Shift-Scheduling with the "Projections First Strategy"

Johannes Gärtner
Abteilung für CSCW, Institut für Gestaltungs- und Wirkungsforschung,
Vienna Technical University
A-1040 Vienna, AUSTRIA
Phone: ++43-1-58 801 - 4419
Fax: ++43-1-504 24 78
E-mail: jgaertne@email.tuwien.ac.at

Silvia Miksch
Austrian Research Institute for Artificial Intelligence (ÖFAI),
Schottengasse 3, A-1010 Vienna,
AUSTRIA
Phone: ++43-1-53 53 281 - 0
Fax: ++43-1-532 06 52
E-mail: silvia@ai.univie.ac.at

Abstract

Shift-scheduling for employees is highly complex due to the size of each solution and the size of the solution space. We model this real-world problem as a CSP-problem and simplify this NP-hard problem dramatically with an algorithm called "projections first strategy".

The projections first strategy benefits from the interaction with the user (e.g., variable ordering) and the extensive usage of aggregated features of possible solutions (projections) to prune the search space and to find building blocks to build up solutions quickly. Our approach captures a human problem solving technique algorithmically, namely modularisation. To give an idea:

> *The merchant asked: How to lay my bricks to get 48 offices, an inviting entrance hall, ...*
>
> *The craftsman answered: First we have to make up our mind on the number of floors, then on the number of rooms. Several floors will be equal. .... Don't bother the bricks at the beginning.*

Projections make also recomputation easier when constraints change. Projections allow chronological backtracking to move through levels of abstraction and to reuse results.

Applying a realistic example we compare the complexity of our approach with back-tracking. In good and bad cases our approach reduces the complexity dramatically.

KEYWORDS: Applied CSP
Shift-scheduling
Search
AI applications

DECLARATION: This paper has not been already accepted by and is not currently under review for a journal or another conference. Nor will it be submitted for such during IJCAI´s review period.

1. Introduction

A very popular approach to scheduling problems is to formulate them as constraint satisfaction problems (CSP) because scheduling is complex and knowledge-intensive. But scheduling needs extensions of pure CSP to tackle search efficiently. Although the amount of reported work about CSP is increasing, there is a lack of mapping CSP technology onto real-world problems (Prosser, 1993). The developed algorithms tend to look at "toy" problems, like graph coloring problem, n-queens problem, confused-n-queens problem, zebra problem, puzzle-solving problem, and scene labeling. These algorithms make assumptions, like binary constraints, which are not always "natural" in real-world problems.

Shift-scheduling is a highly complex scheduling task due to the number of constraints, global constraints, the size of the solution space and the size of each solution. Furthermore it is dynamic due to emerging requirements (e.g., organizational) and bad or incomplete knowledge (e.g., time preferences of employees). We do not discuss the optimization of shift-schedules in principle here (compare e.g., Gärtner 1992; Nachreiner, et al. 1993). Designers are usually satisfied with sub-optimal solutions if they work (Nachreiner, et al. 1993) because of the huge effort of developing further admissible solutions.

Primarily two approaches to shift-scheduling exist. The first one focuses on issues of improved backtracking, multi-attribute optimization/operation research (e.g., see Adelman 1992). This approaches suffer of limitations in capturing domain knowledge and in exploiting that knowledge (Fox 1990; Prosser, et al. 1994). Fröschl (1993) stressed the importance of the human factor to be involved in the solving algorithm. The second approach is founded on user-centered decisions (ShiftPlanAssistent, Gärtner and Wahl 1994) providing a structuring tool that performs analyses and overviews but moves most decisions to the users.

In contrast to these approaches we modeled shift-scheduling as a CSP and extended the pure CSP with interventions of users and aggregations of variables to make the problem solving more efficient and parts of solutions reusable by chronological backtracking. According to Freeman-Benson, et al. (1990) we call the set of solutions that satisfy all the required constraints *admissible solutions*. There may exist many admissible solutions for a given CSP. Our central aim is to find admissible solutions quickly.

A CSP can be established in a general way as follows (Meseguer, 1989)

Let $\{X1,...Xn\}$ be a set of variables with values in a set of discrete and finite domains $\{D_1,...D_n\}$. Let $\{R_k\}$ be a set of constraints each of which shows the values mutually compatible for a variable subset. Thus $R_j \subseteq D_{i,1} x...x D_{i,j}$ denotes the compatible values among the variables $X_{i,1},...X_{i,j}$. The problem is to find an assignment of values to variables such that all the constraints are satisfied. Every different value assignment that satisfies all the constraints is called a solution.

