# Transformational Approach to Transportation Scheduling

Douglas R. Smith and Eduardo A. Parra
Kestrel Institute
3260 Hillview Avenue
Palo Alto, California 94304

**Abstract**

*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. In one such problem, the derived scheduler was able to schedule 15,460 individual movement requirements in 71 cpu seconds. The computed schedules use relatively few resources and satisfy all specified constraints. The speed of this scheduler derives from the synthesis of strong problem-specific constraint checking and constraint propagation code.*

## 1 Introduction

This paper describes our exploration of the transformational development of transportation schedulers. 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 semiautomatically 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 guaranteed to be consistent with the given problem specification. 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 [8]. 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. [6, 5, 22, 21]) and Operations Research [2, 11] 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.

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 transportation requirements of an operation, such as Desert Storm or the Somalia relief effort. We developed a domain theory of TPFDD scheduling defining the concepts of this problem and developed laws for reasoning about them. KIDS (Kestrel Interactive Development System) was used to derive and optimize a variety of global search scheduling algorithms that perform constraint propagation [20]. The resulting code, generically called KTS (Kestrel Transportation Scheduler), has been run on a variety of TPFDDs generated by planners at USTRANSCOM and other sites. With one such TPFDD problem, KTS was able to schedule 15,460 individual movement requirements in 71 cpu seconds. The schedule used relatively few resources and satisfied all specified constraints. KTS is orders of magnitude faster than any other TPFDD scheduler known to us.

## 2 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 [17]. 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) [15], divide-and-conquer [15], global search (binary search, backtrack, branch-and-bound) [16], problem reduction generators (dynamic programming, general branch-and-bound, and game-tree search algorithms), and local search (hillclimbing algorithms) [10].
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 [17]. 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 [4].
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.

# 3 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 or resources (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. After a review of the relevant literature we have identified the following general components of a transportation scheduling domain theory.

1. *Requirements* – A model of the requirements can include their internal structure and characteristics, hierarchies of requirement abstractions, and various operations on requirements.
2. *Resources* – A model of the resources can include their internal structure and characteristics, hierarchies of resource abstractions, and various operations on resources.
3. *Time* – A time model can include a calculus of time-points or time-intervals [1, 9].
4. *Constraints* – A constraint model includes the language for stating constraints and a calculus for reasoning about them. A constraint calculus is used to analyze constraints and to propagate the effects of new constraints through a given constraint set.
5. *Objectives* – Typically we seek to minimize the cost of a schedule. Cost can be measured in terms of time to completion, work-in-progress, total cost of consumed resources, and so on.
6. *Scheduling problem* – Using the above concepts we can formulate a variety of scheduling problems. A *reservation* is a triple consisting of an requirement, a resource, and a time interval. Generally, a *schedule* is a set of reservations that satisfy a collection of constraints and optimize (or produce a reasonably good value of) the objective.

The general components of a transportation scheduling domain theory are specialized to TPFDD scheduling in the following ways. Air and sea assets are used to transport forces, supplies and replacement personnel from a port of embarkation (POE) to port of debarkation (POD). A typical air movement requirement has the following information[1]

[1]UHHZ and VRJT are geographical codes for Robins AF Base, Georgia and Sigonella Airport, Italy, respectively.

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

Resources are characterized by their capacities (both passenger (PAX) and cargo capacities for each movement type), travel rate in knots, initial port, and date available.

As an example, one TPFDD schedules the evacution of noncombatants 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, LRWPs, LRWCs, and sea resources include tankers (small, medium, and large), RO-ROs, LASHs, sea barges, containerships, and breakbulks.

Twelve constraints 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 [20]. The problem of satisfying the above constraints is NP-complete. This problem does not consider certain aspects of transportation scheduling, such as resource utilization rates, load/unload rates, port characteristics, etc. We have have dealt with most of these issues and we are continually developing more complex domain theories, specifications, and schedulers.

# 4 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 [23, 12, 3], case-based reasoning, linear programming, and local search (hill-climbing) 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 [6], OPIS [21], and MicroBoss [14] (all at CMU).

## 4.1 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. prunning 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 [16] 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 [20].

### 4.2 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.

#### 4.2.1 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 [16].

It appears that the bottleneck analysis advocated in the constraint-directed search projects at CMU [5, 14] 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 nonredundant essense. 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 [16, 19, 17, 18].

