# Synthesis of Planning and Scheduling Software

**Douglas R. Smith, Eduardo A. Parra, and Stephen J. Westfold**
Kestrel Institute
3260 Hillview Avenue
Palo Alto, California 94304 USA
email: smith@kestrel.edu

**Abstract**

This report describes our research on transportation planning and scheduling supported by the ARPA/Rome Lab Planning Initiative (ARPI). The main goal of this project was to develop generic tools to support the construction of flexible, high-performance planning and scheduling software. Our technical approach is based on program transformation technology which allows machine-supported development of software from requirement specifications. The development process produces code that is correct by construction and which can be highly efficient.

We have used KIDS (Kestrel Interactive Development System) to derive extremely fast and accurate transportation schedulers from formal specifications. As test data we use strategic transportation plans which are generated by U.S. government planners. A typical problem, with 10,000 movement requirements, takes the derived scheduler 1 – 3 minutes to solve, compared with 2.5 hours for a deployed feasibility estimator (JFAST) and 36 hours for deployed schedulers (FLOGEN, ADANS). The computed schedules use relatively few resources and satisfy all specified constraints. The speed of this scheduler is due to the synthesis of strong constraint checking and constraint propagation code.

## Introduction

This paper describes our exploration of the transformational development of transportation schedulers.[1] Our approach involves several stages. The first step is to develop a formal model of the transportation scheduling domain, called a *domain theory*. Second, the constraints, objectives, and preferences of a particular scheduling problem are stated within a domain theory as a *problem specification*. Finally, an executable scheduler is produced semi-automatically by applying a sequence of *transformations* to the problem specification. The transformations embody programming knowledge about algorithms, data structures, program optimization techniques, etc. The result of the transformation process is executable code that is correct by construction. Furthermore, the resulting code can be extremely efficient.

Transportation scheduling tools currently used by the U.S. government are based on models of the transportation domain that few people understand (Schank 1991). Consequently, users often do not trust that the scheduling results reflect their particular needs. Our approach tries to address this issue by making the domain model and scheduling problem explicit and clear. If a scheduling situation arises which is not treated by existing scheduling tools, the user can specify the problem and generate an situation-specific scheduler.

One of the benefits of a transformational approach to scheduling is the synthesis of specialized constraint management code. Previous systems for performing scheduling in AI (e.g. (Fox, Sadeh, & Baykan 1989; Smith 1989)) and Operations Research (Applegate & Cook 1991; Luenberger 1989) use constraint representations and operations that are geared for a broad class of problems, such as constraint satisfaction problems or linear programs. In contrast, transformational techniques can derive specialized representations for constraints and related data, and also derive efficient specialized code for constraint operations such as constraint propagation and constraint checking.

In Sections 2 and 3 we describe the KIDS program synthesis system and the programming knowledge that it uses to synthesize scheduling algorithms. In Section 4 we describe the application of KIDS to the synthesis of three different schedulers. The first, called KTS, is a strategic transportation scheduler that runs as much as several orders of magnitude faster than currently deployed (and manually written) schedulers. The second, called ITAS, is a laptop-based in-theater scheduler that has been delivered to PACAF at Hickham AFB

[1] This research was supported by ARPA/Rome Laboratories under Contracts F30602-91-C-0043 and F30602-95-C-0247, and also by the Office of Naval Research under Contract N00014-93-C-0056.

in Hawaii and is regarded as being ready for operation during contingencies. The third is a scheduler for power plant maintenance activities.

## KIDS model of program development

KIDS is a program transformation system – one applies a sequence of consistency-preserving transformations to an initial specification and achieves a correct and hopefully efficient program (Smith September 1990). The system emphasizes the application of complex high-level transformations that perform significant and meaningful actions. From the user's point of view the system allows the user to make high-level design decisions like, "design a divide-and-conquer algorithm for that specification" or "simplify that expression in context". We hope that decisions at this level will be both intuitive to the user and be high-level enough that useful programs can be derived within a reasonable number of steps.

The user typically goes through the following steps in using KIDS for program development.

1. *Develop a domain theory* – An application domain is modeled by a domain theory (a collection of types, operations, laws, and inference rules). The domain theory specifies the concepts, operations, and relationships that characterize the application and supports reasoning about the domain via a deductive inference system. Our experience has been that distributive and monotonicity laws provide most of the laws that are needed to support design and optimization of code. KIDS has a theory development component that supports the automated derivation of various kinds of laws.

2. *Create a specification* – The user enters a problem specification stated in terms of the underlying domain theory.