The "projection first strategy" prunes the search process applying *mountain view* and constructs a solution by solving simplified CSPs that are used as *building blocks*.
Mountain view simplifies the original CSP using variable-classes, variable-projections, and variable ordering. Variable-classes use the feature of the problem that many variables with equal constraints and equal domains are used. This allows efficient pruning of domains (doing it once for each class) and efficient computing how many different instantiations of equal variables are needed and are admissible for an admissible solution. This prunes the search space. Variable ordering is done by the users in advance, capturing domain knowledge about basic structures of solutions.
Building blocks are solutions for subproblems (solution-projections) that build an admissible solution for the complete problem by concatenation (e.g., we try to find shift-schedules for two weeks that give a shift-schedule for the whole year by replication) or parallelisation (a number of workers have similar shift-schedules).
Mountain view and building blocks use features of the problem domain: First, a great number of variables with equal constraints and equal domains are used. Second, in admissible solutions a huge number of variables may have equal instantiations. Third, the constraints are cyclic and refer (partially) to global features of a solution.
Chronological backtracking uses the existing knowledge about building blocks and mountain view to prune recomputation. Subsolutions may be reused and the search space is already limited. Additionally, users handle constraint relaxation when no admissible solution is found, and decide whether recomputation does make sense (e.g., when constraints or domains have changed).

In chapter 2 we introduce preliminary notations of the shift-scheduling problem and domain specific as well as formal characteristics of constraints. In the following chapter we describe the "projections first strategy" in detail. Finally, we compare the complexity of our "projections first strategy" with simple backtracking. The "projections first strategy" is strongest where it is difficult to find an admissible solution for large, structured CSP problems.

2. Shift-schedules

In the following we characterize shift-schedules and introduce important notions and constraint. Furthermore we discuss dynamics and constraint relaxation.

2.1. Definition of shift-schedules and important notions

A shift-schedule is the regulation of employees' work hours. They are primarily defined by shifts (figure 1) and rosters (figure 2).

Figure 1: Shifts (black...duties)

Shifts define duties (e.g., morning shift from 6 a.m. to 2 p.m.) relative to a day. There may be different shifts for weekends, etc. (e.g., Knauth 1993) and complex structures (breaks, stand-by duty, etc.).

Employees may have similar shifts on different days and several employees may have the same shift on the same day.

DAY	1	2	3	4	5	6	7	8	9	10
Group 1	M	M	E	E	N	N				
Group 2			M	M	E	E	N	N		
Group 3					M	M	E	E	N	N
Group 4	N	N					M	M	E	E
Group 5	E	E	N	N					M	M

Figure 2: Example Roster (M ... Morning shift, E...Evening shift, N ... Night shift)

Rosters define which (group of) employee(s) has to work on which shift. The given example is a roster with a length of 50 days and 5 groups. (Group 1 works the roster from the 1st row to the 5th row. Group 2 starts in the 2nd row.)

The example shown in Figure 2 is rather simple. Rosters with a more complex shift-sequence, more groups (e.g., 12) and longer duration (e.g., 48 weeks) are broadly spread. A formal definition of schedules is given in the next chapter.

The basic time structure: We use minimal time units to transform the problem into a finite problem. The smallest time unit is 1 hour (abbreviated as 1/24). The most common ones are one day (24 hours), one week (7 days = 168 hours) and maximal cycle (350 days). Smaller time units can be used but they increase computation costs and are rarely needed in practice. The domain of the time variable is an ordered set of units. Applying an enumeration function enables the definition of enumerated time units, like the first day, the first Friday in a month.

Shift-cycle: The length of a repeating pattern of shifts and employees within a schedule (e.g., 50 days in the example in Figure 2). Shift-group: A set of employees that have the same shifts on every day within such a repeating pattern.

Number-of-shift-workers: Number of workers within the same shift on the same day within a basic time structure. E.g., on each Monday 18 workers work on the morning shift. Number-of-workers-on-duty: Number of workers on duty given a particular day and a particular hour. Number-of-shifts: Number of shifts an employee has in a particular week. Length-of-shift: The length (in hours) of the duty for an employee on a particular day. If the employee has no duty it is 0. Operating-hours: The operating hours of SCHEDULE, i.e., the time frame in which workers are on duty.

2.2. Constraints to shift-schedules

Several constraints are applied:

Temporal constraints: A number of different approaches for temporal representation and reasoning exist (e.g., Allen, 1984, 1991; McDermott, 1982; Freksa, 1992). We use temporal intervals, e.g., a starting point of a duty and it's duration instead of begin

and end points of intervals, like (Allen, 1984). Temporal constraints are e.g., *No overlapping work times of one employee*.

Legal constraints: Depending on national laws, collective and company agreements a number of constraints are applied. E.g.,

Average work hours of employees per week have to be less then 40 hours. This average has to be reached within 26 weeks. Maximum work time per 24 hours is limited to 9 hours.

Company based constraints: E.g.,

140 hours operating time per week . Not less then 3 employees and not more then 18 employees at a time. No work on Thursday afternoon due to repairs.

Health based constraints: A huge number of ergonomic criteria has been developed (see e.g., Knauth 1993, Schönfelder 1992), e.g.,

No night work in a row for more then three nights. No night work if other work time is possible. No start-time between 11 p.m. and 5 a.m.

Employee based constraints: E.g.,

No night work for Mr. X.. Same number of work shifts for all employees.

