ITAS:
A Portable, Interactive Transportation Scheduling Tool Using a Search Engine Generated from Formal Specifications

Mark H. Burstein
BBN Systems and Technologies
10 Moulton St.
Cambridge, MA 02138
Email: burstein@bbn.com

Douglas R. Smith
Kestrel Institute
3260 Hillview Avenue
Palo Alto, California 94304
Email: smith@kestrel.edu

Abstract

In a joint project, BBN and Kestrel Institute have developed a prototype of a mixed-initiative scheduling system called ITAS (In-Theater Airlift Scheduler) for the U.S. Air Force, Pacific Command. The system was built in large part using the KIDS (Kestrel Interactive Development System) program synthesis tool. In previous work for the ARPA/Rome Laboratory Planning Initiative (ARPI), Kestrel has used their *program transformation* technology to derive extremely fast and accurate transportation schedulers from formal specifications, as much as several orders of magnitude faster than currently deployed systems. The development process can produce highly efficient code along with a proof of the code's correctness.

This paper describes the current prototype ITAS system and its scheduling algorithm, as a concrete example of a generated scheduling working on a real problem. We outline the generated search algorithm in order to promote and facilitate comparison with other constraint-based scheduling systems. The overall system includes a database and interactive interface that allows users to control shape of the schedule produced in a number of ways. ITAS runs on a Macintosh Powerbook^{tm} notebook computer, for reasons of portability.

Introduction

This report describes a prototype application of Kestrel Institute's research on the transformational development of high-performance transportation schedulers.[1] The system, ITAS, for In-Theater Transportation Scheduler, is designed to assist human schedulers of airplanes who work to produce daily flight plans to move cargo and people within a specific theater of operations. For example, after the Iniki Hurricane in the Hawaiian Islands a few years ago, these airlift schedulers sent planes from Honolulu and other islands to the Hawaiian island of Kauai to deliver supplies and relief workers to that area. This scenario is not atypical, and was used as a working example as we developed the system.

Previously, Kestrel had demonstrated the potential to produce extremely fast and accurate transportation schedulers from formal specifications using the Kestrel Interactive Development System (KIDS)(Smith September 1990). On test data for strategic transportation plans provided by U.S. government planners, the generated schedulers (called KTS for Kestrel Transportation Scheduler) solved problems with impressive speed. 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 was due to the synthesis of strong constraint checking and constraint propagation code.

In 1994 Kestrel and BBN began to develop a scheduler to support PACAF (Pacific Air Force) at Hickham AFB, Honolulu which is tasked with in-theater scheduling of a fleet of 26 C-130 cargo aircraft in the Pacific region. We developed (and are continuing to evolve) a domain model of theater transportation scheduling. Several variants of a theater scheduler (called ITAS for In-Theater Airlift Scheduler) have been developed to date, and more are planned. The ITAS interface and was built on top of a commercial database package (Microsoft FoxPro^{tm}) and integrated with the generated LISP-based scheduler by BBN. ITAS runs on an Apple Powerbook laptop com-

[1]This research was supported by ARPA/Rome Laboratories under Contracts F30602-91-C-014 (BBN) and F30602-91-C-0043 (Kestrel).

puter. The laptop platform makes it attractive both for field and command center operations. ITAS can currently produce ATOs (Air Tasking Orders) based on the schedules that it generates.

The current ITAS scheduler algorithm is another in the KTS family of synthesized algorithms, using a number of new and different constraints from the previous, strategic (inter-theater) model. In this domain, the size of the problems is smaller (small tens of movement requirements, planes and ports), and the time horizon is shorter (typically, less than one week) because the domain is much more reactive. However, there are many more constraints that must be handled, and users need very rapid response from the scheduler in order to be able to iterate on the final schedule, often in less than an hour. On relatively small but realistic problems we have tested so far[2] in ITAS' intended domain of application, the scheduler runs in 5 to 30 seconds on a Macintosh Powerbook 540. Relatively little effort has been spent on optimizing the code to date.

ITAS simultaneously schedules the following classes of resources: (1) aircraft, (2) air crews and their duty day cycles, (3) ground crews for unloading, and (4) ramp space at ports. By its very dynamic nature, the domain presents many special one-time issues that we leave to the user to handle, as discussed later. Basically, ITAS as a system is designed to allow users control the "shape" of the final schedule by editing and "seeding" the schedule in several ways, and to do daily rescheduling, without losing this user input to the process.

