has a long, contiguous block of lessons), and others are political (for example, a senior teacher should not end up with fewer free periods than a more junior one). These qualitative factors make an already difficult problem time consuming indeed.

For a medium-sized school (about 1,000 pupils), even a veteran at

the task takes several weeks to produce a satisfactory timetable. Despite this great investment in time and concentration, the timetable will inevitably contain errors, such as requiring a teacher to be in two places at once. Tight resources, together with the need to juggle many qualitative factors, means that the person creating the timetable only has one shot at producing a solution: The timetable is effectively set in concrete and is only amenable to minor modification. Because of the time-consuming nature of manually producing a timetable, there is no opportunity to try alternative scenarios that might enable cost savings or late extensions to the curriculum.

With the advent of greater access to digital computers, attempts began in the seventies to solve this problem by computer. Given the difficulty of manually producing a timetable, one would expect a computer-based timetable program to be a godsend. Early computer-based timetable programs ran on mainframe computers: The school would send its requirement to the administrative center, which would enter the data and run the timetable program overnight. The resulting timetable was usually unsatisfactory because it violated constraints that were important to the school. As a result, school timetable programs got a bad name for themselves. As far as schools were concerned, computers were incapable of producing a solution of the quality achieved by humans performing the same task.

Previous timetable systems have been based on operations research techniques. Although able to generate solutions, the main failing of this approach is the lack of control available to the user. The only way to alter the solution is to change some of the numeric parameters that control the priority of the various factors. This approach is fairly hit or miss because it is never clear what effect altering some numeric value will have on the overall solution. Based as they are on mathematical programming methods, such timetable programs are difficult to control because their processing is completely opaque to the user.

In contrast, AI approaches to problem solving are characterized by their reliance on symbolic, rather than numeric, methods. This technique makes it much easier for a program to communicate its problem-solving rationale to the user. The application of AI to scheduling has a relatively short history, only starting to gather momentum in the mid-eighties (for example, Fox and Smith [1984] and Keng, Yun, and Rossi [1988]). Constraint-satisfaction problem techniques (in particular) seem to offer a powerful way of solving scheduling problems (compare Dincbas, Simonis, and Van Hentenryck [1988] and Keng and Yun [1989]). SYLLABUS uses a constraint-satisfaction problem approach to producing timetables, rarely fails to produce a timetable, and can usually detect such cases before it begins scheduling. However, it is com-

mon for operations research–based timetable programs to only manage to produce a timetable for some 90 percent of the curriculum, leaving the user to manually insert the remaining lessons. Unfortunately, these lessons are typically the hardest to fit; so, the machine is only solving the simple part of the problem.

The School Timetable Problem

It comes as a surprise to many who are familiar with scheduling problems to learn that in practice, the setting up of school timetables is one of the hardest problems to solve. The main reason is that school resources are about as tightly stretched as they can be without also facing an insoluble timetable problem (for example, one 1600-pupil school, for which SYLLABUS successfully produced a timetable, had 98 percent use of its mathematics teachers). There is also a large degree of interaction between timetable resources: Teachers can teach a range of subjects (in some order of preference), and rooms can have more than one possible use. Therefore, in assigning a mathematics-French teacher to a mathematics lesson, the person producing the timetable must be wary of the fact that s/he might no longer be able to allocate enough teachers to the remaining French lessons. Assessing the ramifications of such assignments is not easy, and as a result, injudicious choices early on can lead to much backtracking.

A particular thorn in the side of those producing a timetable is the option group, which is designed to provide the pupil with greater curricular choices and is used extensively in the upper school. An *option group* is a composite lesson consisting of a set of subjects (options) offered to a set of classes (groups of pupils). Each pupil in each class makes a choice from those subjects offered in the option group and joins the pupils from the other classes who chose the same option. A typical option group is as follows:

Classes:	4A, 4B, 4C, 4D, 4E, 4F, 4G, 4H
Subjects:	Biology, Chemistry, English, Economics, French, Geography, History, Latin, Mathematics, and Physics

Some children from each of the 8 classes might take biology, whereas other children from these classes might study physics. This problem can be hard to solve in a timetable. The person developing the timetable must find a time slot (possibly several periods in length) where all 8 classes are free, and 10 teachers of the appropriate kind are available, as are 10 suitable rooms (for example, the science lessons typically require an appropriate laboratory). As the timetable process unfolds, these lessons become more and more difficult to place. The

assignment of option groups is further complicated by the fact that there is a lot of interaction between them. For example, the previous option group would have to be assigned to a time slot that, through the available resources, enables the classes 4A to 4H to have the same teacher throughout the week.

