The U.S. National Football League Scheduling Problem

Bistra N. Dilkina and **William S. Havens**
Intelligent Systems Lab
Simon Fraser University
Burnaby, British Columbia
Canada V5A 1S6
and
Actenum Corp.
Vancouver, British Columbia
{bnd, havens}@cs.sfu.ca

Abstract

We describe the problem of scheduling the television broadcasts of the U.S. National Football League (NFL). Unlike traditional round-robin tournament scheduling, the NFL problem involves assigning games to broadcast slots under various complex constraints while attempting to satisfy a set of user preferences. As well, a mixed-initiative functionality was required to allow the user to control and assist in the scheduling process. A prototype system was developed for the NFL which produced schedules satisfying many of these constraints and preferences. In this paper, we provide an overview of the constraint solving methodology employed and the implementation of the NFL prototype system.

Introduction

Sports league scheduling is very important for both professional and amateur sports alike. Obtaining good ”playable” schedules under a myriad of league and logistics constraints is extremely difficult yet essential to the success of the league. Thus developing practical techniques for this problem is a very important research area.

Sports league scheduling is particularly interesting type of constraint satisfaction problem (CSP) which has attracted significant research interest (Henz 1999; Nemhauser & Trick 1998). The CSP, in general, and sports scheduling, in particular, are NP-hard requiring exponential computing time in the size of the problem. Developing algorithms capable of solving large instances of these problems has proven to be extremely difficult.

In this paper, we report on the research and development of a prototype system (called *NFLSched*) to solve a very large and difficult sports league scheduling problem for the U.S. National Football League (NFL). This problem differs from the usual round-robin league scheduling in significant ways. The NFL problem is very large comprising 32 teams which play 256 games in a 17 week broadcast schedule. Indeed, the matches (games) are already known ahead of time by agreement between the teams and television broadcast networks. What must be scheduled is the assignment of games to broadcast slots while satisfying a large number of diverse constraints and maximizing a set of preferences. Furthermore, it was desired to intimately involve the user in the scheduling process. The NFL wants to be able to ”tweak” schedules during the scheduling process to obtain preferred broadcasts of particular games in particular slots in the schedule while continuing to satisfy the hard ”playability” constraints. Thus the NFL problem is a mixed-initiative constraint optimization problem.

The NFL scheduling system is required by the league's Broadcasting Department to produce their regular season game schedules. The Broadcasting Department works with the four major U.S. television networks (ABC, CBS, FOX and ESPN) to obtain a broadcast schedule acceptable to all parties. The NFL desired that the scheduling application be able to (sub)optimally schedule professional football games given multiple (and often conflicting) broadcast, physical and fairness constraints. Additionally, the system must support building flexible network game packages and be able to modify the rules and constraints as necessary to find preferred playable schedules which are acceptable to both the NFL teams and the broadcast networks.

Previously, the league schedules were produced by a human expert. But the complexity of the problem has increased (both in size and type of constraints) necessitating automation. Earlier attempts at exhaustive backtrack search for completing (and verifying) partial schedules were unsuccessful. Thus the NFL requested proposals for developing a new mixed-initiative league scheduling application. Of the approximately forty respondents, two groups were selected to build prototype applications. Our NFLSched application was one of these two successful solutions.[1] The application was designed and implemented in a very short 10 week time frame using a proprietary constraint programming environment called ReSolver developed by Actenum.

Our analysis is that the underlying CSP is critically constrained. Many of the constraints reflect the business relationships between the NFL and the television networks. These constraints are often mutually conflicting requiring that “playable” solutions actually break a significant number of these constraints. Our approach to this problem has been to exploit the mixed-initiative capabilities of ReSolver. In particular, we have designed a system which involves the

[1] The other system was developed by a collaboration of Sengen of Marlton, New Jersey and ILOG S.A. of Paris, France.

user (human scheduler) directly in the scheduling process. Both the user and the system work together to search for playable solutions but always under user control. The user can explore different solution possibilities, adding and removing constraints and preferences as the search proceeds. Thus schedules can be produced which include as many desirable properties as possible while still being feasible (playable).

In the remainder of this short paper, we describe the NFL problem in more detail and characterize it as a mixed-initiative constraint satisfaction problem with preferences. Next we describe the constraint solving methodology used to search for solutions to the problem. Some practical details of the NFLSched application are also provided including the essential mixed-initiative (tweaking) functionality. Finally, we conclude with the status of the prototype, the lessons learned and the need for mixed-initiative tools like NFLSched to assist users in finding good solutions to very hard complexity problems.

