ICAPS 2005
Monterey, California

ICAPS05

TU4

Tutorial on Domain Modeling for Planning

Mark Boddy
Adventium Labs, Minneapolis, USA

Robert Goldman
SIFT, LLC, Minneapolis, USA

ICAPS 2005
Monterey, California, USA
June 6-10, 2005

CONFERENCE CO-CHAIRS:
Susanne Biundo
University of Ulm, GERMANY

Karen Myers
SRI International, USA

Kanna Rajan
NASA Ames Research Center, USA

Cover design: L.Castillo@decsai.ugr.es

Tutorial on Domain Modeling for Planning

Mark Boddy
Adventium Labs, Minneapolis, USA

Robert Goldman
SIFT, LLC, Minneapolis, USA

Tutorial on Domain Modeling for Planning

Table of contents

ICAPS 2005
Monterey, California

Tutorial on Domain Modeling for Planning

Preface

For a planning application, how the domain is modeled can mean the difference between success and failure. In this tutorial, we present examples of modeling challenges drawn from a broad range of practical applications, including manufacturing, UAV control, and space operations, as well as some of the domains used in the most recent International Planning Competition. For each of these applications, we discuss and illustrate the pros and cons of various modeling approaches, including PDDL, various HTN schema representations (e.g., SHOP2, ACT, O-Plan) logical formalisms, and constraint-based representations such as NASA's DDL.

Despite an explicit attempt in many of these representations to follow McDermott's dictum regarding representing physics rather than advice, how planning problems are modeled interacts strongly with how they are solved. Some of the ways this manifests may be surprising, for example the presence of preconditions in an operator purely for the purpose of binding a variable, or the ordering of preconditions in a conjunct so as to minimize the number of ground operators or variable bindings considered

Domain modeling for planning is very similar to domain modeling in conventional software engineering. Few if any current languages provide effective engineering support for the modeling process. In addition, for many applications the most significant factors in generating a solution are not easily represented or manipulated in the available formalisms. For example, it is cumbersome at best to model planning domains dominated by resource management in PDDL and other STRIPS derived languages, which have no explicit resource model. Many planning languages make it difficult to encode operators with complex context-dependent effects, such as sending an email message with attachments. Few current planning languages make any attempt to model asynchronous, overlapping continuous change, such as simultaneous charging and drawing from a battery in a space probe, or simultaneous drawing from and filling of a tank in an oil refinery.

We discuss these and other modeling issues, their effects, and work-arounds. In the course of the tutorial, we provide worked examples in multiple formalisms for qualitatively different application domains. Instead of arguing for a preferred language, we illuminate the strengths and weaknesses of current approaches, for various types of application. We will also discuss techniques for construction and maintenance of planner domain models and suggest future directions.

Instructors

- *Mark S. Boddy, Adventium Labs*
  mark.boddy@adventiumlabs.org
- *Robert P. Goldman, Smart Information Flow Technologies, LLC*
  rpgoldman@sift.info

Domain Modeling for Planning

Mark S. Boddy
Adventium Labs
mark.boddy@adventiumlabs.org

Robert P. Goldman
Smart Information Flow Technologies
rpgoldman@sift.info

 1

Outline of the Presentation

- Planning methods and languages
- Domain characteristics and modeling requirements
- Worked examples
  - Intrusion detection
  - UAV
  - Airport surface movement
  - Process-industry manufacturing
- Summary and Conclusions
  - Takeaway messages
  - Things we didn't have time to cover
  - For further information

 2

Role of Modeling in Planning

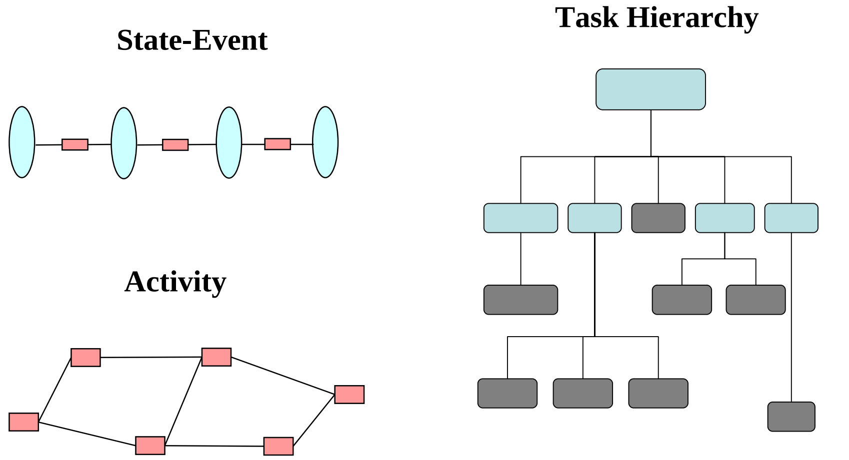
- Planners use a projective model of domain dynamics to find a procedure or program that will satisfy a goal specification.
- Domain modeling is a design problem with two objectives:
  - Providing useful predictions of system behavior, for a given plan.
  - Supporting efficient plan generation, including efficient projection.
- Not surprisingly, since the planning problem is intractable, the details of the model will interact with the precise planning method.
- We will not have time to talk about plan execution, and so will have little to say about managing uncertainty, although both of these are important issues.

Plan Objectives

- Goals of achievement: find a sequence of actions leading to a state in which some property is satisfied.
  - Classical planning
- Goals of maintenance/avoidance: over the extent of the plan, ensure that some property is satisfied.
  - Resources
  - State

While it is certainly possible to represent properties of the whole trajectory in state/event models, it may not be the easiest or most effective method.