In this paper, we describe the current prototype of ITAS. We discuss the constraints from the domain that were handled, and some that have not, as yet, been handled directly in the generated code. We attempt to describe, in informal terms, the algorithm that was produced using KIDS, in an attempt to make clear where this style of scheduling algorithm gets its great speed, and to promote comparison with other constraint-based scheduling systems in the literature. Finally, we briefly describe the nature of user interactions with the scheduler, as they have evolved so far.

An Approach to Synthesizing Schedulers

Kestrel's approach to developing scheduling software 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 consistent with the given problem specification. Furthermore, the resulting code can be extremely efficient.

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 & Smith 1984; Fox, Sadeh, & Baykan 1989; Smith, Fox, & Ow 1986; 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 checking and constraint propagation.

Our approach tries to make the domain model and scheduling problem explicit and clear, so that, ultimately, users can build new schedulers of the same general class by defining the constraints embodied in their problem and generating a new, situation-specific, scheduler. Basically, the idea is to rapidly develop a situation-specific domain model and problem specification using a knowledge-elicitation system, and then to *synthesize* high-performance planning and scheduling tools that are specialized to the current situation. The majority of users' interaction would be codifying the *domain theory* and specification of the current situation, to aid in synthesizing a customized planning/scheduling tool.

Our current scheduling theories have evolved over months of effort into about 3500 lines of text. It currently takes about 90 minutes to transform our most complex scheduling specification (for ITAS) into optimized and compiled Common Lisp code for Sun workstations. 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 how its formatted), which is transformed automatically into about 5200 lines of (largely unreadable) Common LISP code.

[2]Roughly speaking, 10 planes, 10 movement requirements, generating about 50 flights.

Characterizing the Application Domain

A domain theory for scheduling defines the basic concepts of scheduling and the laws for reasoning about the concepts. A scheduling domain model generally consists of models of the *activities* to be scheduled, the *resources* to be used by those activities, *time* (*e.g.*, a time-interval calculus, the *constraints* on the use of resources for activities (capacity constraints), and the ordering of activities (precedence constraints), and the *utility* of the produced schedule (*e.g.*, minimize a cost objective function).

Using the above concepts we can formulate a variety of scheduling problems. A reservation is a triple consisting of an activity, 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. Transportation scheduling specializes this general notion of scheduling: activities correspond to *movement requirements* and resources correspond to transportation assets such as planes, ships, and trucks. For the ITAS domain, the main assets are planes, but other resources must also be managed including: air crews, ground crews (that load and unload the planes), and parking spaces at airports (for the port constraint called MOG or maximum on ground number).

For ITAS and other transportation schedulers, the activities are called *movement requirements*. ITAS movement requirements include the following information (All times are in seconds from time t_0.):

$$\begin{array}{lcl}
POE : \mathit{port} & \mapsto & PHIK \\
POD : \mathit{port} & \mapsto & PHLI \\
ALD : \mathit{time} & \mapsto & 0 \\
EAD : \mathit{time} & \mapsto & 86400 \\
LAD : \mathit{time} & \mapsto & 86400 \\
\mathit{Loads} : \mathit{integer} & \mapsto & 20 \\
\mathit{Load\text{–}type} : \mathit{aircraft\text{-}class} & \mapsto & C130E \\
PAX : \mathit{integer} & \mapsto & 1500 \\
\mathit{Pallets} : \mathit{integer} & \mapsto & 50
\end{array}$$

Here, the Port of Embarkation (POE) and Port of Debarkation (POD) are the origin and destination of the cargo, respectively. PHIK and PHLI are ICAO codes for Hickham AFB and Lihue Airport, respectively. The ALD (available to load date), EAD (earliest arrival date), LAD (latest arrival date) represent the time bounds on a feasible schedule for the movement requirement. The cargo itself is represented somewhat differently from most strategic transportation models. For a problem at this level (smaller planes, local scheduling), packing of the aircraft is a crucial issue, and irregular size and shape loads must be considered. We do not address the packing problem in this system, so we instead rely on the user to specify irregular cargo in *full Loads* for a specific kind of aircraft, and with a particular configuration of the plane's cargo area (indicated by the Load-type). More regular cargo (PAX and Pallets) may be loaded onto any aircraft that has available space for that type of cargo.

