From: AAAI-94 Proceedings. Copyright © 1994, AAAI (www.aaai.org). All rights reserved.

A Constraint-Based Approach to High-School Timetabling Problems: A Case Study

Masazumi Yoshikawa Kazuya Kaneko Yuriko Nomura Masanobu Watanabe

C&C Research Laboratories, NEC Corporation
4-1-1 Miyazaki, Miyamae-ku, Kawasaki 216 JAPAN
{yosikawa, neko, yuriko, watanabe}@swl.cl.nec.co.jp

Abstract

This paper describes a case study on a general-purpose *Constraint Relaxation Problem* solver, COASTOOL. Using COASTOOL, a problem can be solved merely by declaring "what is the problem," without programming "how to solve it." The problem is solved by a novel method that generates a high-quality initial assignment using arc-consistency, and refines it using hill-climbing. This approach has been evaluated successfully by experiments with practical high-school timetabling problems in Japan. Consequently, COASTOOL is shown to be efficient at applications in high-school timetabling problems.

Introduction

Every school must schedule its own timetable. Thousands of schools desire an automatic timetabling system earnestly. However, a timetabling problem is difficult, since it is large in scale and constrained tightly.

For example, Kuki-Hokuyou High-School has a large-scale timetabling problem with 30 classes[1], 60 teachers (including 9 part-time teachers), 34 time-slots during a week, and 806 lessons. The problem is assigning a time slot to each lesson considering various constraints and preferences. It is equivalent to selecting one from $10^{1234}(= 34^{806})$ combinations to solve the problem. Moreover, there are various constraints; a set of lessons must be taken simultaneously or continuously, a class should have no more than one lesson on a subject in a day, and so on. The problem is constrained tightly, since there is a constraint in that no class and no teacher is required to take more than one lesson at a time. Moreover, there is a condition that every class has no free time-slot during a week. Therefore the problem is just filling the weekly class schedules, considering teacher schedules and many other constraints. Consequently, timetabling is so difficult that it requires 100 person-days in the high-school.

A large amount of effort has been focused on solving timetabling problems, e.g., (Gotlieb 1963; Carter 1986; Feldman & Golumbic 1989; Corne, Fang, & Mellish 1993). However, there has been no work that succeeded in constructing a practically acceptable timetable for a real high-school.

This paper describes a general-purpose *Constraint Relaxation Problem (CRP)* solver, COASTOOL (COnstraint-based Assignment and Scheduling TOOL), and its successful experiments on practical high-school timetabling problems in Japan.

COASTOOL has a constraint-based architecture in which a declarative problem description is independent from a problem-solving method. A problem can be solved merely by describing "what is the problem," without describing "how to solve it." Separating them results in high productivity for modeling real-world problems. The effectiveness of this architecture is supported by a problem-solving method. The method generates a high-quality initial assignment using a novel algorithm, *Really-Full-Lookahead Greedy (RFLG)* algorithm, and refines it using a hill-climbing algorithm. Attaching importance to initialization results in producing a high-quality solution in a reasonable time for a large-scale tightly-constrained problem. This approach has been evaluated by successful experiments with practical high-school timetabling problems.

The following three sections describe the problem, the architecture, and the problem-solving method. Then, the experimental results are summarized and evaluated. Related work and conclusion are given in the last two sections.

Problem

COASTOOL is a general-purpose constrained problem solver. In this paper, attention is focused on *Constraint Relaxation Problems (CRPs)*. This section defines a CRP and describes high-school timetabling problems.

A CRP consists of a set of variables and constraints. A variable has a domain that is a set of values applicable for the variable. A constraint is associated with a set of variables. It not only has a condition to be satisfied, but also a *penalty point* to be incurred by a violation. A CRP is the same problem as a *Constraint Satisfaction Problem*, except that a constraint

[1] In this paper, "class" means a group of students.