3. *Apply a design tactic* – The user selects an algorithm design tactic from a menu and applies it to a specification. Currently KIDS has tactics for simple problem reduction (reducing a specification to a library routine), divide-and-conquer, global search (binary search, backtrack, branch-and-bound), constraint propagation, problem reduction generators (dynamic programming, general branch-and-bound, and game-tree search algorithms), and local search (hillclimbing algorithms).

4. *Apply optimizations* – The KIDS system allows the application of optimization techniques such as expression simplification, partial evaluation, finite differencing, case analysis, and other transformations. The user selects an optimization method from a menu and applies it by pointing at a program expression. Each of the optimization methods are fully automatic and, with the exception of simplification (which is arbitrarily hard), take only a few seconds.

5. *Apply data type refinements* – The user can select implementations for the high-level data types in the program. Data type refinement rules carry out the details of constructing the implementation.

6. *Compile* – The resulting code is compiled to executable form. In a sense, KIDS can be regarded as a front-end to a conventional compiler.

Actually, the user is free to apply any subset of the KIDS operations in any order – the above sequence is typical of our experiments in algorithm design.

## Specifying Transportation Scheduling Problems

The essential notion of transportation scheduling is that some movement requirements (e.g. bulk cargo, passengers) are assigned to transportation assets (e.g. planes, ships, trucks) over certain time intervals. Various constraints on the assignments must be satisfied and certain measures of the cost or "goodness" of the assignment may need to be optimized. A domain theory for transportation scheduling defines the basic concepts of transportation scheduling and the laws for reasoning about the concepts.

The U.S. Transportation Command and the component service commands use a relational database scheme called a TPFDD (Time-Phased Force and Deployment Data) for specifying the movement requirements of an operation, such as Desert Storm or the Somalia relief effort. A typical air movement requirement includes the following information:

| | | |
|---|---|---|
| $POE : port$ | $\mapsto$ | $UHHZ$ |
| $POD : port$ | $\mapsto$ | $VRJT$ |
| $movement\text{–}type : symbol$ | $\mapsto$ | $BULK$ |
| $quantity : Short\text{–}Tons$ | $\mapsto$ | 2 |
| $available\text{–}to\text{–}load\text{–}date : time$ | $\mapsto$ | 0 |
| $earliest\text{–}arrival\text{–}date : time$ | $\mapsto$ | 0 |
| $latest\text{–}arrival\text{–}date : time$ | $\mapsto$ | 86400 |
| $distance : nautical\text{–}miles$ | $\mapsto$ | 5340 |
| $mode : symbol$ | $\mapsto$ | $AIR$ |

Here, the Port of Embarkation (POE) and Port of Debarkation (POD) are the origin and destination of the cargo, respectively. UHHZ and VRJT are geographical codes (GEOLOCs) for Robins AF Base, Georgia and Sigonella Airport, Italy, respectively. Movement requirements are classified into a few gross types including BULK (which fits on a standard 463L pallet), OVRsize (which doesn't fit on 463L pallet but fits on most standard cargo transports), OUTsize (which doesn't fit on 463L pallet and requires a heavy lift transport such as a C-5 or C-17), PAX (passengers), and POL (petroleum). Resources are characterized by their capacities (both passenger (PAX) and cargo capacities for each movement type), travel rate in knots, initial port, available date, maintenance profile, etc.

As an example, one TPFDD specifies the evacuation of non-combatants from a Pacific island nation. The TPFDD for this NEO (Non-combatant Evacuation Operation) has 33,291 records (movement requirements for force units, non-unit related cargo and non-unit related passengers) which generates about 12,500 air and 6,000 sea movement requirements to be processed. The scenario includes 103 air POEs (airports), 47 air PODs, 48 sea POEs (seaports), and 29 sea PODs. There are air movement requirements for 1,445,511 STONs (BULK, OVERsize, and OUTsize cargo) and 737,492 passengers, and sea movement requirements for 4,902,129 MTONs (Measurement TONs – a unit of volume, not weight) and 240,090 hundreds of barrels of petroleum products. Available air resources include KC10s, C-141s, C-5s, B-747, and sea resources include tankers (small, medium, and large), RO-ROs, LASHs, sea barges, container ships, and breakbulks.

We list below twelve constraints that partially characterize a feasible schedule for this problem:

1. *Consistent POE and POD* – The POE and POD of each movement requirement on a given trip of a resource must be the same.
2. *Consistent Resource Class* – Each resource can handle only some movement types. For example, a C-141 can handle bulk and oversize movements, but not outsize movements.
3. *Consistent PAX and Cargo Capacity* – The capacity of each resource cannot be exceeded.
4. *Consistent Initial Time* – The start time of the first trip of a transportation asset must not precede its initial available date, taking into account any time needed to position the resource in the appropriate POE.
5. *Consistent Release Time* – The start time of a trip must not precede the available to load dates (ALD) of any of the transported movement requirements.
6. *Consistent Arrival time* – The finish time of a trip must not precede the earliest arrival date (EAD) of any of the transported movement requirements.
7. *Consistent Due time* – The finish time of a trip must not be later than the latest arrival date (LAD) of any of the transported movement requirements.
8. *Consistent Trip Separation* – Movements scheduled on the same resource must start either simultaneously or with enough separation to allow for return trips. The inherently disjunctive and relative nature of this constraint makes it more difficult to satisfy than the others.
9. *Consistent Resource Use* – Only the given resources are used.
10. *Completeness* – All movement requirements must be scheduled.