Three Kinds of Plan Structure



Relative Strengths

- **State-event** models are good for representing:
  - Precondition/effect interactions among discrete actions
  - Complex system dynamics
- **Activity** models are good for representing:
  - Asynchronous, overlapping activities with some interactions (e.g., resources).
  - Reasoning about time "from the side"
- **Task hierarchies** are good for representing:
  - Explicit sequences or trajectories of steps
  - Alternative means of accomplishing something
  - Task interactions that are too messy/obscure to model explicitly

Why This Matters

- Planner performance may be limited by any of the following:
  1. Choice of modeling language
  2. Modeling decisions
  3. Choice of algorithm
  4. Choice of heuristic
- We will focus on ##1 and 2, but will not be able to avoid ## 3 and 4.
- Specific challenges:
  - Scale
  - Complex dynamics
    - Continuous dynamics
    - Procedural attachment
  - Resources
  - Reasoning about trajectories
  - Optimization

Representation Languages

- STRIPS family
  - STRIPS
  - ADL
  - PDDL
  - Temporally-extended actions: PDDL 2.2
  - Continuous dynamics: Opt/PDDL+
- Hierarchical Task networks
  - ACT (Sipe)
  - O-Plan language
  - SHOP2 language
- Constraint-based modeling
  - DDL

Planning Methods

- First principles/precondition-chaining planners
  - Totally-ordered planning
    - Forward, heuristic search
    - Plan graph
    - SAT
    - BDDs
  - Partial-ordered (non-linear) planning
- HTN/decomposition planners
  - Forward
  - Top-down refinement
- Constraint-based planners

STRIPS Family

- There is a family of languages going back to STRIPS, all based on the state/event model of system dynamics.
- STRIPS rules are propositional, and include:
  - A conjunction of preconditions
  - An *add list* of propositions to be made true (added to the state)
  - A *delete list* of propositions to be made false (removed from the state)

- Subsequent development on ADL and several generations of PDDL have extended this representation to include:
  - Quantification in preconditions and effects
  - Context-dependent effects
  - Logical connectives in preconditions (AND, OR)
  - Derived predicates (added back)

PDDL 2.2

```
(:action open
   :parameters (?x - DEVICE)
   :precondition (and (not (= ?x earth))
                 (closed ?x)
                 (forall (?b - DEVICE) (not (affected ?b))))
   :effect (not (closed ?x)))

 (:action close
   :parameters  (?x - DEVICE)
   :precondition (and (not (= ?x earth))
                 (not (closed ?x))
                 (forall (?b - DEVICE) (not (affected ?b))))
   :effect (closed ?x))

(:derived (affected ?x - DEVICE)
      (and (breaker ?x)
          (exists (?sx - SIDE) (unsafe ?x ?sx))))
```


Metric Information

Starting with PDDL 2.1 and IPC-3, PDDL included:

- "Durative" actions
- Metric preconditions and effects (e.g. tool usage)

McDermott's "Opt" language uses a point-based temporal model to represent continuous processes (also proposed in PDDL+).

The concept (as opposed to an efficient implementation) goes back to [Hendrix-73].

Durative Actions and Timed Literals

```
(:durative-action move
 :parameters
     (?a - airplane ?t - airplanetype ?d1 ?d2 - direction
      ?s1 ?s2  - segment)
 :duration (= ?duration (/ (length ?s1) 30))
 :condition
     (and (over all (has-type ?a ?t))
          (over all (is-moving ?a))
          (at start (facing ?a ?d1))
          (over all (not  (exists (?a1 - airplane)
                            (and (not (= ?a1 ?a))
                                 (blocked ?s2 ?a1)))))
                …     )
 :effect
     (and (at start (occupied ?s2)) (at start (blocked ?s2 ?a))
          (at end (not (occupied ?s1)))
          (at end (when   (not (is-blocked ?s1 ?t ?s2 ?d2))
                          (not (blocked ?s1 ?a))))
                …
      )
```

```
(at 34 (blocked seg_rwtw2_0_10 dummy_landing_airplane))
```

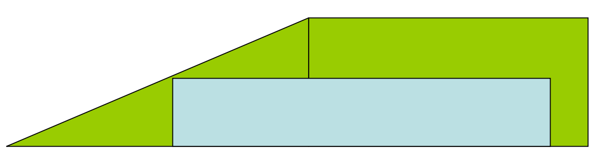

Continuous Change

The general idea is to specify endpoints and a function for how a continuous value changes between those endpoints.

In the simplest case, duration and the rate of change are both fixed.

Or duration may be a function of a continuous value

In more general models, as we will see, one or more of the variables involved (start, end, duration, rate, volume) may be solved for, rather than specified.



Compilation and Preprocessing

So, PDDL and other state-event models can be used to represent metric information, resources, metric time, context-dependent effects, exogenous events, overlapping actions…

All of these extensions can be compiled down into the original STRIPS rules.

"Ground" instantiates PDDL2.2 planning domains, based on the precompiler of the MIPS system (http://ipc.icaps-conference.org/)

- Instantiates all parameters
- Compiles out quantification, disjunction, context-dependent effects, derived predicates.
- Removes constants
- Does symmetry detection

Forward Heuristic Search

Planners using *Forward Heuristic Search* construct a plan by adding actions to the end of a sequence, based on estimates of how close to the goal the resulting state will be.

Sources of these distance estimates include:

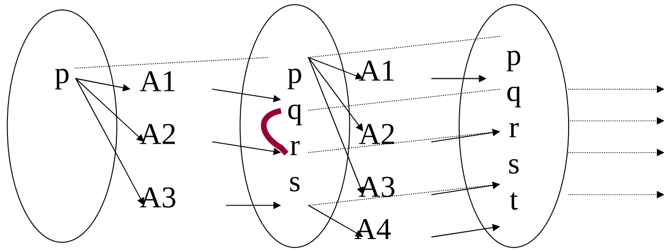
- Various forms of (partially) relaxed plan graphs
- Regression over a plan suffix
- Explicitly coded heuristics

Plan construction then proceeds using some form of search.