Problem	# of lessons (# of variables)	# of time-slots (domain size)	# of classes	# of teachers [# of part-timers]	# of constraints [# of definitions]	Total penalty	Absence ratio
Kuki92	806	34	30	60 [9]	24,987 [54]	220,245	6.6 %
Kuki93	783	34	30	63 [10]	25,881 [80]	231,204	9.5 %
Daito93	633	32	26	69 [29]	26,014 [85]	266,508	33.0 %

Figure 1: Data for the Problems

```
1  (define-set Lesson
2     Math1-1a Math1-1b ··· Gym1-123a ···)
3  (define-set Time          Mon1 Mon2 ··· Sat1 ···)
4  (define-set SmithLesson   Gym1-123a Gym1-123b ···)
5  (define-set SmithAbsence Mon2 Mon3 ··· Sat1 Sat2)
6  (define-constraint PartTimerSmith
7    :objects ((:set lesson SmithLesson))
8    :variables ((v lesson))
9    :condition (not (is-a v SmithAbsence))
10   :penalty 10)
11 (define-problem Timetabling
12   :variables ((Lesson Time))
13   :constraints (PartTimerSmith ···)
14   :minimize TotalPenalty)
```

Figure 2: Problem Description Sample

has penalty and that its goal is minimizing the total penalty due to inconsistent constraints.

A high-school timetabling problem can be formalized as a CRP. Namely, a variable is a lesson and its domain is the set of all time-slots during a week. The problem is assigning a time-slot to every lesson considering various constraints. The teachers and classes of a lesson are given and referred to by constraints with the lesson.

COASTOOL provides a declarative description language for CRPs. Figure 2 shows a problem description sample. Lines 1–3 define a set of variables, `Lesson`[2], and a domain set, `Time`. A constraint definition, in lines 6–10, defines a set of constraints, named `PartTimerSmith`. It specifies that the constraints are defined on all elements of the set `SmithLesson`. Let `lesson` be a `SmithLesson` element. A constraint has a variable `lesson`. Let `v` be its value. The condition to be satisfied is that `v` is not a `SmithAbsence` element, and violating the constraint incurs the 10 point penalty. Lines 11–14 define a CRP, named `Timetabling`, that has a set of variables, set `Lesson`, with a domain, set `Time`, and constraints `PartTimerSmith`, etc. Object function `TotalPenalty` is pre-defined by the system.

In our case study, COASTOOL has been applied to three practical problems for two high-schools in Japan, as shown in Fig. 1. First, COASTOOL was applied to the '92 year timetabling problem (Kuki92) for Saitama-Prefectural Kuki-Hokuyou High-School. Since it is a mammoth high-school with various courses, the

[2]Simultaneous lessons are modeled as one lesson, e.g., `Gym1-123a` that corresponds to three gymnastic lessons for classes 1-1, 1-2, and 1-3.

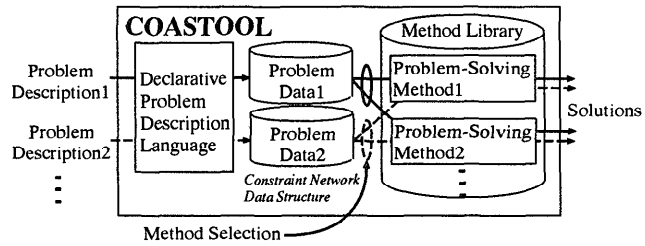

Figure 3: COASTOOL Architecture

timetabling is so difficult that it usually requires 100 person-days to accomplish. In problem Kuki92, satisfying all constraints was desired. It included a subtle constraint, `SeparateTwoCredits`, which specifies that two lessons for a two credits subject should be taken two days or more apart by a class.

Second, the next years timetabling (Kuki93) was also experimented on. Although it was similar to Kuki92, it was constrained more tightly, since it had a higher *absence ratio*[3] and more simultaneous lessons.

Finally, COASTOOL was experimented with another high-school, Daito Bunka University Dai-ichi High-School. Since it is a nongovernmental and academic high-school, the problem (Daito93) was quite different from those for Kuki92 and Kuki93. The absence ratio was very high (33%), because it had many part-time teachers and one training day without a lesson for each teacher. Moreover, since the high-school had many optional courses, there were many simultaneous lessons with various subjects. Daito93 was the most difficult problem and taking two same subject lessons in a day was unavoidable. However, the two lessons should be continuous, if they are taken in a day.