The ITAS schedulers have emphasized efficient search and rich constraint modeling. The current version of ITAS simultaneously schedules the following types of resources, each of which has a variety of constraints associated with it:

1. **Aircraft** are characterized by their capacities, both passenger (PAX) and cargo (pallet) capacities, and travel rate in knots. They also have minimum runway length requirements in order to be acceptable for use at specific airports.

2. **Air crews** typically have 16 hour work days, followed by 12 hours rest. They are not expected to take off until one hour after they are called for duty.[3] They also receive a day off after a short number of days on duty. When only one crew is available for a plane, the plane does not fly during crew rest periods.

3. **Ground crews** are scheduled for loading and unloading planes at each port.

4. **Parking spaces** at each port are used to model the MOG (Maximum on Ground) constraint on ports where this applies.

5. **Airports** are not treated as resources themselves, but have restrictions like their maximum runway length, and operating hours, and whether they are in a combat zone that impact flights into and out of those locations. They also organize other resources (ground crews, parking spaces).

Ground crews and parking slots (MOG) at airports are essentially aggregate resources in a naive description of the ITAS problem, but are scheduled as if they were individual unit resources in the current system, since KIDS cannot currently generate code that treats these constraints in their aggregate form. This is an area of ongoing research.

ITAS' Scheduling Constraints

At present, twenty-three constraints characterize a *feasible schedule* for ITAS[4]. The list below collapses the

[3]Thus far, we only handle one air crew permanently assigned to each aircraft. This should be fixed by the time of this publication.

[4]Some of these constraints (*'d) are not currently given explicitly as part of the top level *Problem Specification* to

descriptions of some of these together[5]:

1. *Consistent POE and POD** – The POE and POD of each movement requirement on a given trip of a resource must be the same as that flight's origin and destination respectively.

2. *Consistent Load Type** – Each resource can handle only loads for some movement requirements. For example, a C-141 aircraft can only carry loads of C-141 cargo.

3. *Consistent PAX and Pallet Capacity** – For cargo in movement requirements expressed as PAX (passengers) or Pallets (pre-packaged cargo), the capacity of each aircraft to carry that type of cargo cannot be exceeded on a given flight. A flight containing a full *load* can carry no additional PAX or Pallets.

4. *Consistent Release Time* – The start time of a movement (the flight's earliest departure time (EDT)) must not precede the release time (ALD) of all movement requirements in the flight's manifest, plus the aircraft's load time.

5. *Consistent Due Time* – The finish time of a movement (the flight's latest departure time + flight duration + unload time) must not be later than the Latest Arrival Date (LAD) of all movement requirements in the flight's manifest.

6. *Consistent Flight Separation* – For flights of the same aircraft, the earliest (latest) departure time plus the flight duration and on-ground time at the destination must be less than the earliest (latest) departure time of the aircraft's next flight.

7. *Consistent Air Crew Usage* – Only use the given air crews.

8. *Consistent Air Crew Duty Day* – For a sequence of flights by one crew without a rest period, the earliest (latest) departure time of the last flight, plus the last flight's duration and unload time, should be less than the length of a duty day after the earliest (latest) departure time of the first flight.

9. *Consistent Air Crew Transition* – By similar formulae, crews get a full 12 hours rest after a duty day plus at least 3.25 hours preparation time before their next flight.

10. *Consistent Air Crew Qualifications* – Crews must be qualified for the aircraft they are assigned to, the mission types to be flown, and must belong to the unit that 'owns' the aircraft. (3 constraints)

11. *Consistent Port Usage* – Only schedule flights into/out of the given ports.

12. *Consistent Port Runway Length* – Aircraft cannot land or take off from airports whose maximum runway length is less than the aircraft's minimum take-off/landing runway length[6].

13. *Consistent Port MOG* – The constraints here are actually on the use of parking spaces, which are assigned individually during search. Basically they state that the earliest (latest) departure time of the aircraft leaving a space must be before the earliest (latest) arrival time of the next aircraft assigned to that space.

14. *Consistent Port Ground Crew* – Similar to MOG, ground crew reservations must be separated by the corresponding load/unload time. For now, it is assumed that there are a constant number of ground crews available whenever a port is open.