Empirically, the main strength of these approaches is that they are fast. Anecdotally, an additional advantage is that generating effective heuristics is (relatively) easy.

These algorithms are highly tuned for goals of achievement, rather than maintenance, and for state/event models of the world. They are not so good for resources, though metric information can be represented.

Plan Graph



Cache interactions between actions and propositions, extending the graph until you reach a node in which the goal proposition is satisfied.

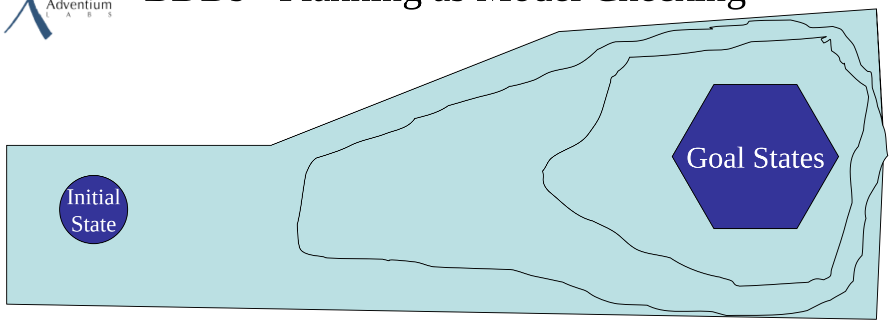
Many, many refinements

- Searching backward over the graph
- Using relaxed versions for distance heuristics
- Adding metric information (cost, time).

Satisfiability (SAT)

- Key insight: There are very high-performance SAT solvers available, so reformulate planning problem as SAT.
  - As with Graphplan, develop (compile) an ancillary data structure that captures possible plans.
  - In this case the data structure is a propositional satisfiability (PSAT) problem.
- SAT problem captures:
  - Sequence of states as sequences of propositional variables for fluents.
  - Actions as constraints on state->state pairs.
    - Preconditions and postconditions must be satisfied.
  - Persistence as constraint on state->state pairs.
    - Fluents persist unless changed by actions.
  - Relationships between actions (mutual exclusion).
- Search for plans within a length bound. Can iteratively relax the length bound.
- Much work on compilation techniques to provide most efficiently-solvable SAT problems.

BDDs - Planning as Model Checking

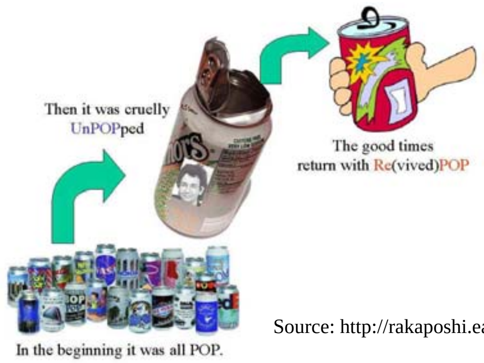


- Key notion: Binary Decision Diagrams permit efficient representation of sets of states and relationships between them.
- Action model provides a reachability relationship between states (or sets of states).
- The *pre-image* of a state set, *S*, is the set of states from which *S* can be reached in one step.
- Finding a plan:
  - Iterate the pre-image operation (find its fixpoint or closure).
  - If the initial state lies in the closure, then we have a plan.
- Can be adapted to more expressive kinds of planning:
  - Extended goals (not just achievement)
  - Planning under uncertainty, and planning in dynamic domains (controller synthesis).

Partial-order Planners (POP)

- Central idea: With knowledge about the weakest preconditions and postconditions for an action (wrt a goal), we can determine whether a plan is well-formed without requiring that it be a simple linear sequence of actions.
- This yields a plan-space planner, where nodes in the search are constrained sets of plans.
- Algorithm (generally) works as follows:
  - Select a plan flaw
    - Precondition not achieved or
    - Possible clobbering
  - Resolve flaw:
    - Introduce new action to achieve precondition and order it before precondition
    - Avoid clobbering by adding ordering or variable-binding constraints
- Notes:
  - A single POP "plan" is actually a compressed representation of a set of fully-ordered plans.*
  - This can yield substantial search space savings in some domains.

POP and UnPOP



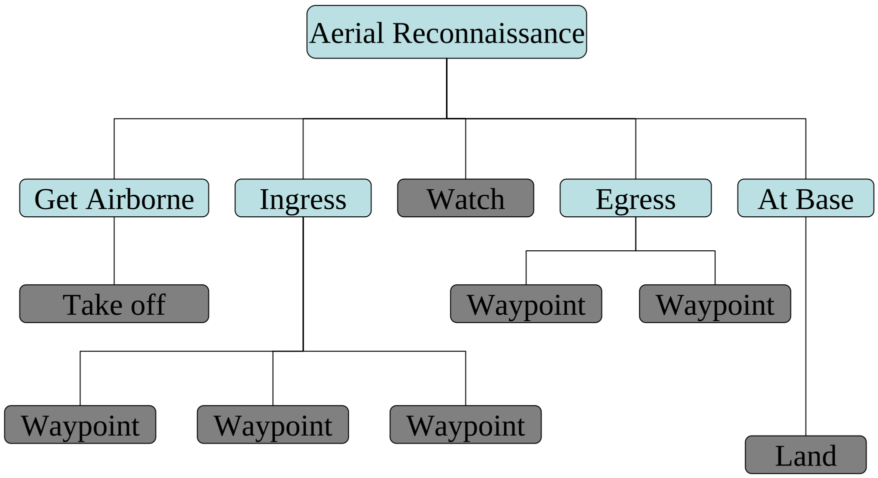
Source: http://rakaposhi.eas.asu.edu/repop.html

- POPs once dominated, but total-order planners have made a big comeback because:
  - It’s easier to handle complex domain mechanics if all you have to do is simple projection/simulation.
  - It has seemed easier to provide informative heuristics for linear planners. Easier to evaluate states than partial plans.