A domain theory formalizing the movement requirement structure, resource models, and constraints may be found in (Smith 1992).

The informal specification above can be expressed as follows:

**function** $TS$
$(mvrs : seq(movement\text{-}record),$
$assets : seq(resource\text{-}name))$
$returns\ (sched : map(resource\text{-}name, seq(trip))\ |$
$Consistent\text{-}POE(sched)$
$\wedge\ Consistent\text{-}POD(sched)$
$\wedge\ Consistent\text{-}Release\text{-}Times(sched)$
$\wedge\ Consistent\text{-}Arrival\text{-}Times(sched)$
$\wedge\ Consistent\text{-}Due\text{-}Times(sched)$
$\wedge\ Consistent\text{-}Trip\text{-}Separation(sched)$
$\wedge\ Consistent\text{-}Pax\text{-}Resource\text{-}Capacity(sched)$
$\wedge\ Consistent\text{-}Cargo\text{-}Resource\text{-}Capacity(sched)$
$\wedge\ Consistent\text{-}Movement\text{-}Type\text{-}and\text{-}Resource(sched)$
$\wedge\ Available\text{-}Resources\text{-}Used(assets, sched)$
$\wedge\ Scheduled\text{-}mvrs(sched) = seq\text{-}to\text{-}set(mvrs))$

This specifies a function called $TS$ that takes two inputs, a sequence of movement records called $mvrs$ and a sequence of resources called $assets$. The function returns a schedule, which has type $map(resource\text{-}name, seq(trip))$ and must satisfy the 11 conjoined constraints. Each constraint is defined in the domain theory; for example:

**function** $CONSISTENT\text{-}DUE\text{-}TIMES$
$(sched : schedule) : boolean$
$= \forall(rsrc : resource\text{-}name,\ trp : integer,$
$mvr : movement\text{-}record)$
$(rsrc \in domain(sched)$
$\wedge\ trp \in [1..size(sched(rsrc))]$
$\wedge\ mvr \in sched(rsrc)(trp).manifest$
$\Longrightarrow$
$sched(rsrc)(trp).start\text{-}time$
$\leq (mvr.due\text{-}date$
$- sched(rsrc)(trp).trip\text{-}duration)$

This predicate expresses the constraint that every scheduled movement-record arrives before its due date.

The problem specified by $TS$ is NP-complete. This problem does not consider certain aspects of transportation scheduling, such as resource utilization rates, crew scheduling, load/unload rates, aircraft maintenance, port characteristics, etc. We have dealt with most of these issues and we are continually developing more complex domain theories, specifications, and schedulers.

## Synthesizing a Scheduler

There are two basic approaches to computing a schedule: local and global. Local methods focus on individual schedules and similarity relationships between them. Once an initial schedule is obtained, it is iteratively improved by "moving" to neighboring schedules. Repair strategies (Zweben, Deale, & Gargan 1990; Minton & Philips 1990; Biefeld & Cooper 1990), case-based reasoning, linear programming, and local search (hillclimbing) are examples of local methods.

Global methods focus on sets of schedules. A feasible or optimal schedule is found by repeatedly splitting an initial set of schedules into subsets until a feasible or optimal schedule can be easily extracted. Backtrack, constraint satisfaction, heuristic search, and branch-and-bound are all examples of global methods. We explore the application of global methods. In the following subsections we discuss the notion of global search abstractly and show how it can be applied to synthesize a scheduler. Other projects taking a global approach include ISIS (Fox & Smith 1984), OPIS (Smith 1989), and MicroBoss (Sadeh 1991) (all at CMU).

### Global Search Theory

The basic idea of global search is to represent and manipulate sets of candidate solutions. The principal operations are to *extract* candidate solutions from a set and to *split* a set into subsets. Derived operations include various *filters* which are used to eliminate sets containing no feasible or optimal solutions (e.g. pruning and constraint propagation). Global search algorithms work as follows: starting from an initial set that contains all solutions to the given problem instance, the algorithm repeatedly extracts solutions, splits sets, and eliminates sets via filters until no sets remain to be split. The process is often described as a tree (or DAG) search in which a node represents a set of candidates and an arc represents the split relationship between set and subset. The filters serve to prune off branches of the tree that cannot lead to solutions.