15. *Consistent Port Operating Hours* – Ports may be closed for take-offs/ landings for some period each day (*e.g.*, night time). (2 constraints)

16. *Consistent Mission Type* – The cargo in a flight manifest must be from movement requirements with the same mission type (*e.g.*, normal land and unload vs. airdrop).

17. *Consistent Aircraft Usage* – Only the given aircraft are to be used.

18. *Completeness* – All movement requirements must be scheduled.

Constraints in the domain theory are defined formally by reference to data structures that are maintained during search, basically mappings of resources (parking spaces, ground crews) to sequences of reservations (tuples referencing the aircraft and flight involved). For example, here is a (slightly simplified)

the KIDS system, but are implicit in the *Global Search Theory*, another part of the domain theory that is essentially an abstract schema describing the branching structure for the search control model to be employed in the generated algorithm. These constraints were "rolled into" that structure for the sake of the efficiency of the generated code. Future systems will make these constraints explicit again.

[5] The number of corresponding formal constraints is noted in parenthesis. Typically, temporal constraints come in pairs, one that is used to propagate the Earliest Departure Time (EDT) forward, and one that is used to propagate the Latest Departure Time (LDT) backward through the schedule.

[6] This constraint could be made considerably more complex if it were to take into account whether the plane was loaded or not, its expected fuel level, and the different true minimums for take-off and landing.

constraint on the LAD of a movement requirement:

function *CONSISTENT-LAD*
$(sched : schedule) : boolean$
$= \forall(ac - indx : integer,\ flt : integer,$
$mvr : movement\text{-}record,\ flt\text{-}indx : integer,\ lat : time)$
$(ac\text{-}indx \in domain(sched)$
$\wedge\ flt\text{-}indx \in domain(sched(ac\text{-}index).flight\text{-}sched)$
$\wedge\ flt = sched(ac\text{-}index).flight\text{-}sched(flt\text{-}indx)$
$\wedge\ mvr \in flt.manifest$
$\wedge\ lat = flt.latest\text{-}departure\text{-}time + flt.iflt\text{-}duration$
$\implies$
$mvr.LAD \geq (lat + flt.unload\text{-}time)$

This predicate expresses the constraint that every scheduled movement-record arrives and is unloaded before its latest arrival date.[7] The generated code checks this constraint and propagates any time bounds changes that its enforcement causes.

Similarly, air crew constraints are defined by reference to a mapping from air crews to a data structure containing the sequence of flights they are scheduled to be on, plus other associated information. These data structures are *defined as part of the domain theory.* They are not generated automatically. Data structure design and refinement is an explicit goal of the next generation KIDS-like environment, Specware (Srinivas & Jüllig 1994).

Preferential Constraints

A typical scheduling problem will involve some choices best expressed as preferences or prioritizations for the use of resources. Although some experiments have been done with KIDS where its search was based in part on a utility or cost function, the current ITAS scheduler does not explicitly handle preferential constraints at all. Instead, at points where choices are made about which resources to use, user-defined functions order the choices. We have experimented with a number of different ordering heuristics, to improve the overall quality of the schedules produced, from the user's perspective.

There are functions (called variously *ASSET-ORDER, AIR-CREW-ORDER ...*) that define the order of consideration of each kind of resource:

Movement Requirements – *E.g.*, sorted by their LAD, EAD, ALD.

Assets (aircraft) – *E.g.*, prefer locally available aircraft.

Air Crews – *E.g.*, prefer the most rested air crews from the aircraft's home unit.

Ground Crews – Prefer the earliest available crew.

Parking Spaces – Prefer the earliest available space.

[7] A schedule is defined as a sequence of aircraft (which are data structures that contain their flight schedules, also sequences). Thus $sched(i)$ indexes the i^{th} aircraft, and $domain(sched)$ is the integers $[1..k]$ if there are k aircraft available.

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 structurally similar schedules. Repair strategies (Zweben, Deale, & Gargan 1990; Minton *et al.* 1990; Biefeld & Cooper 1990; Selman, Levesque, & Mitchell 1992), and fixed-point iteration (Cai & Paige 1989), and linear programming algorithms 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, heuristic search, and branch-and-bound methods are all examples of global methods. We used a global methods to generate the KTS family of schedulers, including ITAS. Other projects taking a global approach include ISIS (Fox & Smith 1984), OPIS/DITOPS (Smith 1989), and MicroBoss (Sadeh 1991) (all at CMU).