Decomposition Planning: HTNs

- HTNs provide a radically different model for planning:
  - Start with top-level, abstract task, and *decompose* it to subtasks, that are increasingly concrete.
  - Terminate when there is a decomposition tree (or DAG) whose leaves are *primitive* (executable) actions.
- HTN planners can be *more expressive* than first-principles planners, because HTNs can capture constraints on the *trajectory* of a plan, instead of just its endpoints.
- HTNs can provide search guidance to planning by encoding known good solution methods.
- HTNs have been used to provide plan maintenance tools, possibly more useful than plan generation tools.
- Fairly recently HTN planning has become better understood, through work by Nau, Hendler, and Erol.

HTN Planning Example



Hierarchical Task Networks

- We focus on SHOP2's language, because it's readily available.
- Tasks may be *methods* or *operators* (primitives).
- Tasks have preconditions, for application (operators) or to limit conditions for applying decompositions.
- Operators have add and delete lists.
- Methods have task networks, ordered or unordered.

```
;; helicopter delivers a pkg…
(:operator
  ;; task spec
  (!deliver ?uav ?pkg ?loc)
   ;; preconditions
   ((loc ?uav ?loc)(time ?t))
   ;; delete
   ((airborne ?uav))
   ;;add
   ((delivered ?pkg ?t)))
(:method
  ;; task spec
  (deliver2 ?pkg1 ?pkg2 ?loc)
   ;; preconditions
   ((loc ?uav ?loc))
   ;; task network
   (:unordered
      (!deliver ?uav ?pkg1 ?loc)
      (!deliver ?uav ?pkg2 ?loc))
```

Planning as a Constraint Satisfaction Problem

CSPs are specified as:
- A set *V* of *variables*
- A set *C* of *constraints*, each constraint a relation specifying tuples of permissible values for some subset of *V*.

The objective is to find a feasible (alt., optimal) complete assignment for *V*, consistent with *C*.

Constraint-based Planners

CSP Variables:
- Activity start and end times
- Activity resource assignments

Constraints:
- Temporal constraints (duration, ordering, release times, deadlines, …)
- Resource constraints (permissible assignments, usage requirements, state information, ...)
- System dynamics (preconditions/effects, allowable state changes, …)

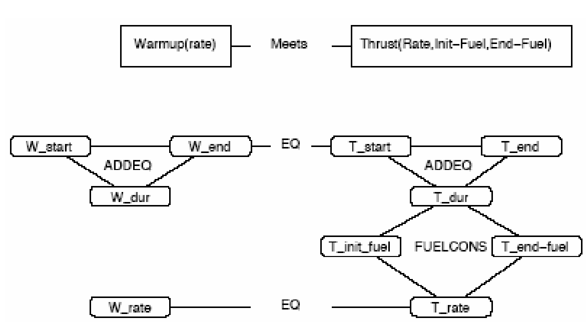
Search over:
- Activity generation
- Resource assignments
- Activity orderings, start and end times

Advantages to a CSP approach

- Flexible, declarative representation.
- Lots of previous and current work on solution methods.
- General techniques:
  - Static structural analysis
  - Propagation
  - Search
- Requirements and scheduling decisions can be represented as constraints.

 27

Example: HSTS/Europa



- Use tokens and *compatibilities*
- Constraints on time, resources, continuous dynamics

A significant issue for many constraint-based planners is finding effective domain-independent solution methods and heuristics.

 28

Domain Characteristics

- Autonomy
- Crowded, resource-bounded schedules
- Over-subscribed schedules with multiple stake-holders
- Coordinated multi-platform operations
- Multi-step operations
- Complex system dynamics
- Unpredictability (uncertain execution)
- Distributed operations
- Real-time requirements
- Model drift
- Replanning ("enterprise" planning)

Interesting Combinations

- **Cyber-security** - complex, uncertain execution, model drift
- **Airport surface movement** - resource-bounded, real-time, uncertain execution
- **UAV** - autonomous, multi-step, uncertain execution
- **Process Industry Manufacturing** - resource-bounded, oversubscribed, complex dynamics
- **NASA domains from previous study:**
  - **Manned space/large telescopes**: resource-bounded, oversubscribed, costly replanning
  - **Rovers/outer-planet tours:** autonomous, multi-step, uncertain execution, real-time, model drift
  - **Spacecraft constellations:** Coordinated, complex dynamics, real-time

Mapping Application Properties to Required Capabilities

- Autonomy
  - Planning and control
- Crowded, resource-bounded schedules
  - Scheduling, temporal planning, planning with resources
- Over-subscribed schedules with multiple stake-holders
  - Optimization, explanation, negotiation, scheduling
- Coordinated multi-platform operations
  - Coordinated (planning and) execution (control)
- Multi-step operations
  - Classical/constraint-based/HTN planning, procedural executives
- Complex system dynamics
  - Complex continuous/hybrid models, high-level control

31

More Capabilities

- Uncertain execution
  - Contingent planning, reactive planning, rapid replanning, information-gathering plans, stochastic/decision-theoretic planning, …
- Distributed operations
  - Collaborative planning, negotiation
- Real-time
  - Real-time planning (not “fast”), generating real-time responses
- Model drift
  - Model updating (both continuous and discrete), acting to gain information
- Cost on replanning
  - Local plan repair, negotiation.