COASTOOL Architecture

COASTOOL is a general-purpose CRP solver. This section describes the constraint-based architecture for COASTOOL. As shown in Fig. 3, it has a declarative description language and a library of problem-solving methods. A problem can be solved by describing the problem and selecting a provided method. The problem description is translated into a common *network* data structure for constrained problems. Then, the selected problem-solving method solves the problem, processing the data structure.

[3] $\frac{\sum^{\text{teachers}}(\text{\# of absent time-slots for the teacher})}{(\text{number of teachers}) \times (\text{number of time-slots})}$.

A significant feature of COASTOOL architecture is that a problem description is declarative and independent from a problem-solving method. It results in a few advantages. First, the architecture provides high productivity for modeling real-world problems. This is because a problem can be solved merely by describing "what is the problem," without describing "how to solve it." Therefore, COASTOOL requires only a minimum amount of information about the problem.

Second, the architecture provides high maintainability. Since a problem description is declarative, a problem can be maintained by adding, deleting and/or modifying a small number of constraints.

The last advantage is that the architecture covers a wide range of applications. This is because a problem-solving method can be selected, according to the feature of a target problem. Moreover, changing a method does not require modifying a problem description.

Consequently, separating a problem description and a problem-solving method results in high productivity, high maintainability, and wide range coverage. However, these advantages rely on the efficiency of the problem-solving method. The following section describes an efficient problem-solving method for large-scale and tightly-constrained CRPs.

Problem-Solving Method

In the beginning of this section, previous general-purpose CRP solving methods are summarized and discussed, focusing on solving a large-scale tightly-constrained problem. Then, an efficient problem-solving method is proposed.

Background

Most general-purpose methods for constrained problems can be classified into consistency algorithms, backtracking, and optimization. Consistency algorithms and backtracking suffer from thrashing for a large-scale problem. In our case study, fast backtracking algorithms, *Forward-Checking Backtracking* and *Iterative Broadening* (Ginsberg 1992), assigned at most twenty percent of the lessons in a *one-week* run. In this paper, attention is focused on optimization methods.

Most optimization methods randomly walk around in problem search space under a bias in order to find a (sub)optimal solution. The bias strength effects computational time and optimality. Namely, a stronger bias results in more efficiency and less optimality.

One of the most common general-purpose optimization methods is *Simulated Annealing (SA)* (Johnson et al. 1989; Johnson et al. 1991). Since SA is a weakly biased method, it finds an optimal solution independently from an initial solution. However, the computational time is too long to use practically for high-school timetabling. For example, problem Kuki92 has a search space with 10^{1234} solutions and a good solution seldom exists. Walking such space under a weak bias results in a long tour.

```
1  Begin REALLY-FULL-LOOKAHEAD GREEDY ALGORITHM
2   Let Vars = set of all variables, ExVars = empty set.
        Var = NULL.
3   While Vars is not empty:
4    Let DsStore = copy of all domains of Vars.
5    Enforce Vars to be arc-consistent with Var.
6    If there is a variable V with an empty domain in Vars,
7     then remove V from Vars and add V into ExVars;
8          restore all domains of Vars from DsStore.
9     else Let Var = a variable in Vars;
10         Let Value = a value in Var's domain;
11         Assign Value to Var and remove Var from Vars.
12   end if
13  end while
14  While ExVars is not empty:
15   Let Var = a variable in ExVars.
16   Let Value = one of the least penalty values
                 in Var's original domain.
17   Assign Value to Var and remove Var from ExVars.
18  end while
19 End REALLY-FULL-LOOKAHEAD GREEDY ALGORITHM
```

Figure 4: Really-Full-Lookahead Greedy Algorithm

In (Minton et al. 1992), a large-scale loosely-constrained problem, a *Million Queens Problem*, has been solved using a *greedy* algorithm for initialization and *Min-Conflicts Hill-Climbing (MCHC)* method for optimization. Since MCHC is one of the most strongly biased optimization methods, it solves a large-scale CRP efficiently. However, as they reported in the paper, an inappropriate initial solution falls into a local optimum for a tightly-constrained problem.