Problem Description

The NFL league has two divisions: NFC and AFC of 16 teams each for a total of 32 teams. Each team plays 16 games during the season both within its division and across divisions. So each team will play 8 home games and 8 away games against the same 8 opponents. The actual pairings of teams into games is decided before the scheduling begins. There are a total of 256 games played across both divisions in the full season. The season lasts 17 weeks with each team playing at most once per week. Each team necessarily will have a "bye" week once during the season which must occur between weeks 3-10 of the season.

The four broadcast networks (CBS, FOX, ESPN, and ABC) divide the games among their respective television shows. CBS and FOX both purchase blocks of 128 games each. Some of these are resold to the ESPN and ABC networks. The broadcast slots comprise Monday Night Football (ABC), ESPN games, FOX/CBS Sunday Double Headers, CBS Sunday and FOX Sunday games. The number of games per slot varies from week to week depending on the match-ups, geography and other constraints. For example, there may be between 0 and 2 games broadcast by ESPN per week while CBS may broadcast between 4 and 7 simulcast regional games on any particular Sunday.

Constraint Satisfaction Problem Definition

The NFL league scheduling problem involves creating a schedule such that a set of games is assigned to television network timeslots with limited capacities within a 17 week season. The NFL season is a combination of two types of problems: constraint satisfaction and optimization. It involves hard constraints, those that need to be fulfilled by any schedule that would be acceptable, and soft constraints, those desirable properties that not need be necessarily fulfilled. Hence the most preferred solution will satisfy all hard constraints and break the least possible number of soft constraints.

Formally defined, a Constraint Satisfaction Problem (CSP), is a triple (V, D, C), where $V = \{v_1, ..., v_n\}$ is a set of variables with corresponding finite domains of possible values for each variable $D = \{D_1, ..., D_n\}$ and C is a set of k-ary hard constraints. A k-ary constraint defines the allowed combinations of values for a subset of k variables $(v_1, ..., v_k)$ from V. A solution to a CSP consists of assigning a value to each variable so that all constraints are satisfied.

The soft constraints, which arise in any real-life problem, account for the quality of the solution. NFL has a number of soft rules that bring penalty points to any solution that does not satisfy them. Finding the best fulfillment of these rules is an optimization problem.

The constraints defined by the NFL are complex, involving many varibles and over various characterstics (games, teams, timeslots) of the schedule. In addition, in our analysis they critically constrain the solution space. Hence, modeling this problem adequately was very difficult. We considered several CSP models for the NFL Scheduling problem: a grid of variables, the 32 teams *versus* the 17 weeks, assigning the opponent away teams (the usual model for Round-Robin schedules); a grid of the home teams *versus* away teams, assigning the timeslots, *et cetera*. We note that there are exactly 92 possible broadcasting timeslot types based on the week, network and broadcast time combinations. Each game needs to be assigned a timeslot. However, since the broadcast rights to each game are predetermined by the NFL ahead of time, each game can only be played in the timeslots of the owning network, which are usually one timeslot time per week such as the ESPN broadcast. Only the doubleheader games of FOX and CBS can be played in two different timeslots in a week. Thus, almost all games variables have 17 possble timeslot type assignments. In fact, we exploit the fact that several games, such as the Monday Night Football games, have fixed or almost fixed timeslots. This reduces the research space, however, our estimate of its size is still approximately 10^{40}. Since each game has the home team and away team attributes, it was easy to express constraints over teams by simply selecting the games that involved these teams. In additional timeslot capacity constraints (such as CBS Sunday can show between 4 and 7 games in a week) were easily expressed over only the games that could be played in the given timeslot type. This model was selected because it compares to the other models in search space size, however it provides for easier incorporation of the various types of constraints and avoids composite values for variables.

Pre-Defined Constraints and Scoring Rules

The search for candidate solutions is guided by league-specified constraints and scoring rules. Basically, the scheduling constraints determine the playability of a candidate schedule, while the scoring rules quantify the quality of the schedule.

Stadium Blocks

Stadium blocks are used to determine when each stadium is available for home games: unavailable (hard); available

but other events make playing there undesirable (soft); the stadium is available but other events may require the game to be moved (swap). The stadium blocks on the home team of a game determines when it can be played.