32

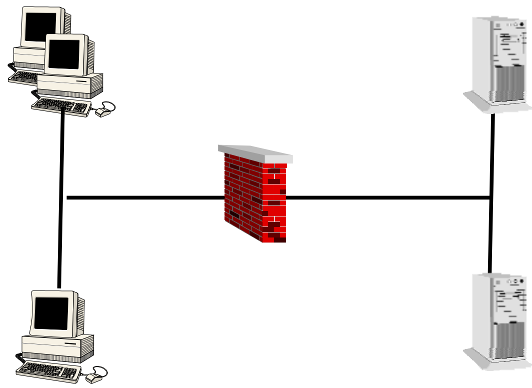
Engineering the Problem Away

- Sometimes you don't *need* a planner!
- Planning as programming
  - Caching a response
  - Planning using programming constructs

The Domains

- Vulnerability analysis for cyber security
- Resource-intensive planner (IPC airport example)
- UAV
- Process-industry manufacturing

End-users Mail Server



Sys Admin
- Password protected account
- Manages user accesses

COI Web Server
- SSL with fixed passwords
- ACLs

The Problem

Finding attack vulnerabilities

For example:

1. Attacker sends an email message, spoofed to be from a colleague, with a new screensaver as an attachment.
2. Attachment is an executable that enables remote login, and captures and relays the users password.
3. Attacker logs into the machine and executes a buffer overflow attack, gaining root (admin) privileges.
4. …

Why Use a Planner?

- Network and system scale, complexity, and dynamism
- Attackers are stealthy
- Many steps in any given attack may be legitimate.
- Some exploits involve actions taken outside the network.
- Some exploits are impossible or expensive to detect.
- Limited supply of experts

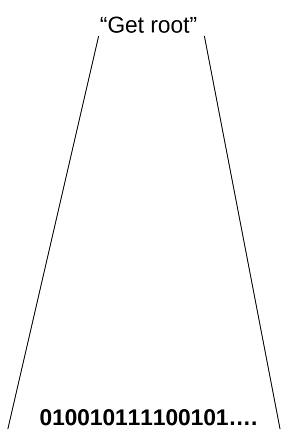
Why Classical Planning?

- Domain characteristics
  - Propositional representation fits well
  - Time is not very important
  - Nor are resources
- Style of inference
  - No *a priori* assumptions about operator sequences
  - Richer state representation
  - Rigorous determination of infeasibility

Components of the Planning Model

- Adversary characteristics
  - Risk tolerance
  - Available resources
- Attack methods
  - Operators
- Network (domain) model
- Adversary objectives
  - Goals

Modeling the Domain: Level of Abstraction



Finding a middle ground:

- Preconditions, actions, effects relevant to user
  - Sending email
  - Logging in
  - Creating/modifying files

- Useful representation of preconditions and effects

Examples: Facts

- (insider bob)
- (in_room bob bobs_office)
- (can_unlock key1 lock1)
- (knows bob root_password)
- (accessible s_iexplore sherpa)
- (can_read_email ms_outlook)
- (trusts_instructions greg adam)

Examples: Goals

```
(:goal (knows bob secret_info))
(:metric minimize (detection_risk))

(and (knows bob secret_info)
      (<= (detection_risk) 5))
```

Examples: Actions

```
(action DMS_ADD_GROUP_ALLOW
    :parameters (?admin - c_human
                  ?chost - c_host
                  ?shost - c_host
                  ?doc - c_file
                  ?gid - c_gid)
    :precondition
        (and (nes_admin_connected ?chost ?shost)
             (at_host ?admin ?chost)
             (insider ?admin)
    :effect (and (dmsacl_read ?doc ?gid)))
```


A Plan

0 : ADAM sits down at BIGFOOT
1 : ADAM enters ADAM_UID as user name for login on host BIGFOOT
2 : ADAM enters password ADAM_PWD for login at host BIGFOOT
3 : Shell B_WEXPLORE is launched on host BIGFOOT for user ADAM_UID
4 : Program WEXPLORER on host BIGFOOT forks a child process
5 : Contents of file B_IEXPLORE begin executing as uid ADAM_UID on host BIGFOOT
6 : BOB sits down at YETI
7 : BOB enters BOB_UID as user name for login on host YETI
8 : BOB enters password BOB_PWD for login at host YETI
9 : Shell Y_WEXPLORE is launched on host YETI for user BOB_UID
10 : Program WEXPLORER on host YETI forks a child process
11 : Contents of file Y_ETHEREAL begin executing as uid BOB_UID on host YETI
12 : ETHEREAL starts sniffing the networks on YETI
13 : ADAM logs onto dms admin server EVEREST from BIGFOOT
14 : BOB reads the sniffer thus learning NES_ADMIN_PASS

Plan, Continued

15 : Program WEXPLORER on host YETI forks a child process
16 : Contents of file Y_IEXPLORE begin executing as uid BOB_UID on host YETI
17 : BOB logs onto dms admin server EVEREST from YETI
18 : DMS session DMSS1 has begun
19 : BOB begins a DMS session on YETI
20 : Connect DMS session DMSS1 to server NES on EVEREST
21 : A route from YETI to DMS server EVEREST exists
22 : BOB enters password BOB_DMS_PWD for the DMS session.
23 : Authenticate BOB_UID in dms session DMSS1 with EVEREST using BOB_DMS_PWD
24 : BOB adds an acl to allow read access of E_SECRET_DOC to the EAST_GID group
25 : BOB begins a DMS request at YETI in session DMSS1
26 : Document E_SECRET_DOC is requested in session DMSS1
27 : Document E_SECRET_DOC is sent and displayed on YETI in session DMSS1
28 : BOB reads E_SECRET_DOC and learns SECRET_INFO

Pragmatic Issues

- Performance (esp. memory consumption)
  - Rewriting the model to avoid “hard actions”
  - Rewriting to minimize the size of the propositional expansion
- Representing processes (e.g., composing and sending email).
- Entities that are created or destroyed
- Derived predicates
- Maintaining large domain models

Rewriting the model to avoid “hard actions”

Metric-FF compiles away much of PDDL’s expressive power:

- Quantification is expanded on the domain.
- Conjunction and disjunction are rewritten.
- Context-dependent effects are not removed.