The sets of candidate solutions are often infinite and even when finite they are rarely represented extensionally. Thus global search algorithms are based on an abstract data type of intensional representations called *space descriptors*. In addition to the extraction and splitting operations mentioned above, the type also includes a predicate *satisfies* that determines when a candidate solution is in the set denoted by a descriptor. See (Smith 1987) for a formal exposition of global search theory.

A simple global search theory of scheduling has the following form. Schedules are represented as maps from resources to sequences of trips, where each trip includes earliest-start-time, latest-start-time, port of embarkation, port of debarkation, and manifest (set of movement requirements). The type of schedules has the invariant (or subtype characteristic) that for each trip, the earliest-start-time is no later than the latest-start-time. A partial schedule is a schedule over a subset of the given movement records.

A set of schedules is represented by a partial schedule. The split operation extends the partial schedule in all possible ways. The initial set of schedules is described by the empty partial schedule – a map from each available resource to the empty sequence of trips. A partial schedule is extended by first selecting a movement record *mvr* to schedule, then selecting a resource $r$, and then a trip $t$ on $r$ (either an existing trip or a newly created one). Finally the extended schedule has *mvr* added to the manifest of trip $t$ on resource $r$. The alternative ways that a partial schedule can be extended naturally gives rise to the branching structure underlying global search algorithms. The formal version of this global search theory of scheduling can be found in (Smith 1992).

### Pruning, Cutting Constraints, and Constraint Propagation

When a partial schedule is extended it is possible that some problem constraints are violated in such a way that further extension to a complete feasible schedule is impossible. In tree search algorithms it is crucial to detect such violations as early as possible.

**Pruning Mechanisms.** *Pruning* tests are derived in the following way. Let *ps* be a partial schedule and let *feasible(sched)* be a predicate that holds if the schedule *sched* satisfies the 12 problem constraints. The test

$$\exists(sched)\ (sched\ extends\ ps\ \wedge\ feasible(sched)) \quad (1)$$

decides whether there exist any feasible completions of partial schedule *ps*. If we could decide this at each node of our branching structure then we would have perfect search – no deadend branches would ever be explored. In reality it would be impossible or horribly complex to compute it, so we rely instead on an inexpensive approximation to it. In fact, if we approximate (1) by weakening it (deriving a necessary condition of it) we obtain a sound pruning test. That is, suppose we can derive a test $\Phi(ps)$ such that

$$\exists(sched)\ (sched\ extends\ ps\ \wedge\ feasible(sched)) \implies \Phi(ps).$$

By the contrapositive of this formula, if $\neg\Phi(ps)$ then there are no feasible extensions of $ps$, so we can prune $ps$. So our backtrack algorithm will test $\Phi$ at each node it explores, pruning those nodes where the test fails.

More generally, necessary conditions on the existence of feasible (or optimal) solutions below a node in a branching structure underlie pruning in backtracking and the bounding and dominance tests of branch-and-bound algorithms (Smith 1987).

It appears that the bottleneck analysis advocated in the constraint-directed search projects at CMU (Fox, Sadeh, & Baykan 1989; Sadeh 1991) leads to a semantic approximation to (1), but neither a necessary nor sufficient condition. Such a *heuristic* evaluation of a node is inherently fallible, but if the approximation is close enough it can provide good search control with relatively little backtracking.

To derive pruning tests for the strategic transportation scheduling problem, we instantiate (1) with our definition of *extends* and *feasible* and use an inference system to derive necessary conditions. The resulting tests are fairly straightforward; of the 12 original feasibility constraints, 6 yield pruning tests on partial schedules. For example, the partial schedule must satisfy *Consistent-POE*, *Consistent-POD*, *Consistent-Pax-Resource-Capacity*, *Consistent-Cargo-Resource-Capacity*, *Consistent-Movement-Type-and-Resource*, and *Available-Resources-Used*. The reader may note that computing these tests on partial schedules is rather expensive and mostly unnecessary – later program optimization steps in KIDS will however reduce these tests to their non-redundant essence. For example, the first test will reduce to checking that when we place a movement record *mvr* on trip $t$, the POE of *mvr* and $t$ are consistent.

For details of deriving pruning mechanisms for other problems see (Smith 1987; 1991; September 1990; Smith & Lowry 1990).

**Cutting Constraints and Constraint Propagation.** Constraint propagation is another mechanism that is crucial for early detection of infeasibility. We developed a general mechanism for deriving constraint propagation code and applied it to scheduling.