Home/Away Spacing

Home/away spacing constraints can be constructed to ensure that all teams play a reasonable home/away pattern over the course of the season. There are a number of home/away spacing constraints including:

- Teams cannot play 3 consecutive home/away games during weeks 1-5
- Teams cannot play 3 consecutive home/away games during weeks 15-17
- Teams may play 1 set of 3 consecutive home/away games during weeks 4-16, but the schedule will receive 1 penalty point (Soft constraint)
- Teams must play at least 2 home/away games every 6 weeks
- Teams must play at least 4 home/away games every 10 weeks
- Teams cannot play 4 consecutive home/away games

Opponent Spacing

When teams play each other twice in a season, as is the case for divisional opponents, the preference is for the two games to be spaced reasonably far apart. Ideally, the games should be at least 6 weeks apart with at least one game after the 8th week of the season:

- Any two opponents may only play 1 game every 8 weeks (at least 1 would have to be after week 8). If this constraint is violated, the schedule receives 1 penalty point. (Soft constraint)
- Any two opponents may only play 1 game every 3 weeks (cannot be violated).

Shared Hometowns

For teams that share a stadium (New York Giants / New York Jets), the schedule imposes constraints that the two teams may not play at home on the same day.

In addition, for the teams that share a local viewing audience (New York Giants / New York Jets, San Francisco 49ers / Oakland Raiders) the schedule ensures that the two teams:

- Do not play at the same time
- Do not play on the same network (i.e. CBS or FOX) on the same afternoon

Scheduling Strategy

The mixed-initiative paradigm involves the user in the solving process at several levels. The user can dynamically add and remove new constraints to the problem description. For example he may wish to enforce one of the preferences as a hard constraint. However, a human user can easily over-constrain the problem by adding additional constraints. Unfortunately most constructive methods do not support explanations (Jussein & Lhomme 2002) and nogood learning but are based on the tree search paradigm. Thus, such a method would simply report a failure to find a solution without any insight for the user about what went wrong.

Further, given the domain expertise in sport scheduling which is hard to convey to the programmers or incorporate in the CSP model, a user interaction during the search process itself can greatly facilitate finding the most desirable schedules (a property that cannot be fully grasped by the formal preferences). However, constructive search does not readily support dynamic constraint addition and retraction during the search process. In addition, working with the partial schedule produced during constructive search is hard because of the hidden hard constraints over the remaining unassigned variables.

All of these characteristics of the NFL problem called for a local search solution which works on a full (although) inconsistent assignment. In this environment, the user could work together with the search algorithm to fix the inconsistent parts of the schedule by manually (re)assigning variables. We developed a new hybrid constraint solving schema, called *systematic local search* (Havens & Dilkina 2004), which retains some systematicity of constructive search. Our method backtracks through a space of complete but possibly inconsistent solutions while supporting the freedom to move arbitrarily under heuristic guidance.

There has been various recent research into hybrid search schemes which combine desirable aspects of constructive and local search methods (Freuder & Wallace 1992; Ginsberg & McAllester 1994; Jussein & Lhomme 2002; Lynce, Baptista, & Marques-Silva 2001; Minton *et al.* 1990; Morris 1993; Prestwich 2001). In the work reported here extends these methods. The scheme operates from a nearly complete instantiation of values to variables (Ginsberg & McAllester 1994; Prestwich 2001). Forward checking of both assigned and unassigned variables is performed (Prestwich 2001). In this process we maintain a count of the number of constraints disallowing each domain value of every variable. Thus reassignment of a variable only involves subtracting the effects of the previous assignment and adding the effects of the new assignment to all neighbour variables. The hill-climbing gradient is the number of constraint violation. Using the minconflict heuristic (Minton *et al.* 1990), every variable chooses the value with the least constraint violations. This notion of a maximally consistent solution relaxes the requirement that the constructed partial solution remain consistent (Freuder & Wallace 1992).

The drawback of local search methods is that they suffer from local maxima and cycling. Different diversification methods have been developed to avoid these undesirabe effects such as simulated annealing, tabu lists(Glover 1990), random restart(Selman, Levesque, & Mitchell 1992), etc. These work extremely well especially on problems with adequate solution density.

Given that the NFL problem is critically constrained, we desire not only diversification but also some amount of sys-

tematicity that will guarantee that large portion of the search space is explored and that revisiting of old solutions is rare. In our schema, systematicity is enforced using a nogood cache of known inconsistent variable assignments (Ginsberg & McAllester 1994; Havens 1997; Jussein & Lhomme 2002; Stallman & Sussman 1977). These inconsistent variable assignments represent the violating constraint tuples at each local maximum as explicit new constraints guaranteeing that the local maximum will not be revisted in the future.