In summary, the high demand for teaching resources, together with the complex interactions caused by option groups, makes school timetable production a difficult scheduling problem. This already difficult problem is further complicated by the need to satisfy the many qualitative factors that a school timetable should incorporate.

Development History

In 1987, my research group was approached by Kent County Council, which wanted to investigate the feasibility of applying AI to its timetable problems. Council members believed that better use could be made of the county's teaching staff by, for example, sharing some teachers between adjacent schools. It was estimated that a ½percent saving could be made in the annual education budget of $680,000,000.

Such a solution would only be acceptable if it did not compromise the educational quality of the current timetables. The first prototype was developed in 15 days at a 1,600-pupil school, using Intellicorp's KEE on a Symbolics Lisp machine. Although adequate, its solutions were not favored by the school doing the evaluation because staff members did not have sufficient control over the process. The first prototype was completely discarded, and in the interests of efficiency, a new version was implemented in Lisp. The user was provided with a graphic interface through which s/he could select a lesson that needed to be scheduled in a timetable, place it on a free period, and select the resources needed by the lesson. The machine would advise him(her) (graphically) in the choice of lesson, time slot, and resources. The user could also ask the machine to explain its advice, so s/he could make a more informed choice when choosing to override the machine's suggestions.

This solution proved satisfactory, but the machine was not solving the problem: It was merely acting as a decision support tool for the user, who was the final arbiter in the choice of lesson allocation. It was clear that one could add an automatic timetable facility to the program by making it follow the advice that it was offering the user. The user could freely switch between automatic and advisory mode, reallocating choices made by the machine that s/he did not like. Together with its use of AI techniques, this design feature was what set SYLLABUS apart from earlier systems.

The prototype's performance was acceptable, taking roughly 3 hours

to produce a timetable for a medium-sized school. However, the hardware was far too expensive (about $85,000). From published Lisp benchmarks, it was estimated that porting the Lisp implementation to 386-based personal computers and 68030-based Macintoshes would degrade the performance of SYLLABUS by a factor of 5 to 7. This performance would have been unacceptable, so it was necessary to redesign the scheduling algorithm.

To maximize portability between the two platforms, the scheduling kernel was completely separated from the user interface by adopting an object-oriented design. The interface is in control of the overall program and treats the kernel as a black box to be interrogated. It communicates with it by sending messages, such as "Timetable another lesson;" "Add a new teacher, called Mr. Peters, who teaches French as a primary subject and German and games as secondary subjects;" or "Why can't I place this lesson here?" The kernel never calls interface functions, it merely obeys commands and occasionally complains about what it has been asked to do (for example, if the teacher Mr. Peters were already defined, then it would signal an error to the interface, which would then decide what to do about it).

The use of object-oriented techniques provided a clean division between interface and kernel, speeded development, improved portability, and facilitated maintenance. The kernel was defined as an object that handles a certain message-passing protocol. This narrow bandwidth communication channel enabled the implementation of a dummy version of the kernel in three days, which meant that the user interface on each platform could be developed independently of the kernel.

With its newly designed scheduling kernel, SYLLABUS is 100 times faster than its predecessor on a Lisp machine. On a 68030-based Macintosh, it is 15 times faster than the earlier Lisp machine version, producing a timetable for a medium-sized school in about 12 minutes. Considering the complexity of the task, this performance is acceptable. The size of the Macintosh application is 3 megabytes (MB); thus, it runs comfortably on a 4-MB machine.

The Macintosh and IBM PC–compatible implementations were developed by a team of 3, one working on the kernel, and the other 2 each working on one of the interfaces. The kernel was developed in 2 person-months, and each interface took about 3 person-months. Another 4 person-months were devoted to optimizing the kernel in an effort to obtain acceptable performance on the IBM PC; the latter goal was not achieved. Despite the fact that they were running the same scheduling kernel, the IBM PC version ran about 5 times more slowly than the Macintosh version and required a minimum of 6 MB. From a hardware perspective, one would not expect code generated by a Lisp compiler

on a 386 to be any slower than that for a 68030. The problem was that the only Microsoft Windows-based Lisp compiler available for the IBM PC–generated 286 code; this code is inherently more verbose than 386 code based on a flat memory model.