Each node in a backtrack search tree can be viewed as a data structure that denotes a set of candidate solutions – in particular the solutions that occur in the subtree rooted at the node. Thus the root denotes the set of all candidate solutions found in the tree.

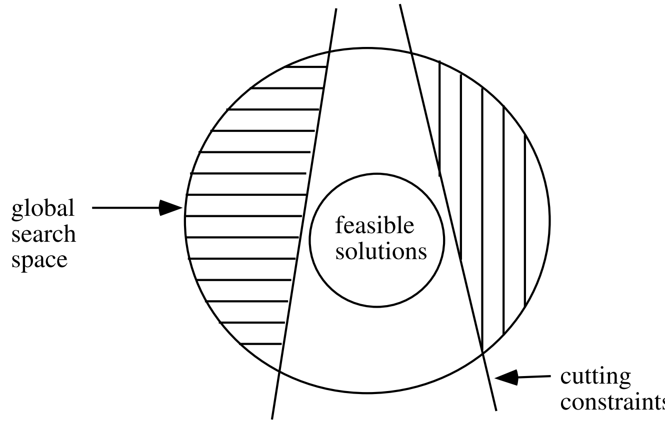


Figure 1: Global Search Subspace and Cutting Constraints

Pruning has the effect of removing a node (set of solutions) from further consideration. In contrast, constraint propagation has the effect of changing the data structure at a node so that it denotes a smaller set of candidate solutions. The basic idea underlying constraint propagation is *cutting constraints* (see Figure 1). Let $\hat{r}$ be a data structure in a global search tree (i.e. a subspace descriptor) that denotes the subspace $S = subspace(\hat{r})$ and let $\psi(z, \hat{r})$ be a cutting constraint where $z$ is a candidate solution. The predicate

$$\xi(\hat{r}) \iff \forall(z)(z \in subspace(\hat{r}) \implies \psi(z, \hat{r}))$$

is a *propagation constraint* that decides whether subspace descriptor $\hat{r}$ incorporates cutting constraint $\psi$. If $\xi$ has a particular Horn-like form described below, then we can mechanically translate $\xi$ into a monotone function on subspace descriptors that has a greatest fixed-point which corresponds to the largest subspace compatible with the constraints. Furthermore, we can then mechanically generate a fixed-point iteration algorithm that converges on a descriptor for that greatest fixed-point. In (Smith, Parra, & Westfold 1995) we describe how to calculate Horn-like propagation constraints from a formal problem specification and how to generate efficient propagation code from Horn-like constraints.

*Definition:* A constraint $\xi$ is *Horn-like* if it has the form

$$B(\hat{r})\ \sqsupseteq\ \hat{r}$$

where $\sqsupseteq$ is a partial order on subspace descriptors such that

$$\hat{r}\ \sqsupseteq\ \hat{s} \implies subspace(\hat{r}) \supseteq subspace(\hat{s}).$$

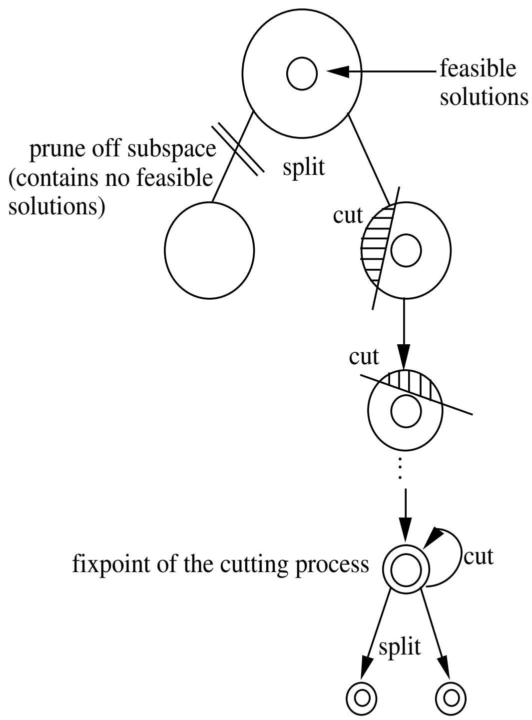


Figure 2: Pruning and Constraint Propagation

The effect of constraint propagation is to propagate information through the subspace descriptor resulting in a tighter descriptor and possibly exposing infeasibility (see Figure 2). In fact the main use for propagation in transportation scheduling is the early detection of infeasibility. For other problems, propagation can serve to reduce the branching factor of the search tree. Propagation can also be used in optimization algorithms to obtain tight lower bounds on the cost of optimal solutions in a subspace (cf. Gomory cutting planes (Nemhauser & Wolsey 1988)).