The nogoods provide the diversification needed to break away from local maximum and at the same time the use of randomized (arbitrary) backtracking (Gomes, Selman, & Kautz 1998; Jussein & Lhomme 2002; Lynce, Baptista, & Marques-Silva 2001; Yokoo 1994) preserves the freedom to move for every assigned variable.

Our Maximal Constraint Solving schema searches heuristically through a space of maximally consistent variable assignments while backtracking on assignments which are not acceptable solutions. It discards the maintenance of a totally consistent partial solution. Variables use value ordering heuristics to choose a *maximal assignment* from their live domain of allowed values (i.e. assignments not prevented by a known nogood). If no allowed values remain for some variable then that variable induces a nogood and backtracks. When all variables have chosen a maximal assignment, these assignments constitute a *maximal solution*. Such a solution is a mutual local maxima for every variable. If the maximal solution does not exhibit full consistency then the solution induces nogoods and again the system backtracks.

The systematic local search algorithm is listed in Figure 1. Given a set of variables, V, and constraints, C, $solve(V, C)$ returns the first solution, α, whose assignments satisfy each constraint in C. The algorithm operates as follows. In the initial global assignment, α, (line 2) every variable is unassigned. The loop beginning in line 3 is repeated until every variable assignment in α is assigned the best value (MAXIMAL) and is still allowed (in line 13). Then the solution, α, is returned in line 14. While every variable is not MAXIMAL (line 4), a variable, x, is chosen via the variable ordering heuristic, $select(V)$. Then the variable, x, is assigned the best allowed domain element. However, if an *empty nogood* is ever derived (line 7) then the algorithm returns failure indicating that no solution exists.

When every variable is MAXIMAL, the current global assignment, α, is a maximal solution but may not be a consistent solution. Beginning in line 9, for every constraint, $c \in C$, which is not satisfied, a new nogood, $\lambda_\perp$, is derived from c and added to the nogood cache, Γ in line 11.[2]

The use of nearly complete instantiation of variables, of nogood recording and explanations allowed for the mixed initiative paradigm in which the user could stop the engine at any time and look at the current state of the solution and constrain additionally the schedule thus guiding the engine towards feasible solutions. In addition, once a playable solution is found the user could modify it, and hence breaking

[2]Without an infinite nogood store, the algorithm stills remains incomplete since it does not systematically cover the entire search space.

```
1  function solve(V, C) {
2     α =all variables unassigned
3     repeat {
4        while (α ≠ MAXIMAL) {
5           let x = select(V);
6           assign(x);
7           if empty nogood is derived return noSolution;
8        }
9        ∀c ∈ C s.t. c is inconsistent {
10          λ⊥ = failure(c);
11          add λ⊥ to Γ, the nogood store;
12       }
13    } until (α = MAXIMAL);
14    return α;
15 }
```

Figure 1: The $solve$ algorithm finds a consistent solution of the variables V for the constraints C.

some constraints, and then the search algorithm would try to find the closest feasible solution incorporating the user decisions by iteratively fixing the broken constraints by following again the minconflict gradient. We provided a simple integration of soft constraint in the prototype by allowing users to dynamically add these constraints to the problem and try to find feasible solutions to the further constrained problem.

Application Description

Our solution consists of three (3) separate modules:

- The Database Module - used for storing scheduling data, constraints, scoring rules, and completed solutions
- The User Interface Module - an easy to use GUI for defining any number of global or team-specific scheduling constraints
- The Intelligent Interactive Search (IIS) Module - an interactive scheduling grid for generating, modifying or 'tweaking, viewing, and printing candidate schedules

Database Module (Schedules, Constraints, Scoring)

The database module, which is Oracle based, maintains all data pertinent to the scheduling application. It is connected to both the User Interface Module and the IIS Module and contains four independent datasets: scheduling parameters, pre-defined scheduling constraints and scoring rules, interactive constraints, and completed schedules.

Each of these datasets contains two uniquely defined indexes, [Season] and [Scenario], which allow users to create and isolate data for any number of "what-if" situations complete with names, date/time stamps, security levels, etc.