Recently, Musick and Russell showed the importance of initialization theoretically (Musick & Russell 1992). Namely, the computational time grows exponentially, when the distance between an initial solution and an optimal solution exceeds a border value.

We have focused on developing a high quality initialization method and combining it with a strongly biased optimization method. The following subsection proposes an efficient general-purpose problem-solving method for large-scale and tightly-constrained CRPs.

Proposed Method

The proposed method is a combined method, involving a novel initialization algorithm, *Really-Full-Lookahead Greedy (RFLG)* algorithm, and a strongly biased optimization algorithm, Min-Conflicts Hill-Climbing (MCHC) in (Minton et al. 1992).

Figure 4 shows the RFLG algorithm. It has two steps. The first step (lines 3–13) assigns a value to each variable consistently using an *arc-consistency* algorithm. The arc-consistency removes values that cannot satisfy a constraint from the domains of variables in `Vars`. Using it avoids assigning a value that is consistent with already assigned variables, but causes later inconsistency with an *as-yet-uninstantiated* variable.

For example, let `Gym2-1a` and `Gym2-1b` be contin-

uous lessons taught by a teacher who will be absent Mon2. A simple greedy algorithm may assign Mon1 to Gym2-1a, because it has no immediate violation until Gym2-1b will be instantiated. In the RFLG algorithm, Mon1 is removed from the domain of Gym2-1a by arc-consistency before selecting a value for the variable.

The first step never instantiates an inconsistent assignment. If there appears a variable V with no consistent candidate, the step suspends assigning a value to V, excludes it from the consistent set of already instantiated variables, and stores V in ExVars. Then, it continues to instantiate other consistent assignments until there will be no more consistent assignments.

If there is no more consistent assignment, the RFLG algorithm proceeds to the second step (lines 14–18). The step assigns inconsistent but the least penalty values to the excluded variables, using a common *greedy* algorithm. In addition, ties are broken randomly in selection of variables and values in both steps (lines 9–10 and 15–16).

A significant point for the RFLG algorithm is that it uses arc-consistency for *constraint relaxation*, not for *constraint satisfaction problems (CSPs)*. Arc-consistency algorithms are commonly used in backtracking for CSPs. They are useful for reducing the number of backtracks, because they prune inconsistent branches in a search tree. In the case of the RFLG algorithm, they are useful for reducing the number of excluded (inconsistent) variables for the same reason.

In (Nadel 1988), Nadel unified several backtracking algorithms as combinations of pure *tree search* and arc-consistency. From this point of view, the RFLG algorithm can be considered as a combination of pure *generation* and arc-consistency. It never backtracks but corresponds to *Really-Full-Lookahead Backtracking*, since both use *full* arc-consistency. Moreover, this idea can be extended to develop several generation algorithms with *partial* arc-consistency, e.g., *Forward-Checking Greedy* algorithm.

Following backtracking techniques, variable and value ordering heuristics, as well as arc-consistency, can be combined into an RFLG algorithm. Our implementation uses *Most-Constrained-Variable-First* heuristic for variable selection in line 9. Several ordering heuristics, e.g., *Most-Constraining-Variable-First*, can be used in order to reduce the number of inconsistent variables. Consequently, the RFLG algorithm generates a high-quality initial assignment, employing several backtracking techniques.

It should be mentioned that COASTOOL has a program synthesizing facility that produces an efficient arc-consistency program for a given CRP. The problem description language compiles a condition form for a constraint definition into an efficient constraint propagation procedure, that will be called by a generic arc-consistency program (Yoshikawa & Wada 1992). This is similar to AC-5 in (Deville & Hentenryck 1991).

The proposed CRP solving method refines an initial assignment, generated by the RFLG algorithm, using MCHC, a strongly biased optimization algorithm. MCHC repeats selecting an inconsistent variable and modifying its value to one of the least penalty values, breaking ties randomly. It efficiently finds a local optimum near the initial assignment, even for a large-scale problem. Although it falls into a local optimum, applying it on a high-quality initial assignment results in a global (sub)optimal solution. Consequently, employing the RFLG algorithm and MCHC results in producing a high-quality solution in a reasonable time for a large-scale tightly-constrained problem.