The researchers in my group believe that the decision to use Lisp, rather than a conventional language such as C, led to the relatively short development time. When delivering on low-specification hardware, it is common practice to recode an AI application in a more conventional language. However, experience has shown that through good design, it is possible to produce Lisp applications that are efficient in both space and time on standard hardware.

Once developed, SYLLABUS was beta tested in 10 separate schools. This period spanned 4 months, during which bugs were removed, and new functions were added. It was relatively easy to incorporate the requests of the beta test sites for two reasons: (1) the features of Lisp that foreshorten development time (object orientation; automatic memory management; wide choice of data structures) also make extensions easier to incorporate and (2) the constraint-based architecture means that new constraints can be added without having to fundamentally alter the kernel's architecture.

The beta test cycle was complete in June 1990, at which point the Macintosh product was fully released in the United Kingdom. In the 6 months since June, it sold about 50 copies; this number is high, given that the United Kingdom Macintosh market is only 10 percent of the size of the personal computer market.

An Outline of the SYLLABUS Functions

The user enters details about the school through a set of data-input windows. The school parameters window is used to define the format of the school week (number of days, periods in each day, break and lunch times, and periods not available for teaching).

The user can then enter details of the subjects to be taught and any pedagogic constraints (for example, avoid setting the subject physics next to chemistry). Teaching profiles are entered through the teachers window (figure 1). In this example, the teacher Thomas Boots is defined as primarily a chemistry teacher; however, he can also teach mathematics and physics. He does not teach on the last 3 periods of Monday; this condition is shown in the grid that represents the week as 5 days (rows) of 6 periods (columns), with breaks after the second and fourth periods. Thomas Boots also requires 10 free periods each week. Further data-input windows are used to define the characteristics of the

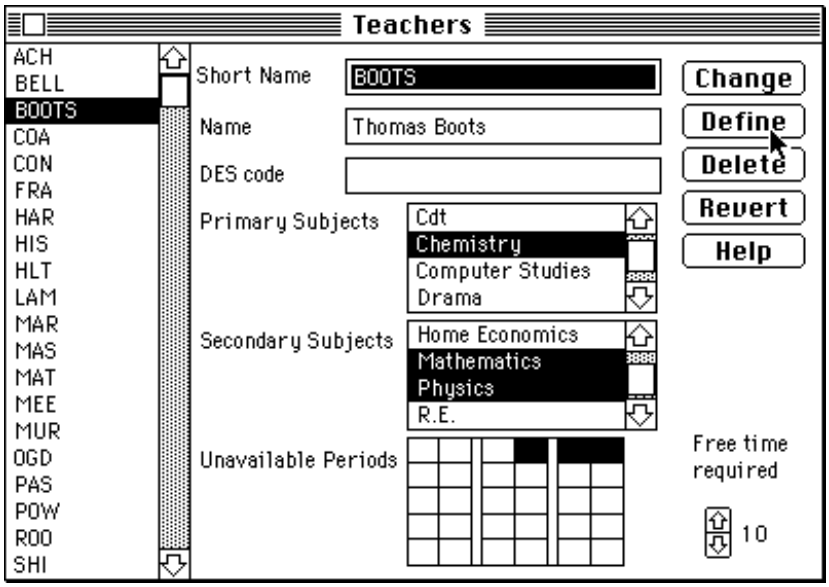

Figure 1. The Definition of a Chemistry Teacher.

classrooms and the structure of the curriculum. The user can then invoke an automatic set of feasibility checks on the data. This module analyzes the data and prints a report of why, if at all, it is impossible to generate a timetable. This facility is invaluable because it is easy for an inexperienced person to enter a curriculum requirement that is impossible to satisfy. Common errors include (1) assigning insufficient teachers for the quantity of subjects being taught, (2) giving a group of pupils more lessons than can fit in a week, and (3) grouping the options within the option groups so that the dependencies between all option groups preclude the production of a valid timetable. Error 3 entails examining the connectivity between option groups, as defined by their possible resource profiles, and is equivalent to the graph-colorability problem. It highlights groups of lessons that must be placed back to back; a timetable cannot be produced if the total length of a lesson sequence is greater than the number of periods in the week.