Handling these datasets independently allows users to:

- Apply different scoring schemes to current/past schedules
- Interactively modify or "tweak" schedules

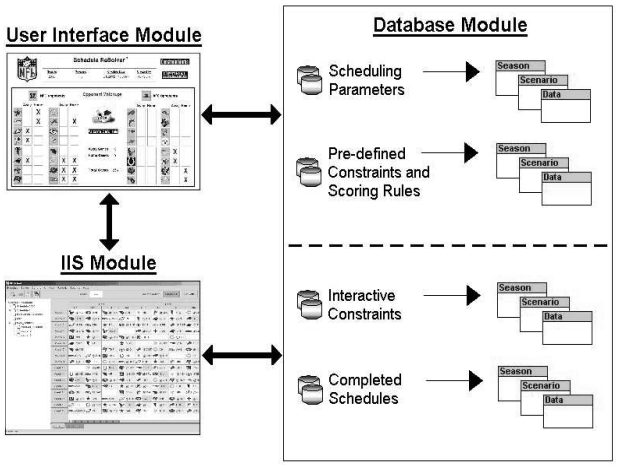

Figure 2: Application Architecture

.

User Interface Module

The schedule definition parameters are the datasets required to build the scheduling framework. They include:

- Season data: Schedule weeks, bye allocation, etc.
- Team data: Team logos, divisional & conference alignments, home/away opponents, etc.
- Network data: Network slots, package requirements, allowable game/slot combinations, etc.

In addition to tying together the Database Module and the IIS Module, the User-Interface Module or "front-end" allows users to:

- Construct the schedule definition requirements and data required for a new season or "what if" scenario
- Modify schedule-definition parameters such as season data, team data, and network data
- Modify scheduling constraints and scoring rules

Intelligent Interactive Search Module

The Intelligent Interactive Search (IIS) Module is the interface between the user and the search engine. It allows users to:

- Initiate the search for solutions
- Add aditional constraints
- Interact with the schedule during search
- Save/Restore schedules

It provided the user with different views of the schedule that made it easier to assess it and interact with it. The views are:

- Games: a grid of game cells including the home and away team, the network and asigned timeslot
- Teams vs. Weeks: 2 grids for the 2 divisions respectively of 16 teams vs. 17 weeks. Each cell included the opponent in the game played by the team in the given week (or 'bye'), the status (home or away), the network, and timeslot
- Networks vs. Weeks: the list of games owed by the particular network played in the given week

Pre-Defined vs. Interactive Constraints

Our proposed solution allows users to define two different types of constraints: pre-defined and interactive. Essentially, pre-defined constraints are used for general schedule requirements, and are constructed using the User Interface Module, while interactive constraints are used for tuning schedules in the IIS Module. The scheduling constraints and scoring rules datasets will be used to guide the scheduling application towards playable and fair "initial" schedules, which may then be tweaked by the user.

The User Interface Module allows users to construct any number of pre-defined constraints and scoring rules that affect stadium blocks, home/away spacing, opponent spacing, game spacing (rest days), shared hometowns, network appearances, game parings, and general fairness rules.

In addition, many of these constraints and scoring rules are "dialable", meaning that they may be adjusted or turned off for any or all of the teams, depending on the scheduling scenario.

Intelligent Interactive Search system offers the user the ability to explicitly specify the date or network of any match-up (or range of match-ups) during the interactive process. The details of these interactive constraints are stored in a dataset and can be can be disabled at any time as desired.

Schedule Interaction

Our proposed solution allows users to interactively add/remove/modify constraints as a way of exploring the solution space to ultimately to find "better" schedules (keeping in mind that better schedules do not always correspond with lower scores). The search engine uses the existing solution as a starting point and searches for solutions that observe the modified constraints.

Any of the following constraint changes may be made interactively to help the user improve upon a found schedule:

- Changes to stadiums blocks or the forcing of bye weeks
- "Manual" game assignments
- Global or team-specific changes to home/away spacing, divisional opponent spacing, game pairing or fairness constraints
- Changes to allowable game/slot options

Application Development and Evaluation

The appication was developed using $Java\ 2\ 1.3$ with $Swing^{TM}$ components using $Borland\ JBuilder^{TM}$ 7 under different operating systems(Sun Microsystems, Macintosh and Windows). The CSP model and search sheme were implemented in $ReSolver^{TM}$, a constraint programming library owed by Actenum Corporation.