The mechanism for deriving cutting constraints is similar to (in fact a generalization of) that for deriving pruning mechanisms. We want a necessary condition that a candidate solution $z$ is feasible:

$$\forall(sched, ps)\ (sched\ extends\ ps\ \wedge\ feasible(sched) \Longrightarrow \Psi(sched, ps)\ )$$

By the contrapositive of this formula, if $\neg\Psi(sched, ps)$ then *sched* cannot be a feasible schedule that extends $ps$. So we can try to incorporate $\Psi$ into $ps$ to obtain a new descriptor.

For *TS*, we derive Horn-like constraints and then transform them into a propagation algorithm based on the following recurrence equations, where $est_i$ denotes the earliest-start-time for trip $i$ (analogously, $lst_i$ denotes latest-start-time and $ead_i$ denotes earliest-arrival-time). For each resource $r$ and the $i^{th}$ trip on $r$,

$$est_i \ = \ max \begin{cases} est_i, \\ ead_i - duration(r, POE_i, POD_i), \\ est_{i-1} + dur(i-1, r), \\ max\text{–}release\text{–}time(manifest_i) \end{cases}$$

$$lst_i \ = \ min \begin{cases} lst_i, \\ lst_{i+1} - dur(i, r), \\ min\text{–}finish\text{–}time(manifest_i) \\ \quad -duration(r, POE_i, POD_i) \end{cases}$$

Here $POE_i$ and $POD_i$ represent the ports of embarkation and debarkation of trip $i$ respectively; $dur(r,i)$ is the roundtrip time for trip $i$ on resource $r$:

$$dur(r,i) = duration(r, POE_i, POD_i) \\ \qquad + duration(r, POD_i, POE_{i+1})$$

where *duration(r,p1,p2)* is the travel time from port $p1$ to port $p2$ using resource $r$; *max-release-time*$(manifest_i)$ computes the max over all of the available to load dates, earliest arrival dates - $duration(POE_i, POD_i)$ of movement records in the manifest of trip $i$; also, $est_{i-1}$ is the earliest-start-time of the trip preceding trip $i$, and so on. The boundary cases must be handled appropriately.

The effect of iterating the recurrence equations after adding a new movement record to some trip will be to shrink the $\langle est, lst\rangle$ window of each trip on the same resource. If the window becomes negative for any trip, then the partial schedule is necessarily infeasible and it can be pruned.

Our model of constraint propagation generalizes the concepts of Gomory cutting planes (Nemhauser & Wolsey 1988) and the forms of propagation studied in the constraint satisfaction literature (e.g. (Hentenryck 1989)).

**Constraint Relaxation.** Many scheduling problems are overconstrained. Overconstrained problems are typically handled by relaxing the constraints. The usual method, known as Lagrangian Relaxation (Nemhauser & Wolsey 1988), is to move constraints into the objective function. This entails reformulating the constraint so that it yields a quantitative measure of how well it has been satisfied.

Another approach is to relax the input data just enough that a feasible solution exists. To test this approach, we hand-modified one version of KTS so it relaxes the LAD (Latest Arrival Date) constraint. The relaxation takes place only when there is no feasible solution to the problem data. KTS keeps track of a quantitative measure of each LAD violation (e.g. the difference between the arrival date of a trip and the LAD of a movement requirement in that trip). If there is no feasible reservation for the movement requirement being scheduled, then KTS uses the recorded information to relax its the LAD. The relaxation is such as to minimally delay the arrival of the requirement to its POD.

## Applications

### Strategic Transportation Scheduling

During 1992–1994 we used KIDS to synthesize a variety of TPFDD schedulers, generically called KTS (Kestrel Transportation Scheduler). The *TS* specification presented above was the first in a series of increasingly rich specifications of strategic scheduling as performed by the U.S. Airlift Mobility Command at Scott AFB.

The KTS schedulers are extremely fast and accurate. The chart in Figure 3 lists 4 TPFDD problems, and for each problem (1) the number of TPFDD lines (each requirement line contains up to several hundred fields), (2) the number of individual movement requirements obtained from the TPFDD line (each line can specify several individual movements requirements), (3) the number of movement requirements obtained after splitting (some requirements are too large to fit on a single aircraft or ship so they must be split), (4) the cpu time to generate a complete schedule (on a SUN Sparcstation 5), and (5) time spent per scheduled movement. Similar results were obtained for sea movements.