Experimental Results

This section evaluates the proposed approach through experiments with practical high-school timetabling problems, Kuki92, Kuki93, and Daito93.

Experiment with Kuki92

In Kuki-Hokuyou High-School, ten teachers usually work on timetabling for ten days. We interviewed a key person to determine the data and requirements. Then, the problem was solved using COASTOOL. Constructed timetables were evaluated by the key person.

COASTOOL Architecture The interview took three hours and only three person-days were required to describe the problem. The description includes 1,500 lines of data definitions and 500 lines of problem specification. There were 54 constraint definitions (See Fig. 1). The results show that the productivity for modeling real-world problems is considerably high, compared with expert systems approaches.

Two difficulties became clear. First, a *global* constraint[4] cannot be modeled declaratively in a CRP or cannot be solved efficiently. For example, a teacher should have three or four lessons at intervals everyday. This preference was specially considered in tie-breaks. Second, set definition language structures are too simple to describe various data conveniently. A frame language or a domain-specific interface are worth being considered to be integrated.

Problem-Solving Method Figure 5 shows the transition of total penalty by the proposed method (RFLG-MCHC), the combination of Greedy algorithm and MCHC (G-MCHC), and the combination of Greedy algorithm and Simulated Annealing (G-SA). G-MCHC falls into a local optimum solution in several minutes. The teacher evaluated that the completion degree was less than 70%.[5] G-SA was experimentally applied with several values for cooling parameters. However, SA is too slow to construct a practical

[4] A constraint with all variable or a constraint with which related variables are determined dynamically according to the assignments.

[5] Namely, the solution had a lower quality than those at 70% of a human timetabling task.

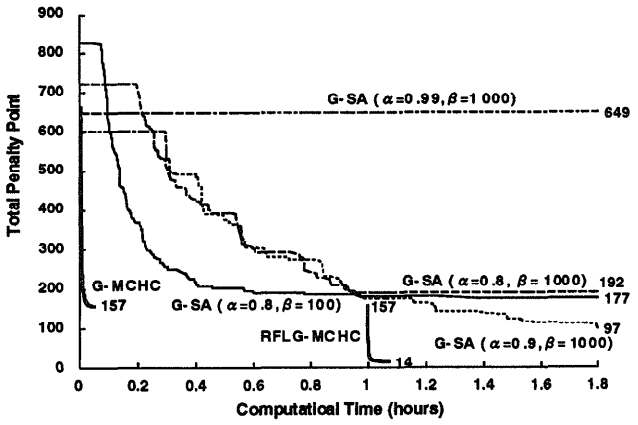

Figure 5: Experimental Results for Problem Kuki92: In SA, α is cooling ratio and β is the number of repeating times at the same temperature.

timetable. It required at least three days to reach the 90% completion point.

With the proposed method, although the RFLG algorithm required about one hour, it generated a high-quality initial assignment. MCHC refined it and constructed an almost perfect timetable (more than 95%). There was the best solution with only one inconsistent constraint (`SeparateTwoCredits` described earlier) in sixty trials. Consequently, the method constructs an almost perfect timetable in a reasonable time. It is considerably faster than G-SA and results in a considerably higher quality timetable than G-MCHC.

Other combinations of initialization and optimization should be considered. RFLG-SA is nonsense, because SA walks to much worse solutions from an initial assignment at first. Figure 5 looks as if it is worth applying MCHC on a solution in the middle of SA cooling. However, such a solution resulted in an unacceptable local optimum, according to our experiences.

Total Performance As a whole, it required only three person-days to describe the problem Kuki92, and an almost perfect timetable was constructed in about one hour. Consequently, using COASTOOL reduced the cost for timetabling from one hundred person-days to less than one person-week.

Experiment with Kuki93

We experimented with the high-school for the second straight year (Kuki93) in order to confirm the successful result.