Once any problems discovered by the feasibility checks are rectified, the user can start producing the timetable. The task is performed through the timetable pane (figure 2). SYLLABUS sorts the classes (shown by the leftmost list) in terms of how constrained they are. The user should select the uppermost class in the list because it has the most constrained lesson in the whole school. The lessons for the selected class are then displayed in the adjacent list, again sorted by how constrained

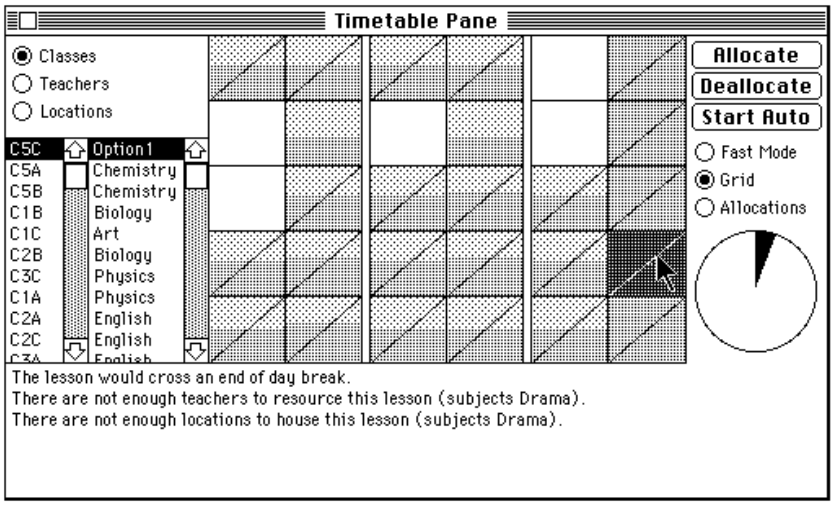

Figure 2. Creating Timetable Option1 Interactively.

they are. The user then selects the uppermost lesson in this list and is presented with a shaded timetable grid. The lesson cannot be assigned to those time slots that are shaded with a diagonal line; those that are shaded but have no diagonal line are inadvisable but valid (for example, cell (2,2) is inadvisable because option 1 is a double lesson and, thus, would straddle the midmorning break). The unshaded cells are those that SYLLABUS recommends for the selected lesson.

The figure shows the user clicking on the last period on Thursday to find out why the lesson cannot be placed there. The reasons appear in the pane below the shaded grid. Note that each barred period has a different level of grey; this level signifies how impossible the period is. Those with a higher level of grey have more reasons why they cannot be used than those with a lower level. In the limit, where there are no periods available for allocation, the different levels of grey enable the user to identify those periods with the fewest problems. For example, a period with only a room missing is fairly easy to satisfy; the user only needs to shift the lesson currently in the room to a different period. However, a period that is short of several teachers and rooms and for which several of the classes are unavailable would be hard to free up.

In interactive mode, the user is not forced to follow the system's advice. In fact, interactive mode is generally used when the user wants to override the SYLLABUS choices, for example, when allocating lessons that are fixed points in the timetable (for example, the lessons are the same every year). The user can pass control to SYLLABUS by clicking on the

start auto button. SYLLABUS then starts producing the timetable by selecting the most constrained lesson, shading the grid, choosing the best cell for the lesson, and then allocating it together with an appropriate set of resources. The user can monitor this process and interrupt at any time to override a system choice (for example, edit the resources chosen by SYLLABUS, or move an allocated lesson to another cell).

An Overview of the Implementation

In automatic mode, the SYLLABUS control structure consists of three operations: First is *choose-event*. This operation selects the most critical lesson. Informally, this lesson has the least flexibility at a given point in time; in other words, it has the fewest time slots and resources available to it.

Second is *choose-time*. Given a most critical lesson, this operation chooses the best time slot for this lesson and combines the following factors: (1) a good spread of similar subjects (for example, English lessons are evenly spread throughout the week, and French lessons are kept apart from German ones); (2) multiple-period lessons evenly spread throughout the week; (3) a smooth distribution of teaching load; (4) a given group of pupils always taught a given subject by the same teacher; (5) multiple-period lessons not crossing an intraday break unless absolutely necessary; and (6) a time that maximizes the use of primary, rather than secondary, resources.

Third is *choose-resources*. Given a lesson and a selected time slot, this operation selects the best set of resources for the lesson on the basis of factors 3, 4, and 5 for the choose-time operation.