2.3. Domains of shift-schedules

The scope of our CSP is restricted to finite and discrete domains for each variable. In the following we describe the properties of the used domains.

DOMAIN DAYS: In principle, the domain of days for a shift-schedule could be infinite. For practical reasons (technological and economical changes) shift-schedules are limited to a year or less (e.g., 50 weeks) and "reused" if possible. By this: `E.g., Domain (days)={ 1,...350}`

DOMAIN EMPLOYEES: The same argument holds true for employees. `E.g., Domain (employees) = { 1,...200}`

DOMAIN STARTING_TIME: This domain can be made finite by the introduction of a minimal time unit (see above). We use the representation of a 24 hours day.

`E.g., Domain (starting_time) = { 1,...24}`

DOMAIN DURATION: Also the domain of duration has to be made finite by a minimal time unit. In practice duration is limited to some degree (e.g., legal constraints). `E.g., Domain (duration) = { 6,7,8,9,10}`

2.4. Dynamic constraints and constraint relaxation

Shift-schedules affect the organization and the employees in many ways (e.g., cooperation between departments, customers, health and life-styles of employees). New constraints and changes of domains emerge through changes in the environment, experiences with shift-schedules etc.. If no admissible schedule can be found constraints must be relaxed.

Constraint relaxation can be handled by the users determining preferences of constraints in advance (e.g., Mr. X prefers morning shift over night shifts), by direct inventions of the users specifying which constraint should be relaxed first. Freemann-

Benson, et al. (1992) propose an incremental constraint satisfaction system for interactive applications where the constraint hierarchy evolves gradually.
These approaches are questionable in the domain of shift-scheduling. Case studies (e.g., Wahl 1995) and organizational-theory (e.g., Perrow 1986) stress that constraint relaxation and constraint hierarchy is solved in bargaining learning processes and can't be foreseen. Therefore constraint relaxation has to be handled by the users. The computer system should only support recomputation and book-keeping functions as used in REDUX' (Petrie, et al. 1994).

2.5. Characteristics of constraints

Rossi et al. (1990) have proven that binary and non-binary CSPs are equivalent when they are mutually reducible, in the sense that they contain the "same information". In their scheme, both constraint and variable redundancy are allowed in CSPs belonging to the same equivalence class.
The "nature" of some constraints applying to shift-schedules are non-binary in the sense that they address global features of a solution (e.g., average work time of employees). This non-binary CSP could be to transformed into a binary CSP but on high costs. We choose a different approach which tries to use these global features to find a solution more efficiently. This approach is described in chapter 3.

3. The "projections first strategy"

3.1. Overview

A large number of problems can be viewed as special cases of CSP with a large number of different approaches to improve efficiency and consistency (e.g., "Generate & Test", simple backtracking to intelligent - selective and dependency-directed - algorithms like constraint propagation, lookahead, lookback algorithms; compare Meseguer, 1989, Kumar, 1992).
Backtracking algorithms are a prominent processing technique in AI, in particular in CSP. Standard backtracking attempts to assign values to variables so that all constraints are satisfied. Given a variable order it starts with the first variable assigning values as long as each assigned value is consistent with the values assigned to preceding variables. At dead-end situation backtracking takes place. The two major drawbacks are thrashing and redundant assignment work (Kumar, 1992).
Lookahead algorithms are based on the idea that each step towards a solution should have some evidence that the following path does not lead to a dead-end situation. Moreover, they eliminate values of future variables because they will never appear in any solution. *Forward checking* and *full lookahead* belong to this group. *Full lookahead* guarantees that every future variable has at least one compatible value with the current value assignment to past and current variables and every future variable has at least one compatible value with any other future variable. *Forward checking* is equivalent to full lookahead but without any consistency test among future variables (Meseguer, 1989).

Basically two methods are used to improve efficiency and consistency: first, to modify the search space to make the search easier; second, to use heuristics to guide this search. We use *mountain view* to modify the search space and *building blocks* to guide the search process.

Mountain view works extensively with variable-classes for variables with equal constraints and equal domains. It is a feature of this CSP (and of others) that the number of similar variables is very high. Firstly, variable-classes are used to prune the domain of each variable (doing it once for each class). Secondly, to compute how many *different* instantiations of a variable-class are needed and admissible for a solution. E.g., we compute admissible numbers of employees and instantiate with suitable sets. Thirdly, users order this smaller number of variable-classes capturing domain knowledge about the structure of solutions (e.g., shift-schedules are ordered by days). We called the strategy *mountain view* (the name comes from the Austrian experience that one does not see all valleys, but one knows main directions of possible movements).

Additionally, we apply a decomposition strategy (*building blocks*) by looking for subproblems that allow the construction of a solution by pure concatenation, by concatenation with minor adaptations or by parallelisation (e.g., we try to find shift-schedules for two weeks that give a schedule for the whole year by replication). This strategy guides our search process. Figure 3 shows the basic idea of the *building block* approach (= solution-projection).