Maintainability

The problem description for Kuki92 could be modified for Kuki93 by only 1.5 person-days work. Most of the maintenance was data maintenance. The problem specification could be modified, merely by adding, deleting and rewriting a number of constraints.

In addition, the problem data or requirements were modified several times in the middle of the timetabling task. For example, teachers in charge of lessons were changed according to their assignability. Every minor change required only about ten minutes. These results show the high maintainability of COASTOOL.

Partial Arc-Consistency The computation was also successful and practical. It required about 2.4 hours to construct a timetable at 90% (76 penalty points). However, the RFLG algorithm required 2.2 times as long computational time as that in Kuki92, even though they were problems for the same school. This is because the RFLG algorithm uses an arc-consistency algorithm repeatedly.

Although the result was successful enough, we tried to decrease the computational time, using arc-consistency *partially*. Namely, only important variables and constraints were processed by arc-consistency. The other variables were assigned in the second step of the RFLG algorithm. The other constraints were considered in value selection. As a result, the proposed method constructed a timetable at 70% (257 points) in 26.2 minutes. The RFLG algorithm has flexibility to tune smoothly the trade-off between efficiency and optimality.

Experiment with Daito93

As the final experiment, we applied COASTOOL to the timetabling for a different kind of high-school, Daito Bunka University Dai-ichi High-School. Usually, a teacher uses a timetabling software, SHIRAKU (Shiraku), and works on the problem for ten days. SHIRAKU's input files were converted into a COASTOOL problem description.

Phased Approach Since Daito93 is much more difficult than "Kuki" problems, 2.8 hours were required for achieving a 70% complete timetable. The proposed method fell into an unacceptable local optimum.

We tried a *phased* approach, following the teacher's expert manner. Namely, important lessons were assigned first and modified by a human, then the others were assigned and modified. It required 37.6 minutes for the first assignment, twenty minutes for modification, 91.5 minutes for the second assignment, and three hours for modification. The modification was done by a non-expert person using a graphical user interface and produced a 90% complete timetable. Consequently, COASTOOL reduced the total cost from ten days work by an expert to four days with a non-expert person.

It should be mentioned that the human modification repeated a kind of *local search* that finds a plan to repair several lessons to satisfy constraints, according to constraint priority. At least all absolute requirements must be satisfied. Since the plan temporarily passes higher penalty solutions, it can not be simulated by a hill-climbing method. Automating such planning is interesting work for the future.

From our case study, the results illustrate that COASTOOL is efficient as a base for practical applications in high-school timetabling problems.

Related Work

High-school Timetabling School timetabling has been studied for the last three decades, based on *Operations Research (OR)* techniques, e.g., (Gotlieb 1963; Even, Itai, & Shamir 1976; Carter 1986). Since an OR technique depends on a simple problem definition, they might fail to solve real-world complex problems. There were also *Genetic Algorithm* (Corne, Fang, & Mellish 1993) and *Backtracking* (Feldman & Golumbic 1989) approaches.

Most previous studies focused on university or examination or student timetabling, in which a student might have free time in a scheduling period. High-school timetabling is constrained much more tightly, since a student has no free time-slot during a week. The authors do not know about any successful work on practical high-school timetabling problems.

COASTOOL Architecture *Constraint Logic Programming* languages (Cohen 1990) provide declarative frameworks to describe and solve a problem. Our previous system (Yoshikawa & Wada 1992) and MULTITAC (Minton 1993) also provide a declarative framework and a problem-solving program synthesizing facility. However, they have not used an iterative optimization method yet. Although ODO in (Davis & Fox 1993) is the most closely related work, it is focused on Job-shop Scheduling.

Problem-Solving Method In (Minton et al. 1992), Minton et al. developed an initialization method for *Graph Coloring Problems*, using problem-specific ordering heuristics. The basic idea may be similar to the RFLG algorithm.

According to (Haralick & Elliot 1980), partial use of arc-consistency is important for the backtracking efficiency. From our experience, full use of arc-consistency is important for the quality of initialization by the RFLG algorithm. Evaluation of the RFLG algorithm with partial arc-consistency and variable and value ordering heuristics is interesting work for the future.