We compared the performance of KTS with several other TPFDD scheduling systems. We do not have direct access to JFAST and FLOGEN, but these are (or were) operational tools at AMC (Airlift Mobility Command, Scott AFB). According to (Schank 1991) and David Brown (retired military planner consulting with the Planning Initiative), on a typical TPFDD of about 10,000 movement records, JFAST takes several hours (on a Unix workstation) and FLOGEN about 36 hours (on a mainframe). KTS on a TPFDD of this size will produce a detailed schedule in *one to three minutes*. So KTS seems to be a factor of about 25 times faster than JFAST and over 250 times faster than FLOGEN. The currently operational ADANS system reportedly runs at about the same speed as FLOGEN (running on a Korbex machine). When comparing schedulers it is also important to compare the transportation models that they support. KTS has a richer model than JFAST (i.e. handles more constraints and problem features), but less rich than ADANS. The ITAS effort described in the next section reflects our efforts to synthesize schedulers that have at least the richness of the ADANS model.

### In-Theater Airlift Scheduling

The PACAF (Pacific Air Force) Airlift Operations Center at Hickam AFB, Honolulu is tasked with in-theater scheduling of a fleet of 26 C-130 aircraft (plus assorted strategic aircraft on loan) throughout the Pacific region. Current scheduling practice is essentially manual; for example, the relief effort for Hurricane Iniki which struck the island of Kauai in September 1992 was sketched out on 2 sheets of legal paper and required hours of labor.

Since Spring 1994 researchers from Kestrel Institute and BBN (Bolt, Beranek, and Newman) have been working with personnel from PACAF to model the in-theater scheduling problem. The resulting domain theory has been used to synthesize an series of schedulers generically called ITAS (In-Theater Airlift Scheduler). ITAS runs on a laptop computer which makes it useful for both field and command center operations. ITAS can currently produce ATOs (Air Tasking Orders) based on the schedules that it generates. BBN has built the user interface based on the commercial Foxpro database package.

ITAS schedules the Hurricane Iniki data in a few seconds (Burstein & Smith 1996). ITAS has been used in several exercises and was the sole scheduler used in JWID-95 (an international exercise) during September 1995. It is regarded as being ready to use for contingency purposes.

To produce "flyable" schedules it has been necessary to model and schedule a variety of resources, including aircraft, air crews and their duty days, ground crews, parking space for aircraft, and other port restrictions. We have gone through many cycles of learning about the problem from the customer/end-user, elaborating our domain theory, generating new code, and observing PACAF personnel using the scheduler. Although this is a time-consuming process, it seems essential to developing an application that will be used. Nevertheless there has been significant payoff to us as researchers, since the problem features required by the end-user have forced us to generalize and deepen our theories of algorithm design.

The ITAS scheduling theory has evolved over months of effort into about 3500 lines of text. It currently takes about 90 minutes to transform our most complex scheduling specification into optimized and compiled CommonLISP code. Evolution of the scheduler is performed by evolving the domain theory and specification, followed by regeneration of code. The resulting code, after optimizing transformations, is roughly 3000 lines of code in the REFINE language (depending on

| Data Sets (Air only) | # of input TPFDD records (ULNs) | # of individual movements | # of scheduled items after splitting | Solution time | Msec per scheduled item |
|---|---|---|---|---|---|
| CDART | | 296 | 330 | 0.5 sec | 1.5 |
| CSRT01 | 1,600 | 1,261 | 3,557 | 25 sec | 6 |
| 096-KS | 20,400 | 4,644 | 6,183 | 45 sec | 7 |
| 9002T Borneo | 28,900 | 10,623 | 15,119 | 160 sec | 12 |

Figure 3: KTS Scheduling Statistics

how its formatted), which is transformed automatically into about 5200 lines of (largely unreadable) Common-LISP code.

### Power Plant Outage Scheduling

We are continuing to develop new scheduling applications using KIDS. A joint project with the EPRI (Electric Power Research Institute) in Palo Alto, California and Rome Laboratory, focuses on the scheduling of maintenance activities during an outage period at nuclear power plants (Gomes & Smith 1995). Since an outage period costs millions of dollars per day, there is great incentive to quickly generate the shortest possible schedule while achieving all maintenance goals and ensuring safe operations. We have developed a prototype scheduling system, called ROMAN, which has received positive reviews from prospective users.

Current schedulers used by the utility industry are slow and handle only a small subset of the important features of the problem. We have particularly focused on safety constraints since they are critical and they are currently not handled by scheduling tools used in industry.

One safety constraint handled by ROMAN deals with maintaining a sufficient number of backup sources of electric power. We found that this constraint has the same abstract structure as the "MOG" problem in airlift scheduling (ensuring that adequate resources exist to handle the aircraft flow at a particular airport). This insight led us to develop an abstract theory of asynchronously sharable resources that specializes to many domains, including power plant outages, transportation scheduling, parallel processing, manufacturing, and power management (Smith & Westfold 1996). Reusable theories of various classes of resources will help to speed up the process of building domain theories and specifications and cut the time necessary to synthesize new scheduling applications.