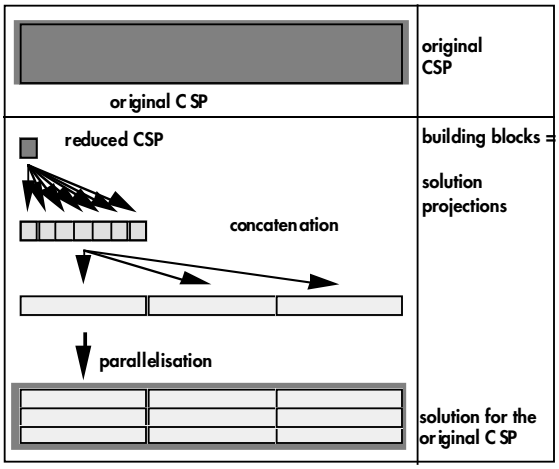


Figure 3: Building blocks' approach

3.2 The problem definition in CSP manner and preparation

In this chapter we introduce a way to state the shift-scheduling as a CSP and prepare the algorithm exemplified on the problem of shift-scheduling.

```
STEP 0:   (a) variables, variable-classes and domains
          (b) constraints
          (c) pruning of variable-class domains
          (d) variable ordering & blocks
```

STEP 0 (a): VARIABLES, VARIABLE-CLASSES and DOMAINS

Corresponding to the definition of a CSP given in chapter 1 the set of variables $\{X_1,...X_n\}$ with a set of discrete and finite domains $\{D_1, ...D_n\}$ have to be defined. Variables of a CSP can be viewed as instances of variable-classes. In many cases this view is quite natural (e.g., in the n-queens problem each $\texttt{QUEEN}_i$ is an instance of class `QUEEN`). This view of variables as instances of classes allows an easier formulation of

a problem in a CSP manner whenever a large number of similar variables are used. All instances of such a variable-class have to have the same domain, all constraints have to hold for them.

EXAMPLE: SHIFT-SCHEDULE: Find an admissible instantiation for

SCHEDULE := { $DUTY_1$, $DUTY_2$, ...$DUTY_m$ }

DUTY :=(DAY, EMPLOYEE, SHIFT)

SHIFT:=(START_TIME, DURATION) where VALUE(DURATION) ≠ 0

The following similar notation is sometimes used to increase readability.

SCHEDULE = { (DAY_i, $EMPLOYEE_i$, $START\_TIME_i$, $DURATION_i$) }

DAY... is a variable-class of days (e.g., 3rd day of the schedule).

EMPLOYEE... is a variable-class of specific employees (e.g., McMac).

START_TIME. is a variable-class of the starting time of a shift (e.g., 6 a.m.)

DURATION... is a variable-class of the temporal duration of a shift (e.g., 8h)

The following table gives an overview about the variables, their domains, and variable-classes used in the shift-scheduling problem.

VARIABLE	**DOMAIN (example)**	**Variable-class**
DAY_i	D_DAY ={ day_1,...day_{350} }	CLASS_DAY
$EMPLOYEE_i$	D_EMPLOYEE ={$employee_1$,...$employee_{200}$ }	CLASS_EMPLOYEE
$SHIFT_i$	D_SHIFT ={h_1,...h_{24} }x{1h,...24h}	CLASS_SHIFT

Table 1: Overview of used variables, domains, and corresponding variable -classes

EXAMPLE

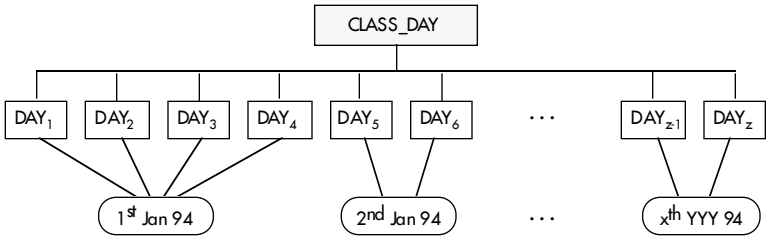


Figure 4: Variable-classes, variables and domains illustrated with "day"

STEP 0 (b): CONSTRAINTS

Corresponding to the definition of a CSP a set of constraints $\{R_k\}$ has to be defined. Theoretically $\{R_k\}$ is given and the only problem is to find an admissible assignment. In practice these sets have to be edited or computed most of the time. In our example and in other practical problems this would be very expensive. Therefore we reformulate the problem by formulating constraints as decideable expressions. Given finite domains and decideable expressions the set $\{R_k\}$ could be computed. Further we spare work by computing elements of $\{R_k\}$ only when necessary.

Examples for functions in the field of shift-scheduling where given in chapter 2.2. (e.g., Length_of_shift). We show how to formulate such constraints.