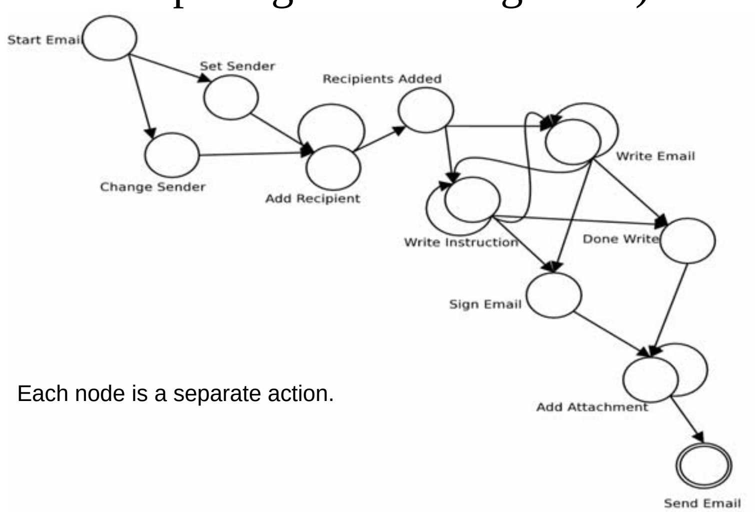
- “Hard actions” appear to be those whose preconditions are not in DNF. So, we can rewrite

  (and foo (or bar baz)

  to be

  (or (and foo bar) (and foo baz))

Representing processes (e.g., composing and sending email).



Each node is a separate action.

Created Entities

In a cyber domain, there are numerous "handles" whose specific value is unimportant, many of which are created on the fly.

- Process IDs
- File IDs
- Sockets, sessions, etc…

A propositional planner is not smart enough to know when trying a different ID might help, and when it won't.

Modular Domains in PDDL

PDDL input consists of:

- A *domain model*, specifying object types, predicates, and actions
- A *problem statement*, specifying all objects, the initial state and a goal.

A more natural way to specify a complex domain is in separate modules, but this aggregation is inconsistent with the PDDL spec.

Cyber-Security Models in HTN

QuickTime™ and a
TIFF (LZW) decompressor
are needed to see this picture.

Airport Surface Movement Planning

- Problem: From an initial configuration, move inbound planes from runway to gates, and move outbound planes from gates to runways (and out).
- Features of the problem:
  - Complex dynamics
    - Planes moving in two-space
    - Need to avoid collisions (deconfliction)
      - Also need to avoid other aircrafts' wakes
    - Planes are heterogeneous (affects how they use space)
  - Temporal planning
  - Resources (areas)
  - Optimization
- Some simplifications
  - Discrete space: taxiways are chopped into *segments*

IPC Modeling Approach

- Chosen to make problem available to as many planners as possible (best solution approach was not the primary criterion!).
- Four versions were made:
  - Non-temporal
    - STRIPS [won't discuss] and ADL variants
    - Goals are simply to get aircraft, parked, airborne, etc.
  - Temporal
  - Temporal time-windows
    - Allows us to specify when runways will become blocked by scheduled landings
  - Temporal time-windows compiled – avoid time-windows by compiling them into operators. [we won't discuss these]

Domain Model

- Operators
  - Pushback - push back from the gate
  - Startup
  - Move – Move between segments on the ground
  - Takeoff
  - Park
- Predicates
  - Aircraft state
    - has-type
    - at-segment
    - facing
    - Mode: is-moving, is-pushing; **sink states:** is-parked, airborne
  - Topology
    - (can-move ?s1 ?s2 - segment ?d - direction)
    - (can-pushback ?s1 ?s2 - segment ?d - direction)
    - (move-dir ?s1 ?s2 - segment ?d - direction)
    - (move-back-dir ?s1 ?s2 - segment ?d - direction)
    - (is-blocked ?s1 - segment ?t - airplanetype ?s2 - segment ?d - direction)
    - (is-start-runway ?s - segment ?d - direction)
  - Airport state
    - (occupied ?s - segment)
    - (blocked ?s - segment ?a - airplane)

Compiling Dynamics into Effects

- Deconfliction:
  - Aircraft block segments they occupy
  - Wake effect: If an aircraft's engines are running, they block segments behind them
    - Range of wake effect depends on type of aircraft
    - Effect is compiled into `is-blocked`/4 relationship
- Postconditions of `(move ?a ?t ?dir1 ?seg1 ?dir2 ?seg2)`
  - Update aircraft state:
    - `(at-segment ?a ?seg2)`
    - `(not (at-segment ?a ?seg1))`
    - `(when (not (= ?d1 ?d2)) (not (facing ?a ?d1)))`
  - Blocking
    - `(blocked ?s2 ?a)`
    - ```
      (forall (?s - segment)
          (when (is-blocked ?s ?t ?s2 ?d2)
                (blocked ?s ?a)))
      ```
    - There must be similar rules for deleting blocking predicates
    - These rules must be replicated, *mutatis mutandis*, in pushback and parking operators
    - Unpleasant software engineering practice…


```
(:action move
 :parameters
     (?a - airplane ?t - airplanetype ?d1 - direction ?s1 ?s2  - segment ?d2 - direction)
 :precondition
     (and        (has-type ?a ?t)
                 (is-moving ?a)
                 (not (= ?s1 ?s2))
                 (facing ?a ?d1)
                 (can-move ?s1 ?s2 ?d1)
                 (move-dir ?s1 ?s2 ?d2)
                 (at-segment ?a ?s1)
                 (not    (exists (?a1 - airplane)        (and    (not (= ?a1 ?a))
                                                                 (blocked ?s2 ?a1))))
                 (forall (?s - segment)  (imply  (and    (is-blocked ?s ?t ?s2 ?d2)
                                                         (not (= ?s ?s1)))
                                         (not (occupied ?s)))))
 :effect
     (and
                 (occupied ?s2)
                 (blocked ?s2 ?a)
                 (not (occupied ?s1))
                 (when   (not (is-blocked ?s1 ?t ?s2 ?d2))
                         (not (blocked ?s1 ?a)))
                 (when   (not (= ?d1 ?d2))
                         (not (facing ?a ?d1)))
                 (not (at-segment ?a ?s1))
                 (forall (?s - segment)  (when   (is-blocked ?s ?t ?s2 ?d2)
                                         (blocked ?s ?a)
                                 ))
                 (forall (?s - segment)  (when   (and    (is-blocked ?s ?t ?s1 ?d1)
                                                 (not (= ?s ?s2))
                                                 (not (is-blocked ?s ?t ?s2 ?d2))
                                         )
                                         (not (blocked ?s ?a))
                                 ))
                 (at-segment ?a ?s2)
                 (when   (not (= ?d1 ?d2))
                         (facing ?a ?d2))))
```
```

Derived Rule Alternative