Figure 1: Global Search Subspace and Cutting Constraints

### 4.2.2 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.

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 $c(z, \hat{t})$ be a cutting constraint where $z$ is a candidate solution and $\hat{t}$ an arbitrary subspace descriptor. We define an operation, called $Cut$, that takes $\hat{r}$ and $c$ and generates a new descriptor $\hat{s}$ such that

$$Cut(\hat{r}) = \hat{s}$$
$$\iff subspace(\hat{s}) = \{\ z \mid z \in subspace(\hat{r}) \ \wedge\ c(z, \hat{r})\ \}$$

Constraint propagation is the iteration of $Cut$ until we reach a fixpoint $Cut(\hat{t}) = \hat{t}$ (see Figure 2).

The effect of constraint propagation is to propagate information through the subspace descriptor resulting in a tighter descriptor and possibly exposing infeasibility. 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 [13]).

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:

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

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.

The derived Cut operation has the following form, where $est_i$ denotes the earliest-start-time for trip $i$ and $est'_i$ denotes the earliest-start-time for trip $i$ after applying $Cut$ (analogously, $lst_i$ denotes latest-start-time

Figure 2: Pruning and Constraint Propagation

and $ead_i$ denotes earliest-arrival-time). For each resource $r$ and the $i^{th}$ trip on $r$,

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

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

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 $r$: $dur(r, i) = duration(r, POE_i, POD_i) + duration(r, POD_i, POE_{i+1}))$ where $duration(r, p1, p2)$ is the duration from port $p1$ to port $p2$ using resource $r$; $max\text{–}release\text{–}time(manifest_i)$ 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 this *Cut* operation after adding a new movement record to some trip will be to shrink the $\langle earliest\text{–}start\text{–}time, latest\text{–}start\text{–}time\rangle$ 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 [13] and the forms of propagation studied in the constraint satisfaction literature (e.g. [7]).

### 4.2.3 Constraint Relaxation

Many scheduling problems are overconstrained. Overconstrained problems are typically handled by relaxing the constraints. The usual method, known as Lagrangian Relaxation [13], 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.

## 5 Concluding Remarks

The KTS schedulers synthesized using the KIDS program transformation system 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, and (5) time spent per scheduled movement. Similar results were obtained for sea movements.

We have compared these results with other schedulers. The OPIS project at CMU takes a similar declarative approach to modeling scheduling as a constraint satisfaction problem. However, OPIS effectively interprets its problem constraints, whereas the transformational approach can produce highly optimized "compiled" code. OPIS emphasizes complex heuristics for guiding the search away from potential bottlenecks. In contrast KTS uses simple depth-first search but emphasizes the use of strong and extremely fast pruning and constraint propagation code. OPIS requires about 30 minutes to solve the CDART data and it does not find a complete feasible schedule (some latest arrival dates are relaxed). KTS finds a complete feasible solution in 0.4 seconds – a factor of 4500 faster.

We have also compared these results with the PFE (Prototype Feasibility Estimator), which is a CommonLisp re-implementation of a military feasibility estimator called TFE (Transportation Feasibility Estimator). Including preprocessing time, PFE takes about 206 seconds on the 090TP/PFE data to schedule the sea movements and estimate the schedulability of the air movements. KTS is 78% faster, taking 43 seconds to produce a detailed feasible schedule for *both* air and sea movements. Furthermore the KTS schedule produces 75% less delay in the sea movements and provides a far more accurate measure of the number of planes required for the air movements.

Our conception of the scheduling effort has evolved in significant ways. Our 1991 demonstration system was based on reuse of a general-purpose object base manager and the compilation of declarative constraints into object base demons. We also used a Simplex code to check feasibility of start-times in a generated schedule. The results were somewhat disappointing in that for the 403 movement record problem (CDART), our first code couldn't solve it running overnight, and our second code only produced an incomplete schedule after several minutes. Since speed is of the essence during the scheduling process and the object base and Simplex algorithm are problem-independent, it seemed wise to exploit our transformational techniques to try to derive codes that are problem-specific and highly efficient. Rather than

| Data Sets (Air only) | # of input TPFDD records | # of individual movements | # of scheduled movements after splitting | Solution time | Msec per scheduled movement |
|---|---|---|---|---|---|
| CDART | | 403 | 539 | 0.4 sec | 1.0 |
| CSRT01 | 624 | 1271 | 3120 | 8.3 sec | 2.6 |
| 090TP/PFE | 4471 | 6160 | 8085 | 27 sec | 3.3 |
| 9002T Borneo | 9480 | 12370 | 15460 | 71 sec | 4.6 |

Figure 3: Scheduling Statistics

compile constraints onto an active database, we now derive pruning mechanisms and constraint propagation code operating on problem-specific data structures. Rather than use a Simplex algorithm for finding feasible start-times, the constraint propagation code maintains feasible start-times throughout the scheduling process. The result is that KTS finds a complete feasible solution to the CDART problem in 0.4 seconds.