Global search algorithms 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. Starting from an initial set that contains all solutions to the given problem instance, these algorithms 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.

In a simple global search theory of scheduling, schedules are represented as maps from resources to sequences of trips, where each trip includes earliest-start-time, latest-start-time, travel-time, port of embarkation, port of debarkation, and a manifest describing the cargo. This 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. Thus the root denotes the set of all candidate solutions found in the tree. This initial (partial) schedule is just the empty schedule – a map from the available resources 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) – the triple $\langle mvr, r, t\rangle$ represents a possible refinement of the current partial schedule in the search space. The alternative ways that a partial schedule can be extended naturally gives rise to the branching structure underlying global search algorithms.

A global search algorithm checks for consistency of the partial solution 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).

Cutting Constraints and Temporal Constraint Propagation

A key technical achievement of the Kestrel work was discovering and implementing technology for generating efficient constraint propagation code. The speed of the KTS schedulers derives from the extremely fast checking and propagation of constraint information at every node of the runtime search tree. Whereas some knowledge-based approaches to scheduling will search a tree at the rate of several nodes per second, some of the synthesized schedulers search *several hundred thousand* nodes per second.

The idea is to derive and utilize the necessary conditions on feasibility of a candidate (partial) schedule. These conditions are called *cutting constraints*. The derived cutting constraints for a particular scheduling problem are analyzed to produce code that iteratively fixes violated constraints (for the current KTS schedulers, by tightening time bounds on reservations) until the cutting constraints are satisfied. This iterative process subsumes the well-known processes of constraint propagation in the AI literature and the notion of cutting planes from the Operations Research literature(Smith & Westfold 1995).

Kestrel researchers developed a general mechanism for deriving constraint propagation code and applied it to scheduling. This model of constraint propagation generalizes the concepts of cutting planes in the Operations Research literature (Nemhauser & Wolsey 1988) and the forms of propagation studied in the constraint satisfaction literature (e.g. (Hentenryck 1989)). [8] The use of fixed-point iteration for constraint propagation is similar to Paige's work on fixed-point iteration in RAPTS (Cai & Paige 1989).

KIDS generates propagation code automatically from a subset of the constraints given for a problem, using information in the global search theory schema as a guide. Stephen Westfold of Kestrel was responsible for the design and implementation of the propagation generation subsystem of KIDS (Smith, Parra, & Westfold 1995). The constraint propagation code that was generated for the original KTS scheduler is nearly as fast as handwritten propagation code for the same problem (cf. Appendix C in (Smith 1992)). The propagation code for the current ITAS is many times more complicated, and it would have been quite difficult to generate it by hand.

The generated propagation code called at each node during search operates much the way a temporal constraint system does, although the code is targeted to the problem domain using specific knowledge of the particular constraints handled, the choices being considered at that point by the scheduler, and a logical analysis of the possible effects of tightening particular time bounds. The result is code tailored to the specific scheduling problem addressed, based on the constraints specified in the domain theory. The overall effect is very efficient pruning of the search space. At each node in the space, after each potential choice is made by satisfying all relevant non-temporal necessary constraints, the propagation code is run to see if the schedule is still viable. If so, the system recurs on the new refined state. Otherwise, a new choice is made.

The ITAS Scheduler Algorithm

This section provides a brief outline of the generated scheduler's algorithm, in order to convey a sense of how the system works, and (hopefully) why it is fast. Due to space limitations, we have greatly simplified and summarized what goes on. The code itself is extremely complex and barely readable, with few function breaks and numerous generated intermediate variables.

In ITAS, a *schedule* is a list of all of the aircraft, where each aircraft is an object that contains its (partial) schedule. A *sortie* is essentially a short sequence of flights that contains (at most) a positioning flight to get to a POE, a POE-to-POD flight, and a recovery flight away from the POD to a refueling or resting station.

The top level function, KTS, just calls $KTS\text{-}AUX$, which recursively searches the space of partial schedules. KTS then tests to see if the result is defined and

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

UNSCHED-MVRS is empty, in which case it extracts a solution from the current search state.