EXAMPLE 1: Max. work hours per day for each employee ≤ 9h.

can be formulated rather easily in the following ways

(a) " instances $DUTY_i$ of CLASS_DUTY: Length_of_shift(DUTYi) ≤ 9h

(b) ∀ instances DAY_i of CLASS_DAY
instances $EMPLOYEE_i$ of CLASS_EMPLOYEE
Length_of_shift(DAY_i,$EMPLOYEE_i$) ≤ 9h

Furthermore we omit "instances DAY_i of CLASS_DAY" and just write DAY_i .

EXAMPLE 2: Every 2nd week, no workers are needed from Friday 2pm to Saturday 6am due to repairs can be formulated as

∀ DAY_i ,VALUE (DAY_i) = $Friday_{2nd}$, $hour_t$ ε {2pm...12pm} AND

∀ DAY_i ,VALUE (DAY_i) = $Saturday_{2nd}$, $hour_t$ ε {1am...5am}

Number_of_workers_on_duty(DAY_i,$hour_t$) = 0

Many constraints refer to values of the domain of a variable (see example above).

Instead of writing: ∀ DAY_i : VALUE (DAY_i) = day_j ε D_DAYS

we just write: ∀ day_j ...

CONSTRAINTS IN THE SHIFT-SCHEDULING EXAMPLE (informal notation):

(1) Each hour number of employees on duty is less then 30

(2) Max. work hours per day for each employee with a duty ≤ 9h

(3) Min. work hours per day for each employee with a duty ≥ 7h

(4) 40 work hours for each employee who is hired per week

(5) Max. 5 duties a week for each employee

(6) Mr. Meier does not work on Wednesdays

(7) Every second week, no workers are needed between Friday 2pm and Saturday 6am due to repairs.

(8) 216.000 total work hours per year

CONSTRAINTS IN THE SHIFT-SCHEDULING EXAMPLE (formal notation):

(1) ∀ day_i ∀ $hour_t$ Number_of_workers_on_duty (day_i,$hour_t$) < 30

(2) ∀ day_i ∀ $employee_k$ Length_of_shift (day_i,$employee_k$) ≤ 9

(3) ∀ day_i ∀ $employee_k$ On_duty (day_i, $employee_k$) :⇒
Length_of_shift(day_i,$employee_k$) ≥ 7

(4) ∀ $employee_k$ ∀ $week_j$ ∑ Length_of_shift (day_i,$employee_k$) = 40
(day_i ∈ $week_j$ & On_duty (day_i, $employee_k$)
OR ¬∃ day_i : On_duty (day_i, $employee_k$))

(5) ∀ $week_j$ ∀ $employee_k$ Number_of_shifts ($week_j$,$employee_k$) ≤ 5

(6) ∀ $week_j$ On_duty (Wednesday , Mr.Meier) is FALSE

(7) ∀ $14days_j$ ∑ Number_of_workers_on_duty (day_i,$hour_t$) = 0
(day_i = 2ndFriday & $hours_t$ ∈ {2pm...12pm}) OR
(day_i = 2ndSaturday & $hours_t$ ∈ {1am...5am})

(8) $\sum_{\{dayi\},\{hourt\}}$ Number_of_workers_on_duty (day_i,$hour_t$) = 216.000

STEP 0 (c): PRUNING OF VARIABLE CLASS DOMAINS

We prune the domains which could be used for the variables of each variable class by applying the constraints which influence them directly (node-checking).

We use the notation of DP_** for the pruned domain D_**

$D\_DAY = \{day_1, \ldots day_{350}\}$	DP_DAY = D_DAY
$D\_EMPLOYEE = \{employee_1, \ldots employee_{200}\}$	DP_EMPLOYEE = D_EMPLOYEE
$D\_SHIFT = \{h_1, \ldots h_{24}\} \times \{1h, \ldots 24h\}$	$DP\_SHIFT = \{h_1, \ldots h_{24}\} \times \{7h, \ldots 9h\}$ by (2),(3)

STEP 0 (d): VARIABLE ORDERING & BLOCKS

Two orders have to be defined by the user in advance. Firstly, the variable ordering which is always possible. Secondly, the order of block sizes (i.e., should the algorithm work with concatenation or with parallelisation) which has weak preconditions. It can be done if a semi-ring together with multiplication by elements of **N** is defined. This precondition holds in many practical cases (including: time, length, temperature, weight, might of sets). Basically, it is necessary to be able to compute LCM's (Least Common Multiples). E.g.,

1. VARIABLE: DAY
   ORDER CONCATENATION: less $\prec$ many days
   (i.e., Days are the most structuring element of shift-schedules; short shift-cycles are preferred because of their simplicity; furthermore many constraints refer to temporal patterns.)
2. VARIABLE: EMPLOYEE
   ORDER PARALLELISATION: many similar $\prec$ a few similar employees
   (i.e., Employees are an important structuring element of shift-schedules and large groups of employees that have the same schedule are preferred.)
3. VARIABLE: SHIFT
   ORDER CONCATENATION: less $\prec$ many shifts
   (i.e., shifts are not very important to structure a schedules, still less shifts make scheduling easier.)