The *NFLsched* prototype was build with extremely limited resources: only 10 weeks of development time, 5 programmers and a sports scheduling consultant [3]. The modeling of the problem and the definition of the constraints from the NFL specifications required domain expertise that the development team lacked in the beginning. The longest and hardest development stage was designing the search algorithm. In addition, an important part was designing meaningful graphical interface that allowed the user to visualize the schedule from different perspectives and conviniently interact with it.

The system was able to find within 1 to 2 hour playable schedules of preference quality around 30 when run on an COMPAQ 1.6G MHz PC with 500MB RAM. However, we estimate that when used by a domain expert it can find schedules faster and of better quality. It was tested on the season scheduling data from 2002 provided by NFL at the start of the development period. Finally, the NFLSched application was compared in performance to the other prototype system on the scheduling data for 2003 provided one day in advance of the testing date. Both systems were run as a batch solver and both were able to find feasible solutions in about one hour. The mixed-initiative capabilities of our system were not used in the trial test and the final contract was awarded to our competitor.

The prototype developed provides easy interface for entering the season data such as games selected, network rights, hard constraints, etc. The mixed-initiative functionality and dynamic constraint support allows for easy experimentation with different rules, preference scoring, for intimate particiaption in the scheduling process and use of human exprtise. In particular, satisfying additional broadcast preferences would have substantial monetary advantage.

Conclusion

In this short paper, we provided a specification of the NFL scheduling problem, its model as a constraint optimization problem, and the description of a prototype scheduling application. Our solution used the constraint programming paradigm applying techniques both from Artificial Intelligence and Operations Research such as constraint propagation, nogood learning and inference, and local search.

The development of this prototype required the innovation of a new hybrid search schema, described in detail in (Havens & Dilkina 2004). In addition, it raised important research issues that have not been currently adequately addressed such as mixing hard satisfaction constraints with soft constraints or preferences. There has been recent work on solving soft CSPs, but not for what one can call *'hybrid' CSP*, which have the two distinct categories of constraints, those that necessarily must hold and those that could be unpreferably broken. We discovered that the mixed-initiative paradigm calls for further understanding and research in the area of explanations (Jussein & Lhomme 2002).

Despite the complexity and the unusual nature of this sport scheduling problem, we were successful at building a functional prototype that can find playable schedules for the NFL league.

[3] Rick Stone, President of Optimal Planning Solutions

References

Freuder, E., and Wallace, R. 1992. Partial constraint satisfaction. *Artificial Intelligence* 58:21–70.

Ginsberg, M., and McAllester, D. 1994. Gsat and dynamic backtracking. In *proc. 4th Int. Conf. on Principles of Knowledge Representation and Reasoning*.

Glover, F. 1990. Tabu search: a tutorial. *Interfaces* 20:74–94.

Gomes, C.; Selman, B.; and Kautz, H. 1998. Boosting combinatorial search through randomization. In *proc. Fifteenth National Conference on Artificial Intelligence*, 431–437. AAAI Press.

Havens, W., and Dilkina, B. 2004. A hybrid scheme for systematic local search. In *proc. Seventeenth Canadian Conference on Artificial Intelligence*.

Havens, W. 1997. Nogood caching for multiagent backtrack search. In *proc. AAAI-97 Constraints and Agents Workshop*.

Henz, M. 1999. Constraint based round robin tournament planning. In *proc. Sixteenth International Conference on Logic Programming*, 545–557. MIT Press.

Jussein, N., and Lhomme, O. 2002. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence* 139:21–45.

Lynce, I.; Baptista, L.; and Marques-Silva, J. P. 2001. Stochastic systematic search algorithms for satisfiability. In *proc. LICS Workshop on Theory and Applications of Satisfiability Testing (LICS-SAT)*.

Minton, S.; Johnston, M.; Phillips, A.; and Laird, P. 1990. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence* 58:161–205.

Morris, P. 1993. The breakout method for escaping from local minima. In *proc. AAAI-93*, 40–45.

Nemhauser, G., and Trick, M. 1998. Scheduling a major college basketball conference. *Operations Research* 46(1):1–8.

Prestwich, S. 2001. Local search and backtracking vs nonsystematic backtracking. In *proc. AAAI 2001 Fall Symposium on Using Uncertainty within Computation*.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A new method for solving hard satisfiability problems. In *proc. 10th National Conf. on Artificial Intelligence*, 440–446.

Stallman, R., and Sussman, G. 1977. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence* 9:135–196.

Yokoo, M. 1994. Weak commitment search for solving constraint satisfaction problems. In *proc. AAAI-94*, 313–318.