## Concluding Remarks

In conclusion, there are several advantages to a transformational approach to scheduling. The first is based on the observation that there is no one scheduling problem. Instead there are families of related problems. The problems can differ in the mix of constraints to satisfy, cost objectives to minimize, and preferences to take into account. In this paper we have mainly treated the problem of finding a feasible detailed schedule. Another kind of problem is to find an estimate of the resources needed to bring about a desired completion date. Another kind of problem is to work backwards from a given completion date to feasible start dates for individual movements. Another kind of problem is incremental or reactive scheduling. We believe that transformation systems such as KIDS will provide the most economical means for generating such families of schedulers. We have observed a great deal of reuse of concepts and laws from the underlying domain theory and of the programming knowledge in the transformations.

A second advantage is the reuse of best-practice programming knowledge. The systematic development of global search algorithms has helped us exploit problem structure in ways that other projects sometimes overlook. The surprising efficiency of KTS stems from two sources. First, the derived pruning and propagation tests are surprisingly strong. The stronger the test, the smaller the size of the runtime search tree. In fact, on many of the TPFDD problems we've tried so far, KTS finds a complete feasible schedule without backtracking! The pruning and propagation tests are derived as necessary conditions on feasibility, but they are so strong as to be virtually sufficient conditions. The second reason for KTS' efficiency is the specialized representation of the problem constraints and the development of specialized and highly optimized constraint operations. The result is that KTS explores the runtime search tree at a rate of several hundred thousand nodes per second, almost all of which are quickly eliminated.

## References


Applegate, D., and Cook, W. 1991. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing* 3(2):149–156.

Biefeld, E., and Cooper, L. 1990. Operations mission planner. Technical Report JPL 90-16, Jet Propulsion Laboratory.

Burstein, M., and Smith, D. 1996. ITAS: A portable interactive transportation scheduling tool using a search engine generated from formal specifications. In *Proceedings of the Third International Conference on Artificial Intelligence Planning System (AIPS-96)*.

Fox, M. S., and Smith, S. F. 1984. ISIS – a knowledge-based system for factory scheduling. *Expert Systems* 1(1):25–49.

Fox, M. S.; Sadeh, N.; and Baykan, C. 1989. Constrained heuristic search. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 309–315.

Gomes, C., and Smith, D. R. 1995. Synthesis of power plant outage schedulers. Technical report, Kestrel Institute. submitted for publication.

Hentenryck, P. V. 1989. *Constraint Satisfaction in Logic Programming*. Cambridge, MA: Massachusetts Institute of Technology.

Luenberger, D. G. 1989. *Linear and Nonlinear Programming*. Reading, MA: Addison-Wesley Publishing Company, Inc.

Minton, S., and Philips, A. B. 1990. Applying a heuristic repair method to the HST scheduling problem. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, 215–219. San Diego, CA: DARPA.

Nemhauser, G. L., and Wolsey, L. A. 1988. *Integer and Combinatorial Optimization*. New York: John Wiley & Sons, Inc.

Sadeh, N. 1991. Look-ahead techniques for micro-opportunistic job shop scheduling. Technical Report CMU-CS-91-102, Carenegie-Mellon University.

Schank, J. 1991. *A Review of Strategic Mobility Models and Analysis*. Santa Monica, CA: Rand Corporation.

Smith, D. R. 1987. Structure and design of global search algorithms. Technical Report KES.U.87.12, Kestrel Institute. to appear in *Acta Informatica*.

Smith, D. R. September 1990. KIDS – a semi-automatic program development system. *IEEE Transactions on Software Engineering Special Issue on Formal Methods in Software Engineering* 16(9):1024–1043.

Smith, D. R., and Lowry, M. R. 1990. Algorithm theories and design tactics. *Science of Computer Programming* 14(2-3):305–321.

Smith, D. R. 1991. KIDS: A knowledge-based software development system. In Lowry, M., and McCartney, R., eds., *Automating Software Design*. Menlo Park: MIT Press. 483–514.

Smith, D. R. 1992. Transformational approach to scheduling. Technical Report KES.U.92.2, Kestrel Institute.

Smith, D. R.; Parra, E. A.; and Westfold, S. J. 1995. Synthesis of high-performance transportation schedulers. Technical Report KES.U.95.6, Kestrel Institute.

Smith, D. R., and Westfold, S. J. 1996. Scheduling an asynchronous shared resource. Technical report, Kestrel Institute. submitted for publication.

Smith, S. F. 1989. The OPIS framework for modeling manufacturing systems. Technical Report CMU-RI-TR-89-30, The Robotics Institute, Carenegie-Mellon University.

Zweben, M.; Deale, M.; and Gargan, R. 1990. Anytime rescheduling. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, 215–219. San Diego, CA: DARPA.