3.3 Finding a solution for the smallest block-size

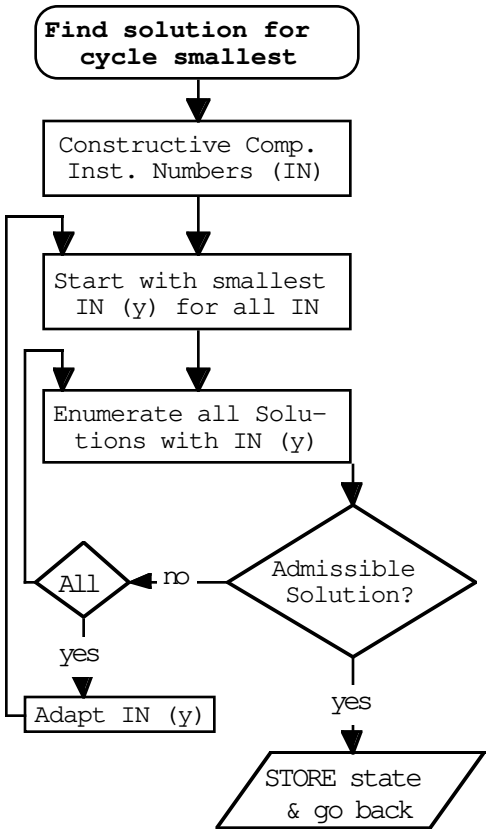


Figure 5: Smallest Block

First, an admissible solution for the smallest block is computed (figure 5).

The smallest block sizes are computed straight forward from the constraints and the domains (constraints refer mostly to temporal cycles; i.e., 1 hour = min. time unit; 1 group of employees with 200 members = max. of domain; one shift = min.).

Each solution of a CSP is based on instantiations. The number of instantiated variables of each class may be very large (e.g., in the example roughly 70.000). But many variables may have the same value. We call the number of different values of instantiations of a variable-class *instantiation-number* (IN) (e.g., 3 different shifts). IN are very useful to prune the search space. E.g.,

Consider the organization of a conference with m sessions in n tracks. Let k possible chairpersons be available. We can say that each solution has at least n different and a maximum of n x m or k chairpersons.

Constructive Computation of Instantiation Numbers (IN): The original set of instantiation numbers is indirectly described by the size of the pruned domains of each variable-class (e.g., IN(DAYS)={0,...|{1,...350}|}={0...350}). Some constraints allow rather straight forward reduction of this sets with respect to different block sizes used in the constraints. For each variable-class:

(a) compute block-cycles used by the quantifiers of constraints
(b) compute the closed set of LCM's for such cycles
(c) prune the sets of instantiation numbers (IN)

(a),(b) CYCLES (DAYS) = { 1/24, 1, 7, 14, 350}
CYCLES (EMPLOYEE) = { 1 }

We use two ways to prune the IN-sets constructively. Either a constraints refers directly to the number of possible instantiations (e.g., constraint (1)) or applying rule of proportion with the involved quantifiers (e.g., constraints (4) and (8)).

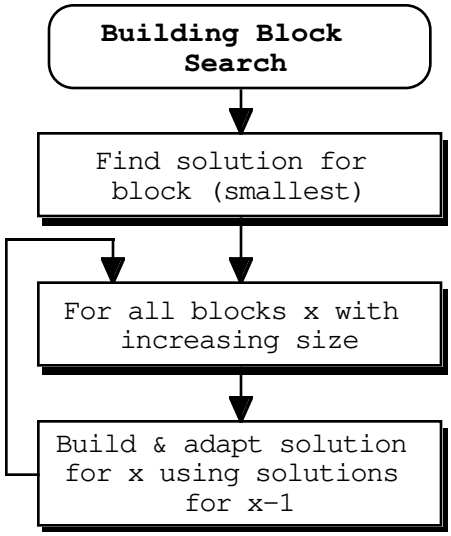
(c1) IN(EMPLOYEES, 1/24) = {1,...29} by (1)
(c2) IN(EMPLOYEES, 350) = {108} by (4), (8)
(c3) IN(DAYS, 350) = {311...350} by (1), (8)

A more powerful reduction of IN-sets were possible but costly. We do it later hand in hand with the computation of solutions.

Start with smallest IN for all IN: The algorithm starts with the smallest element of each IN-set. The enumeration of sets of IN is done by backtracking with respect to

the variable order. Hand in hand with the enumeration of solutions for each IN-element the IN sets are further pruned (Adapt IN(y)), e.g., if no admissible solution is found with "2" different shifts than "2" is eliminated from the corresponding IN-set. This is done for all IN-sets for variable-classes and for all their combinations (e.g., cycle length 14 days, 4 groups of employees, 3 classes of shifts). If an admissible solution is found the state of computation is stored for latter continuation and the result is returned. If no admissible solution is found for any element of the IN-set the algorithm stops.

3.4 Computing larger solutions



Given (an) admissible solution(s) for a smaller block larger blocks are constructed in the way shown in figure 6.
The computation of a solution for the smallest block was discussed above.