The SYLLABUS approach is based on the view that producing a timetable is a constraint-satisfaction problem. A constraint-satisfaction problem consists of a set of variables, $V = \{v_1,\ldots, v_n\}$; a set of related domains $\{D_1,\ldots, D_n\}$; and a set of constraint relations between these variables, $\{C_1,\ldots, C_m\}$. Each variable, v_i, takes its values from the finite domain, D_i. Each constraint, C_j, is expressed with respect to some subset of the variables in V and defines the consistent tuples of domain values that this subset can take (that is, $C_j \subseteq D_1 \times \ldots \times D_n$).

Each lesson is associated with a set of constraint variables: time, teachers, and locations. In the simple case, a lesson has only one teacher and one location; however, with option groups, the lesson typically involves a number of teachers teaching in different locations. Given a set of variables in a lesson, the actual legal subset of values that these variables can take is defined by the set of constraints.

Although time can be viewed as a resource, SYLLABUS treats this variable in a special manner for two reasons: (1) Users are familiar with

the notion of allocating a lesson with a set of resources to a given time period; therefore, it makes sense to schedule from this perspective because the advice and explanations will more easily be understood. (2) Time is a particularly constraining resource: A lesson (option group) can have a set of teachers and a set of locations, but it can only be assigned one time period.

Each time period for a lesson is associated with a number denoting the *resource overflow-underflow* (RVF). These values are incrementally updated as resources are allocated to, and deallocated from, other lessons. For example, if a lesson requires eight teachers, but only three of the appropriate kind are available during the first period, then RVF would be –5. SYLLABUS sums the positive values in this time vector to obtain a measure of the *survivability* of the lesson. The lesson with the lowest survivability is the most critical and is chosen by choose-event. When a resource is allocated to a lesson, the survivability of all lessons that could have used this resource is updated. Here, the key to efficiency lies in making incremental updates rather than recomputing the survivability from scratch on each cycle. When a set of resources, R_i, is assigned to a lesson, l_i, at time, t_i, then for each lesson that might have used one or more of R_i, SYLLABUS decreases RVF for t_i. If this value is already negative, then the survivability remains unchanged. However, if this value is positive, then the incremental change is applied to the survivability measure. The inverse procedure is applied when freeing resources.

SYLLABUS makes extensive use of bit vectors to rapidly compute the effects of assignments. For example, each resource is associated with a bit vector that represents those lessons that it is allowed to use in the timetable (regardless of time period). When a given set of resources is assigned to, or removed from, a lesson, the set of lessons whose survivability is affected can rapidly be computed from the inclusive Or of these bit vectors (that is, the union of these sets). The set of resources available at a given time period is represented as a bit vector (the *resource-availability-bv*). To compute the set of resources available to a particular lesson at a given time period, SYLLABUS takes the bit vector that represents those resources that can supply the subject taught in the lesson and computes its logical And and the resource-availability-bv.

As the timetable process progresses, the variable domains are gradually reduced by forward checking (Haralick and Elliot 1980; Van Hentenryck and Dincbas 1986). In this scheme, when the domain of a variable is reduced by some action, the invalidated domain elements of other variables are immediately removed, saving a number of the consistency checks that must be performed. For example, if a group of pupils has five science lessons during the week, and the first of these

lessons is assigned a teacher, r_i, the other four pupils have their teacher domains reduced to $\{r_i\}$. This number, in turn, usually restricts the time periods available to the other four lessons (because r_i might be busy at these times).

SYLLABUS is both space efficient and fast. Much of this speed and compactness results from the use of bit vectors to represent the domains of variables and perform set union, intersection, and complement operations.

Discussion

SYLLABUS provides its users with a number of benefits. The most obvious of these benefits is the time saved by the senior staff member who normally has to spend weeks producing the timetable. A customer survey shows that new users take between one and five days to produce a final timetable. This total time includes the time taken to enter all the raw data (note that entering the raw data goes quickly after the first year because much of them remain constant from year to year). The upper bound of five days is surprisingly quick because it relates to users who are computer naive and have no experience producing a timetable. Those who are experienced producing a timetable manually complete the task in under two days and are delighted with the time saved and the quality of the results.