In summary, 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.

## Acknowledgements

This research was supported by DARPA/Rome Laboratories Contract F30602-91-C-0043 and also by the Office of Naval Research under Contract N00014-90-J-1733 and the Air Force Office of Scientific Research under Contract F49620-91-C-0073.

## References

[1] Allen, J. F. Maintaining knowledge about temporal intervals. *Communications of the ACM 26*, 11 (November 1983), 832–843.

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

[3] Biefeld, E., and Cooper, L. Operations mission planner. Tech. Rep. JPL 90-16, Jet Propulsion Laboratory, March 1990.

[4] Blaine, L., and Goldberg, A. DTRE – a semi-automatic transformation system. In *Constructing Programs from Specifications*, B. Möller, Ed. North-Holland, Amsterdam, 1991, pp. 165–204.

[5] Fox, M. S., Sadeh, N., and Baykan, C. Constrained heuristic search. In *Proceedings of the*

*Eleventh International Joint Conference on Artificial Intelligence* (Detroit, MI, August 20–25, 1989), pp. 309–315.

[6] FOX, M. S., AND SMITH, S. F. ISIS – a knowledge-based system for factory scheduling. *Expert Systems 1*, 1 (July 1984), 25–49.

[7] HENTENRYCK, P. V. *Constraint Satisfaction in Logic Programming.* Massachusetts Institute of Technology, Cambridge, MA, 1989.

[8] JOHN SCHANK. ET. AL. *A Review of Strategic Mobility Models and Analysis.* Rand Corporation, Santa Monica, CA, 1991.

[9] LADKIN, P. Specification of time dependencies and synthesis of concurrent processes. In *9th International Conference on Software Engineering* (Monterey, CA, March 30–April 2, 1987), pp. 106–116. Technical Report KES.U.87.1, Kestrel Institute, March 1987.

[10] LOWRY, M. R. Automating the design of local search algorithms. In *Automating Software Design*, M. Lowry and R. McCartney, Eds. AAAI/MIT Press, Menlo Park, 1991, pp. 515–546.

[11] LUENBERGER, D. G. *Linear and Nonlinear Programming.* Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[12] MINTON, S., AND PHILIPS, A. B. Applying a heuristic repair method to the HST scheduling problem. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control* (San Diego, CA, November 5–8, 1990), DARPA, pp. 215–219.

[13] NEMHAUSER, G. L., AND WOLSEY, L. A. *Integer and Combinatorial Optimization.* John Wiley & Sons, Inc., New York, 1988.

[14] SADEH, N. Look-ahead techniques for micro-opportunistic job shop scheduling. Tech. Rep. CMU-CS-91-102, Carenegie-Mellon University, March 1991.

[15] SMITH, D. R. Top-down synthesis of divide-and-conquer algorithms. *Artificial Intelligence 27*, 1 (September 1985), 43–96. (Reprinted in *Readings in Artificial Intelligence and Software Engineering*, C. Rich and R. Waters, Eds., Los Altos, CA, Morgan Kaufmann, 1986.).

[16] SMITH, D. R. Structure and design of global search algorithms. Tech. Rep. KES.U.87.12, Kestrel Institute, November 1987.

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

[18] SMITH, D. R., AND LOWRY, M. R. Algorithm theories and design tactics. *Science of Computer Programming 14*, 2-3 (October 1990), 305–321.

[19] SMITH, D. R. KIDS: A knowledge-based software development system. In *Automating Software Design*, M. Lowry and R. McCartney, Eds. MIT Press, Menlo Park, 1991, pp. 483–514.

[20] SMITH, D. R. Transformational approach to scheduling. Tech. Rep. KES.U.92.2, Kestrel Institute, November 1992.

[21] SMITH, S. F. The OPIS framework for modeling manufacturing systems. Tech. Rep. CMU-RI-TR-89-30, The Robotics Institute, Carenegie-Mellon University, December 1989.

[22] SMITH, S. F., FOX, M. S., AND OW, P. S. Constructing and maintaining detailed production plans: Investigations into the development of knowledge-based factory scheduling systems. *AI Magazine 7*, 4 (Fall 1986), 45–61.

[23] ZWEBEN, M., DEALE, M., AND GARGAN, R. Anytime rescheduling. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control* (San Diego, CA, November 5–8, 1990), DARPA, pp. 215–219.