Figure 6: Building Block Search

Increase of size of blocks is primarily defined over the order of LCM's computed in the step Constructive Computation of Instantiation Numbers(IN) (see above) (e.g., DAYS: 1/24, 1, 7, 14, 350). If there does not exist this ordering then multiples of the basic unit of block size are used.
Solutions for larger blocks are build up by concatenation or by parallelisation of smaller blocks (as defined by the user in step 0(d)).If a larger block is admissible the algorithm proceeds to the next block size. If not than the solution has to be adapted. Admissible smaller blocks are combinated systematically. If no admissible larger block can be found the algorithm stops.

3.5 Constraint Relaxation & dynamic constraints

The embedding of constraint relaxation into an organizational setting and our focus point on recomputation and bookkeeping was discussed in chapter 2. The user has to decide how far recomputation makes sense.
If new constraints evolve recomputation is supported by the already computed hierarchy of smaller admissible blocks and the sets of instantiation numbers. In recomputation the algorithm steps back to smaller blocks until these blocks(!) are admissible (*chronological backtracking).* If constraints were relaxed the user has to decide from which level on recomputation should be done. Changes in the block hierarchy (e.g.,

caused by a constraint like: every 4 weeks...) do not change the process in principle. Bookkeeping for all results (admissible or not) is too expensive. Bookkeeping for instantiation numbers (i.e. which constraints let to the elimination of an element of an IN-set) reduces recomputation costs drastically.

4. Complexity

Backtracking and the "Projections First Strategy" are compared using a simplified version of the example given above (200 employees, 350 days, 1 hour minimal time unit, 36 constraints). As costs are case dependent we discuss two cases. In the best case example we show a detailed analysis of costs. In bad cases we consider the most dominant computation step.

4.1. No constraints lead to any kind of backtracking

We assume a meta-heuristic that instantiates all variables such that no backtracking applies. Only checks are necessary. The simplified complexity analysis (figure 8) is read: "For the work time of employees there refer 6 constraints per day. Results have to be checked for 350 days ...". Costs = (number of variables) * (number of constraints to check) * (number of results to check).

Constraints refer to		#	Results to check	#	Total	Variables involved	#	Total	COSTS =
Working time of employees	day	6	for each day for each employee	350 200	70000	shifts	1	1	**420.000**
	week	6	for each week for each employee	50 200	10000	shifts	7	7	**420.000**
	year	6	for each employee	200	200	shifts	350	350	**420.000**
Operating hours	day	6	for each day	350	350	employees	200	200	**420.000**
	week	6	for each week	50	50	employees days	200 7	1400	**420.000**
	year	6	for the year	1	1	employees days	200 350	70000	**420.000**

Figure 7: Simplified complexity analysis of simple backtracking.

Constr.				Domainsize		COSTS
Node Check		36	shifts	576		20.736
		36	days	350		12.600
		36	employees	200		7.200
Comp. IN		36	days	350		12.600
		36	employees	200		7.200
		36	shifts	576		20.736
			Variables	**#**	**Total**	
1	.solutio	12	days	1	200	2.400
			employees	200		
			shifts	1		
2	.solutio	12	days	7	1400	16.800
			employees	200		
			shifts	1		
3	.solutio	12	days	350	70000	840.000
			employees	200		
			shifts	1		
					SUM	**940.272**

Figure 8: Best case "Projections first strategy"

Figure 8 shows the costs for the projections first strategy in the same example. In the first part of the figure the costs for preparation are computed. In the second part the costs for a straight forward solution are stated.

Note that only one result has to be checked at each step, which reduces costs significantly.

If no constraints lead to backtracking costs are reduced by 60% (from 2.520.000 to 940.000).

4.2. Bad cases

The analysis of bad cases (i.e., the whole search space has to be scanned) is done by a computation of boundaries instead of a detailed analysis. The number of possible combinations of instantiations of variables is a boundary for backtracking (if the computational costs for constraints are constant).The "projections first strategy" is bounded by 2 times the dominant computational step (the worst behavior of building blocks is: max + max/2 + max/4 + max/8 + ...) and small costs caused by node-checking and preparation.

(1) The worst case for simple backtracking is bounded by $....(24 * 24)^{200 * 350}$

(2) If only variable-projection is applied, the projections first strategy is bounded by $.....2 * (3 * 24)^{108 * 350} * \F(200!;108!*(200-108)!)$

(3) If variable-projection and building blocks with 14 days, 4 groups, 3 shifts are applied (n = number of admissible solutions for the 14days block), the boundary is $.....2 * 3^{4 * 14} * \F(200!;27!^4*(200-4*27)!) + (27 * 14) * (n)^{25}$

(4) If all employees are equal the boundary further decreases to $..... 2 * 3^{4 * 14} + (27 * 14) * (n)^{25}$

The differences are impressive, but still we hopefully don't trigger bad cases. If the solution space for building blocks is dense AND has to be fully scanned AND all combinations of building blocks have to be checked than the boundary is near the one of backtracking. If the solution space is sparse OR only few admissible solutions for smaller building blocks have to be found OR it is easy to combine smaller building blocks than the boundary decreases extremely. Practical experience shows that either the solution space for shift scheduling is extremely sparse OR it is trivial to combine smaller building blocks. Both cases favor the *Projections-first-strategy*.