Because of the time saved, it is now possible to experiment with different timetable scenarios; thus, SYLLABUS becomes a planning tool. Schools can now obtain answers to questions such as "Can we add environmental studies to the curriculum without hiring another teacher?" This ability not only increases the educational choice of the pupils but gives the school a competitive advantage over others. This bonus is important for private schools that are competing with others for a limited supply of paying pupils. Because SYLLABUS can dynamically reschedule those parts of the timetable that are affected by an unexpected external event (for example, absent teacher or storm or fire damage to one of the school buildings), it can be used as a crisis-management tool.

SYLLABUS gives schools the opportunity for substantial cost savings. Sharing teachers between adjacent schools has already been mentioned; in a private school, cutting the staffing needs by one teacher can save about $42,000 each year (which favorably compares with the cost of SYLLABUS: $1,700). One private school has used SYLLABUS to remove the need for Saturday morning teaching from their timetable, representing the first time that it has been able to generate a full timetable that fits in less than 5½ days. Furthermore, this five-day

timetable was judged to be of better quality than any of the 5½–day schedules manually produced in previous years. Some schools can use SYLLABUS to generate income by releasing resources such as sports halls and language laboratories. This task is achieved by making the rooms in question unavailable for, say, a complete afternoon and then running SYLLABUS to see whether it is still possible to produce a satisfactory timetable. A similar approach can be used to save on winter fuel bills by switching off the heating in some blocks of the school building.

To date, these potential savings remain an unexploited benefit of SYLLABUS. Schools prefer to use SYLLABUS to improve the educational quality of the timetable (after all, education, rather than cost savings, should be the primary goal). It would be a shame if SYLLABUS were used to ruthlessly cut costs at the expense of educational quality. Schools in the United Kingdom are currently going through a transition period from a state in which cash savings are returned to the education authority to one in which the school is allocated a budget and can use the funds as it sees fit. Only when this transition is complete will it be worth using SYLLABUS to make cash savings that can be redeployed within the school.

Those who have been responsible in the past for producing a timetable by hand are particularly taken with the feasibility checks because they can now make a watertight case that a given curriculum cannot fit in a timetable. In the past, they have had to waste weeks struggling to produce a timetable before the head of the school finally relented and allowed changes to the curriculum.

In delivering SYLLABUS, it was important to the research staff that it look like any other application. In this market, the use of AI has a negligible effect on users' perception of the product. This situation meant that SYLLABUS had to be compact and responsive; users would not be satisfied with excuses such as "Well yes, it is slow, but it is doing some very clever things." SYLLABUS is sold as a school timetable program, not a Lisp-based AI application. The researchers in my group believe that it is unique in being a mass-market, off-the-shelf AI application implemented in Lisp. Having successfully delivered the final product, they believe that the time is ripe for using Lisp to deliver mass-market, microcomputer-based AI applications.

Acknowledgments

Apart from myself, many people have been involved in the development of SYLLABUS. Geoff Forster has been involved with the project from the start. Mark Dalgarno and Stuart Watt worked on the design and implementation of the first product. Ian Assersohn, Diana Billigheimer, and Jane Pusey have been involved with augmenting the SYL-

LABUS functions. Pauline Wilson has been advising schools with timetable problems for 20 years; her input to SYLLABUS has and continues to be invaluable.

References

Dincbas, M.; Simonis, H.; and Van Hentenryck, P. 1988. Solving the Car-Sequencing Problem in Constraint Logic Programming. In Proceedings of the Eighth European Conference on Artificial Intelligence, 290–295. London: Pitman.

Fox, M. S., and Smith, S. F. 1984. ISIS: A Knowledge-Based System for Factory Scheduling. *Expert Systems* 1(1): 25–49.

Haralick, R. M., and Elliot, G. L. 1980. Increasing Tree Search Efficiency for Constraint-Satisfaction Problems. *Artificial Intelligence* 14: 263–313.

Keng, N. P., and Yun, D. Y. Y. 1989. A Planning/Scheduling Methodology for the Constrained Resource Problem. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 998–1003. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Keng, N. P.; Yun, D. Y. Y.; and Rossi, M. 1988. Interaction-Sensitive Planning System for Job-Shop Scheduling. In *Expert Systems and Intelligent Manufacturing*, ed. M. D. Oliff, 57–69. Amsterdam: North-Holland.

Van Hentenryck, P., and Dincbas, M. 1986. Domains in Logic Programming. In Proceedings of the Fifth National Conference on Artificial Intelligence, 759–765. Menlo Park, Calif.: American Association for Artificial Intelligence.