The arguments to *KTS-AUX* constitute a state in the search space:

1. *MVRS*: a sorted list of all original movement requirements
2. *ASSETS*: a list of all available aircraft for the problem
3. *PORTS*: a list of all of the ports in the problem
4. *AIR-CREWS*: a list of all air crews
5. *PSCHED*: a partially completed schedule
6. *SCHED-MVRS*: all of the movement requirements that are already scheduled
7. *UNSCHED-MVRS*: all of the remaining unscheduled movement requirements
8. *AIR-CREW-MAP*: a mapping of air crews to their schedules
9. *GND-CREW-MAP*: a mapping of ground crews to their schedules
10. *GS-FLTS*: flights remaining to be scheduled for the current sortie
11. *PRK-SLOT-MAP*: a mapping of parking spots at ports to their reservations

KTS-AUX for the ITAS problem splits the search into four main branches, as defined by the global search theory schema of the domain theory. The first two branches consider refinements of the current partial schedule for flights that are part of the current sortie. A current sortie exists while a movement requirement is being processed, but not all of the necessary flights (positioning, cargo-moving, depositioning) have been scheduled (so *GS-FLTS* is not empty). The third and fourth branches consider the next movement requirement, and search to place as much of that requirement as possible on an existing or new flight, respectively. In the process, a new sortie is created.

Within each of the four main branches of *KTS-AUX*, one of two versions of *SPLIT&PROPAGATE* is called, to generate the proposed new state, and run the propagation routines for each temporal constraint that might be violated at that point. If *SPLIT&PROPAGATE* succeeds and returns a new state, When *KTS-AUX* finds a possible a new state, it calls *SPLIT&PROPAGATE* with these additional arguments *S-State* to represent the currently considered resources:

1. *FLT*: the currently considered flight, if any
2. *POE-TO-POD-FLT*: the load-carrying flight for this move reqt., if known
3. *AIR-CREW*: the flight's air crew
4. *CURRENT-DUTY-DAY?*: Boolean indicating to search in the current (next) duty day
5. *GROUND-CREW*: the ground crew scheduled to unload the plane, if defined
6. *GND-CREW-RES*: the reservation for that crew, if defined.
7. *PARKING-SLOT*: the parking slot to be used at the destination, if defined

The branches, considered sequentially, are:

1. If *GS-FLTS* is not empty, consider all possible ground crews and parking slots for
   $$FLT = first(GS\text{-}FLTS)$$
   and
   $$MVR = first(UNSCHED\text{-}MVRS) \quad ,$$
   by calling *SPLIT&PROPAGATE1* on each, but considering *only flights in the current duty day* of the flight's air crew.
2. If *GS-FLTS* is not empty, consider all possible ground crews and parking slots for
   $$FLT = first(GS\text{-}FLTS)$$
   and
   $$MVR = first(UNSCHED\text{-}MVRS) \quad ,$$
   by calling *SPLIT&PROPAGATE1* on each, but considering *only flights in the next duty day* of the flight's air crew.
3. If *GS-FLTS* is empty, let
   $$MVR = first(UNSCHED\text{-}MVRS) \quad ,$$
   and consider all cargo-carrying flights of aircraft in the current *S-State*'s partial schedule that have remaining capacity for *MVR* by calling *SPLIT&PROPAGATE2* on each.
4. If *GS-FLTS* is empty, let
   $$MVR = first(UNSCHED\text{-}MVRS) \quad ,$$
   and consider all aircraft in *ASSET-ORDER*, calling *SPLIT&PROPAGATE2* on each, which creates a flight sortie for that *MVR*, and tests that requirement's LAD can be met.
5. (optional) If no feasible (partial) schedule is found, relax the LAD of the last movement requirement.

Each *SPLIT&PROPAGATE* function contains a version of the generated constraint checking and propagation code for a different set of conditions, depending on how the partial schedule has been changed. Essentially, these functions take the constraints stated in the problem description, in turn, and, if they are temporal constraints, propagate their bounds either forward or backward in time, depending on whether the earliest (EDT) or latest departure time (LDT) of a flight is referenced in the constraint. For example, *SPLIT&PROPAGATE1*, which is used to schedule an existing flight in *GS-FLTS*, does the following:

1. Check *CONSISTENT-RUNWAY-LENGTH*.
2. Check *CONSISTENT-MISSION-TYPE*.
3. Generate a state vector with reservations for the Ground Crew, Air Crew, and Parking Slot assignments for the flight
4. Propagate EDT, checking *CONSISTENT-AIR-CREW-DUTY-DAY-EDT*
5. Propagate LDT, checking *CONSISTENT-AIR-CREW-DUTY-DAY-LDT*
6. Propagate EDT, checking *CONSISTENT-FLIGHT-SEPARATION-EDT* for all aircraft used in *PSCHED*.
7. Propagate LDT, checking *CONSISTENT-FLIGHT-SEPARATION-LDT* for all aircraft used in *PSCHED*.
8. Propagate EDT, checking *CONSISTENT-ALD* for all MVR in the manifest.
9. Propagate EDT, checking *CONSISTENT-GROUND-UNLOAD-CREW-EDT* for all of the current ground crews at the destination.
10. Propagate LDT, checking *CONSISTENT-GROUND-UNLOAD-CREW-LDT* for all of the current ground crews at the destination.

Each propagation of the EDT or LDT above is based on recursively checking the corresponding constraints for *CONSISTENT-FLIGHT-SEPARATION*, *CONSISTENT-AIR-CREW-DUTY-DAY*, *CONSISTENT-AIR-CREW-TRANSITION*, and *CONSISTENT-MOG*.

If the state is still defined (consistent) at the end of all of this, the proposed state is accepted, the movement requirement is marked scheduled and *KTS-AUX* is called recursively.

SPLIT&PROPAGATE2 is similar, but somewhat more complicated, since it generates a sortie of flights for the next movement requirement, and finds an aircraft and crew for that sortie.

Constraint Relaxation

Many scheduling problems are over-constrained. Over-constrained 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.

Our current experimental approach is to relax the input data just enough that a feasible solution exists (the optional branch 5 of *KTS-AUX*). This approach is available as an option in ITAS to avoid an exhaustive search when no feasible solution may exist. This mode relaxes the LAD (Latest Arrival Date) constraint, when the current search reaches an impasse, rather than backing up. The relaxation takes place only when there is no feasible solution to the problem data. ITAS uses the difference between the arrival date of a trip and the LAD of a movement requirement in that trip as a measure of how much to relax the LAD. The relaxation is such as to minimally delay the arrival of the requirement to its POD.

System Design and User Interactions Issues with ITAS

The target user community for ITAS needed a tool that was fast, easy to use, flexible and portable. These factors motivated us to develop the system on Macintoshes, so that it could be deployed on a Mac Powerbook notebook computer. Given the speed of the scheduler, size of the problems to be faced, and the short scheduling horizon, we expect that most problems will run in under a minute with the current system. We used Microsoft FoxPro^{tm} as the primary database and interface substrate, because it runs on Macintoshes and it was simple to understand. For the current prototype, data exchanges with the scheduler (running in Macintosh Commonlisp) are done using a simple combination of Applescript and file I/O. On our current suite of test problems, it takes as long to do the inter-module I/O as it does to build a schedule.

The interface uses Apple's menu system and set of data entry screens, organized around two types of data: *background data* (for caches of locations (airports), units, aircraft types and characteristics, frequently used aircraft, etc.) and *situation data*, the current collection of aircraft, crews, ports and movement requirements for which a schedule is being built. The primary graphical schedule display is a mouse sensitive version of a schedule Gantt chart modeled after what the airlift schedulers call their 'Rainbow' chart, because the different types of aircraft are assigned different colors.

There is also a flight editor that enables users to make modifications to the existing schedule before exporting it, and to iterate on a solution with the scheduler in the loop. Users of this tool need to be able to edit the schedules produced, when circumstances not captured by the available constraints come up. We needed to make it possible for parts of the schedule to be specified directly by the user, and not revised by the scheduler.

The ITAS domain is extremely reactive. The people who now generate schedules by hand do so a day at a time, since their information is often only good enough to *estimate* what will be required three, or even two days hence. Given a tool that will build schedules for them, they will likely extend their planning horizon (currently 10 days or less), at least for plan/resource evaluation purposes, but for day to day operations, they will continue to schedule only a few days at a time, and do daily rescheduling, as needed.

In support of rescheduling, users can mark (or edit then mark) flights in the existing current schedule as 'Frozen'. ITAS-KTS takes as part of its input the leading frozen edge of a schedule, and uses it only to properly reflect resource availability, and so that those flights and their itineraries are reflected in its output. Because the ITAS-KTS scheduling algorithm generates candidate flights in a time forward fashion, a frozen flight necessarily means that all prior flights by that aircraft are also frozen. Thus the current scheduler does not give users as much flexibility to edit and then rerun schedules as is found in some other global-search based rescheduling systems whose search is not time ordered (*e.g.,* (Smith 1989)) or in systems using local repair strategies(Zweben, Deale, & Gargan 1990). We are planning to generate an ITAS scheduler that inserts flights in a schedule rather than adding them to the end of schedule. The interesting issue is to evaluate the resulting tradeoff between the slowdown due to greatly increased search tree branching and the extra flexibility gained during (re)scheduling.