5. Conclusion

We formulated the *real-world* problem of shift-scheduling as a CSP and presented our solution algorithm "projections first strategy". The projections first strategy benefits from the interaction with the user (e.g., variable ordering) and the extensive usage of aggregated features of possible solutions (projections) to prune the search space (*mountain view*), and to find *building blocks* that build up solutions quickly.

Our "projections first strategy" algorithm delivers good results for problems were a problem decomposition in small sub-problems is admissible (= solution-projection) which are later used as building blocks. Additionally, it benefits of large numbers of similar variables with equal domains and equal constraints (variable-projection). Our approach develops its strength when a high number of similar variables involved as well as when constraints referring to non-local features of a solution. Such problems can be found in other areas too, e.g., in construction, architecture, processor-design.

The comparison of the complexity of the "projections first strategy" with simple backtracking based on a realistic example showed that in good and bad cases we could reduce the complexity dramatically. Topics of future research are the development and application more efficient forms of decomposition and instantiation. Additionally, we will improve the handling of dynamic constraints and constraint relaxation.

6. References

Adelman L.: Evaluating Decision Support and Expert Systems, *Wiley*, Chichester, UK, 1992.

Allen J.F.: Time and Time Again: The Many Ways to Represent Time, *International Journal of Intelligent Systems*, Vol. 6, pp.341-55, 1991.

Allen J.F.: Towards a General Theory of Action and Time, *Artificial Intelligence*, 23(2), pp.123-154, 1984.

Fox, M.S.: AI and Expert Systems Myths, Legends and Facts, *IEEE Expert,* 5(1), pp. 13-25, 1990 cited in (Prosser, P., Buchanan, I. 1994)

Freeman-Benson B., Maloney J., Borning A.: An Incremental Constraint Solver, *Communications of the ACM*, 33(1), pp.54-63, 1990.

Freksa C.: Temporal Reasoning Based on Semi-Intervals, *Artificial Intelligence*, 54(1-2), pp.199-227, 1992.

Fröschl K.A.: Two Paradigms of Combinatorial Production Scheduling Operation Research and Artificial Intelligence, in Dorn J., Fröschl K.A.(eds.), *Scheduling of Production Processes*, Ellis Horwood, New York, pp.1-21, 1993.

Gärtner J.: CATS-Computer Aided Time Scheduling - Ein Modell für die computer-unterstützte (Schicht-) Arbeitszeitplanung; *Dissertation*, TU-Wien, 1992.

Gärtner J., Wahl, S: Working Time Lab - Participative Organizational Planning of Shift Schedules, *Participative Design Conference*, pp. 121-122, 1994.

Knauth P.: The Design of Shift Systems, *Ergonomics*, 36(1), pp. 77-83, 1993.

Kumar V.: Algorithms for Constraint-Satisfaction Problems: A Survey, *AI Magazine*, pp.32-44, 1992.

McDermott D.: A Temporal Logic for Reasoning About Processes and Plans, *Cognitive Science*, 6(2), pp.101-156, 1982.

Meseguer P.: Constraint Satisfaction Problems: An Overview, *AI Communications*, 2(1), pp.3-17, 1989.

Nachreiner F., Ling Q., Grzech H., Hedden I.: Computer Aided Design of Shift Schedules, *Ergonomics*, 1993.

Petrie C., Cutkosky M., Park H.: Design Space Navigation as a Collaborative Aid, in Gero J., Sudweeks F. (eds.), *Artificial Intelligence in Design ´94*, Kluwer Academic Publishers, Netherlands, pp. 611-623, 1994.

Perrow C.: Complex Organizations - a Critical Essay, Random House, New-York 1986.

Prosser P.: Scheduling as a Constraint Satisfaction Problem: Theory and Practice, in Dorn J., Fröschl K.A.(eds.), *Scheduling of Production Processes*, Ellis Horwood, New York, pp.22-30, 1993.

Prosser P., Buchanan I.: Intelligent Scheduling: Past, Present and Future, *Engineering Intelligent Systems*, Summer, 1994.

Rossi F., Petrie C., Dhar V.: On the Equivalence of Constraint Satisfaction Problems, in Aiello L.(ed.), *Proceedings of the 9th European Conference on Artificial Intelligence (ECAI-90)*, Pitman, London, pp.550-557, 1990.

Schönfelder E.: Entwicklung eines Verfahrens zur Bewertung von Schichtsystemen nach arbeitswissenschaftlichen Kriterien, *Peter Lang Verlag*, 1992.

Wahl, S.: Computerunterstützte Schichtarbeitszeitplanung, to appear, 1995.