Conclusion

A general-purpose *Constraint Relaxation Problem* solver, COASTOOL, and its successful experiments on practical high-school timetabling problems have been described. Concerning COASTOOL architecture, separating "what is the problem" and "how to solve it" results in high productivity for modeling real-world problems. In order to solve large-scale tightly-constrained problems, a high-quality initial assignment is required by a hill-climbing algorithm and it can be obtained using arc-consistency. The successful experiments indicate that COASTOOL is efficient as a base for practical applications in high-school timetabling problems.

Acknowledgements

The authors would like to thank Masahiro Yamamoto and Takeshi Yoshimura of NEC Corporation for encouragement to pursue this research. They also thank Kin-ya Ohuchi, Mikio Nakabayashi, and Mitsuhiro Masuda of Kuki-Hokuyou High-School and Yosuke Kamine of Daito Bunka University Dai-Ichi High-School for cooperation in the case study.

References

Carter, M. W. 1986. A Survey of Practical Applications of Examination Timetabling Algorithms. *Operations Research* 34(2): 193–202.

Cohen, J. 1990. Constraint Logic Programming Languages. *Communications of the ACM* 33(7): 52–68.

Corne, D.; Fang, H. L.; and Mellish, C. 1993. Solving the Modular Exam Scheduling Problem with Genetic Algorithms. In Proceedings of the Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Eepert Systems.

Davis, E. and Fox, M. 1993. ODO: A Constraint-based Architecture for Representing and Reasoning About Scheduling Problems. In Workshop Notes of the IJCAI-93 Workshop on Knowledge-Based Production Planning, Scheduling, and Control, 325–330.

Deville, Y. and Hentenryck, P. V. 1991. An Efficient Arc Consistency Algorithm for a Class of CSP Problems. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, 325–330.

Even, S.; Itai, A.; and Shamir, A. 1976. On the Complexity of Timetable and Multicommodity Flow Problems. *SIAM J. Comput.* 5(4): 691–703.

Feldman, R. and Golumbic, M. C. 1989. Constraint Satisfiability Algorithms for Interactive Student Scheduling. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 1010–1016.

Ginsberg, M. L. 1992. Iterative Broadening. *Artificial Intelligence* 55: 210–215.

Gotlieb, C. C. 1963. The Construction of Class-Teacher Time-Tables. In Proceedings of IFIP Congress 62, 73–77.

Haralick, R. and Elliot, G. 1980. Increasing Tree Search Efficiency for Constraint-Satisfaction Problems. *Artificial Intelligence* 14(3): 263–313.

Johnson, D. S.; Aragon, C. R.; McGeoch, L. A.; and Schevon, C. 1989. Optimization by Simulated Annealing: Experimental Evaluation; Part I, Graph Partitioning. *Operations Research* 37(6): 865–892.

Johnson, D. S.; Aragon, C. R.; McGeoch, L. A.; and Schevon, C. 1991. Optimization by Simulated Annealing: Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. *Operations Research* 39(3): 378–406.

Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1992. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence* 58: 160–205.

Minton, S. 1993. Integrating Heuristics for Constraint Satisfaction Problems: A Case Study. In Proceedings of the Eleventh National Conference on Artificial Intelligence, 120–126.

Musick, R. and Russell, S. 1992. How Long Will It Take? In Proceedings of the Tenth National Conference on Artificial Intelligence, 466–471.

Nadel, B. 1988. Tree Search and Arc Consistency in Constraint-Satisfaction Algorithms. In *Search in Artificial Intelligence*, eds. Kanal, L. and Kumar, V., 287–342, New York: Springer-Verlag.

Shiraku II Ver.2.1 Users Manual (In Japanese), 1992. Yasukawa Information Systems Corporation, Saitama, Japan.

Yoshikawa, M. and Wada, S. 1992. Constraint Satisfaction with a Multi-Dimensional Domain. In Artificial Intelligence Planning Systems: Proceedings of the First International Conference (AIPS92), 252–259.