Our users also wished to be able to lay out patterns of flight itineraries for some sets of aircraft, based on their paper process. For example, it is customary for them, when scheduling a group of planes carrying loads to the same destination, to stagger the departure times to assure a uniform flow into an airport. We have not yet been able to reproduce this style of output automatically, but have provided a mechanism that allows users to lay out itineraries (sequences of ports to visit) for sets of aircraft, each departure offset by some specified time interval. Flights of this kind that are seen by ITAS-KTS as having fixed earliest departure times, but infinite latest departure times, so they can effectively be "slid forward" in time to accommodate port and ground crew constraints. A planned extension of this capability is to provide these "flows" to the scheduler as patterns that can be used repeatedly in normal scheduling, especially where the distance is great enough that an intermediate stop en route is required when transporting some cargo. The current scheduler does not support multi-leg trips between cargo sources and destinations.

Conclusions

This project was conceived of as a short term effort to demonstrate the potential and practical utility of automatically generated scheduling software. Given the previously demonstrated speed of the algorithms produced and this demonstration that the technique can be scaled up to realistic sets of constraints, we believe that those conclusions are clearly justified. This paper is an attempt to make clear how the current class of generated schedulers work, why they are fast, and what some of the tradeoffs were in terms of flexibility. We believe that it will often be better to generate, and make available through a single interface, a suite of quick schedulers, tailored to different constraints, than to have one scheduler that is slow on all problems.

The great advantage of the synthesis approach is the ability to expose problem structure and exploit it by transformationally deriving efficient problem-specific code. In this case, the reuse of design knowledge (global search, constraint propagation) made an enormous difference in performance. Future directions include making it easier for domain experts to specify their own constraints, and generate their own problem-specific schedulers, and broadening the classes of scheduling algorithms that can be generated, and the types of constraints that can be handled. The next generation of KIDS, Specware, will also address the need for more automatic and controllable data structure design. We should note that a preliminary test indicates that there is at least *an order of magnitude speedup* still to be had, since the current scheduler uses *lists* for all of its object data structures! Specware will be able to use a variety of data types, and generate code in other languages than LISP (*e.g.*, C++).

On the interface side, we seek to exploit the great speed of even the current class of schedulers by greatly increasing the interactive nature of the scheduling task, providing more automatic capabilities in the area of resource analysis and comparative schedule analysis.

Acknowledgments – The authors would like to thank the following people for their important contributions to the ITAS effort: Don Roberts and John Lemmer of Rome Laboratory; Stephen Westfold of Kestrel Institute; and LtCol Chris Stuhldreher, Maj Keith LaCrosse, and Cpt Rob Burgess of the PACAF Airlift Operations Center.

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.

Cai, J., and Paige, R. 1989. Program derivation by fixed point computation. *Science of Computer Programming* 11:197–261.

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.

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.; Johnson, M.; Philips, A. B.; and Laird, P. 1990. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the Eighth National Conferenceon Artificial Intelligence*, 290–295.

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.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conferenceon Artificial Intelligence*, 440–446.

Smith, D. R. 1987. Structure and design of global search algorithms. Technical Report KES.U.87.12, Kestrel Institute.

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., and Westfold, S. J. 1995. Synthesis of constraint algorithms. In Saraswat, V., and Hentenryck, P. V., eds., *Principles and Practice of Constraint Programming*. Cambridge, MA: The MIT Press.

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, S. F.; Fox, M. S.; and Ow, P. S. 1986. Constructing and maintaining detailed production plans: Investigations into the development of knowledge-based factory scheduling systems. *AI Magazine* 7(4):45–61.

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

Srinivas, Y. V., and Jüllig, R. 1994. Specware:^{tm} formal support for composing software. Technical Report KES.U.94.5, Kestrel Institute. To appear in Proceedings of the Conference on Mathematics of Program Construction, Kloster Irsee, Germany, July 1995.

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.