```
(:derived (blocked ?seg ?a)
    (or (at-segment ?a ?seg)
        (exists (?t - airplanetype)
         (exists (?s - segment)
           (exists (?d - direction)
            (and (has-type ?a ?t)
                 (at-segment ?a ?s)
                 (is-moving ?a)
                 (facing ?a ?d)
                 (is-blocked ?seg ?t ?s ?d)))))))
```

- Seems cleaner, but many planners don't handle derived predicates well…

Adding Time: PDDL Mechanisms

- Durative actions
  - PDDL 2.2
  - Actions have durations
  - Actions have *conditions* instead of preconditions
    - at start
    - at end
    - over all
  - Actions have effects at different times
    - at start
    - at end
- Timed initial predicates
  - Propositions that are scheduled to become true at a particular time in the future (wrt the start of the problem)

Adding Time: Examples

```
(:durative-action move
 :parameters
     (?a - airplane ?t - airplanetype ?d1 ?d2 - direction
      ?s1 ?s2  - segment)
 :duration (= ?duration (/ (length ?s1) 30))
 :condition
     (and (over all (has-type ?a ?t))
          (over all (is-moving ?a))
          (at start (facing ?a ?d1))
          (over all (not  (exists (?a1 - airplane)
                             (and (not (= ?a1 ?a))
                                   (blocked ?s2 ?a1)))))
                  …      )
 :effect
     (and (at start (occupied ?s2)) (at start (blocked ?s2 ?a))
          (at end (not (occupied ?s1)))
          (at end (when   (not (is-blocked ?s1 ?t ?s2 ?d2))
                          (not (blocked ?s1 ?a))))
                  …
      )
```

```
(at 34 (blocked seg_rwtw2_0_10 dummy_landing_airplane))
```



```
(:durative-action move
 :parameters
     (?a - airplane ?t - airplanetype ?d1 - direction ?s1 ?s2  - segment ?d2 - direction)
 :duration
     (= ?duration (/ (length ?s1) 30))
 :condition
     (and       (over all (has-type ?a ?t))
                (over all (is-moving ?a))
                (over all (not (= ?s1 ?s2)))
                (at start (facing ?a ?d1))
                (over all (can-move ?s1 ?s2 ?d1))
                (over all (move-dir ?s1 ?s2 ?d2))
                (at start (at-segment ?a ?s1))
                (over all (not  (exists (?a1 - airplane)
                                  (and    (not (= ?a1 ?a))
                                          (blocked ?s2 ?a1)))))
                (over all (forall (?s - segment)
                            (imply  (and    (is-blocked ?s ?t ?s2 ?d2)
                                            (not (= ?s ?s1)))
                                            (not (occupied ?s))))))
:effect
     (and
                (at start (occupied ?s2))
                (at start (blocked ?s2 ?a))
                (at end (not (occupied ?s1)))
                (at end (when   (not (is-blocked ?s1 ?t ?s2 ?d2))
                                (not (blocked ?s1 ?a))))
                (at end (when   (not (= ?d1 ?d2))
                                (not (facing ?a ?d1))))
                (at end (not (at-segment ?a ?s1)))
                (at end         (forall (?s - segment)  (when   (is-blocked ?s ?t ?s2 ?d2)
                                                                (blocked ?s ?a))))
                (at end (forall (?s - segment)
                          (when (and (is-blocked ?s ?t ?s1 ?d1)
                                     (not (= ?s ?s2))
                                     (not (is-blocked ?s ?t ?s2 ?d2)))
                                (not (blocked ?s ?a)))))
                (at end (at-segment ?a ?s2))
                (at end (when   (not (= ?d1 ?d2))
                                  (facing ?a ?d2)))))
```

Challenge: Optimization

- Optimization criterion for application: minimize the summed travel time for all aircraft.
  If aircraft 1 arrives at the airport first, it should *not* reach the gate last!
  - This may itself be a simplification!
- Not available to non-temporal versions – plan length is the best we can do.
- Problem for even temporal PDDL in the context of the IPC
  - Requires access to a distinguished current-time fluent.
  - Durative actions would advance current-time fluent.
  - Whenever a durative action terminates:
    - Update current time
    - Incur cost for each aircraft that is moving
    - Cost proportional to advance in time
- Difficult to do for competition organizers, because it requires access to the semantics of the planner.
- Feasible for an application-builder, esp. one with access to planner source.
  - We (RPG) used method like this in UAV application.

UAV Reconnaissance Planning

- Core problem: Fly an autonomous vehicle to a target region to carry out some sensing, and then return to base.
- Based on SIFT experience building a planner for small UAVs in conjunction with Geneva Aerospace (airframer).
- Characteristics:
  - Complex dynamics: flight in three-space (procedural attachment)
  - Autonomy
  - Planning and control
  - Temporal planning
  - Multi-step operations
  - Non-achievement goals: go out, scan some location, and return to base before the end of the window
- Extensions of the problem:
  - *Extended* surveillance over time
  - Multiple UAVs with availability windows
  - Heterogeneous UAVs
  - Other UAV tasks (e.g., deliver packages)
  - Fuel
  - Airspace deconfliction
  - Optimization

Features of the Model

- State of the UAVs
  - Location in three-space (lat, lon, altitude)
  - Heading
  - Pitch (*ignored*)
  - Speed (little variance)
  - Maneuver (*implicit*)
  - Scanning
- State of the environment
  - Digital map data (DTED) (*implicit*)
  - Location of points of interest
- Other
  - Windows of availability
  - Time

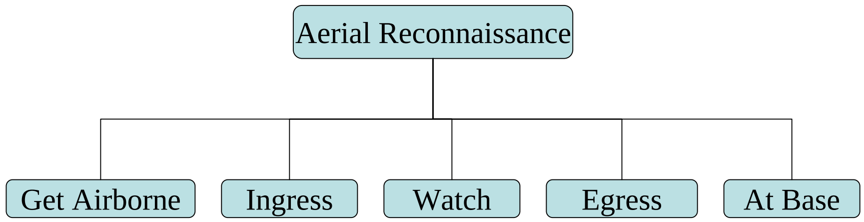
Simple Approach

- For single UAV, represent state of vehicle in fluents
- Use discretized representation of four-D space (x,y,z,ϕ)
  - Ignore pitch, assuming only gentle altitude climb and dive
  - Mostly assume nominal cruising speed
- State-encode goal achievement:
  - Start-scanning operator will record that scanning has happened, allowing our trajectory constraint to be captured as an achievement goal:

```
(and (scanned alpha)
     (forall (uav ?u) (at-base ?u)))
```

  - Still potentially pointless search trying to get UAVs home and *then* scan target!
- Such a model could be handled by a conventional forward planner.
- Problems:
  - How to ensure that the scanning occurs on time?
  - How to avoid getting lost in the very large search space for navigation?

HTN Approach



- First step: Provide top-level operator that will constrain the structure of the plan
- Avoids need for state-encoding goal
- Avoids plans that try to go home and then scan target

Route Planning: Procedural Attachment

- In our application, handled by *procedural attachment:*
  - Preconditions for methods call out to a special-purpose A* search algorithm to find routes.
  - Advantages
    - Allows the A* search to have special-purpose heuristic.
    - Allows A* search data structures to be optimized (search states needn't carry around full plan state information, can use special-purpose location structures).
  - Disadvantages
    - Hard to combine different optimization criteria.
    - If planner rejects a path, may have to backtrack blindly into path generation (or just fail).
    - Deconfliction isn't easily handled.

Adding Time

- UAVs have time windows of availability
  - Abstract representation of fuel constraints, etc.
- Goal is to reach the target area and watch it over some window of time.
  - E.g., watch area ALPHA for 15 minutes, starting a half an hour from now.
- Durative actions are easy in HTNs:
  - Durative action *foo* -> `foo-start` and `foo-end` operator pair
  - Distinguished time literals

Adding Time: Example

```
(:method (do-hover ?uav ?newloc ?hover-duration ?start ?end) ()
   (:ordered
       (:task :immediate !hover-start ?uav ?newloc ?hover-duration ?start)
       (!hover-end ?uav ?end)))

(:operator (!hover-start ?uav ?loc ?speed ?time)
           ;; preconditions
           ((airspeed ?uav ?old-airspeed)
            (location ?uav ?old-loc)
            (yaw ?uav ?old-yaw)
            (time ?time))
           ;; delete
           ((airspeed ?uav ?old-airspeed)
            (location ?uav ?old-loc)
            (next-waypoint ?uav ?index)
            (yaw ?uav ?old-yaw))
           ;; add
           ((location ?uav ?loc)
            (yaw ?uav :undefined)
            (airspeed ?uav 0)))

(:operator (!hover-end ?uav ?newtime)
           ;; preconditions
           ((time ?time)
            (check-constraint (<= ?time ?newtime)))
           ((time ?time)) ;; delete
           ((time ?newtime))) ;; add
```

Reasoning "From the Side"

- To solve this planning problem we must be able to allocate resources (notably time) over the trajectory of the plan.
  - Time to reach the target
  - Time to scan the target
  - Time to return to base
- The problem is even harder when it takes multiple missions (i.e., multiple UAVs) to cover a single target time window.
- This kind of reasoning is very difficult for a forward planner to do.
  - If a forward state-space planner fails to satisfy these constraints, it has little search guidance for plan repair.
- A forward HTN like SHOP2 can do this a little:
  - Use preconditions of top-level operator to allocate blocks of time to the three phases of the plan.
  - But the system does not provide direct support for this kind of abstraction.
  - When the predictions at the top level are violated, it isn't possible to backtrack and repair them.
- A classical (non-forward) HTN planner might do better.

Optimization

- We use optimization as an aid in overconstrained problems.
  - What if we cannot do what the user wants?
    - We don't have aircraft available for long enough, or
    - We can't get a UAV to the target fast enough, etc.
  - Simply saying "no," is not very helpful!
  - We try to find the most coverage possible in these situations.
- But "off the shelf" optimization techniques aren't appropriate
  - We're not optimizing makespan!
  - We're not minimizing total cost of the operators.
- Cost function: incur $1/minute of goal scanning window when not scanning.
- As a side effect of advancing time, we compute the cost for a time period.
- This is not well supported by any existing modeling framework; we have hacked it into our planning algorithm instead.
- In general, it would be desirable to have cost functions that can be computed over state trajectories, rather than action sequences.
  - Yes, they are *expressively* equivalent, but not *conveniently* equivalent, as a matter of engineering.

How Did It Work?

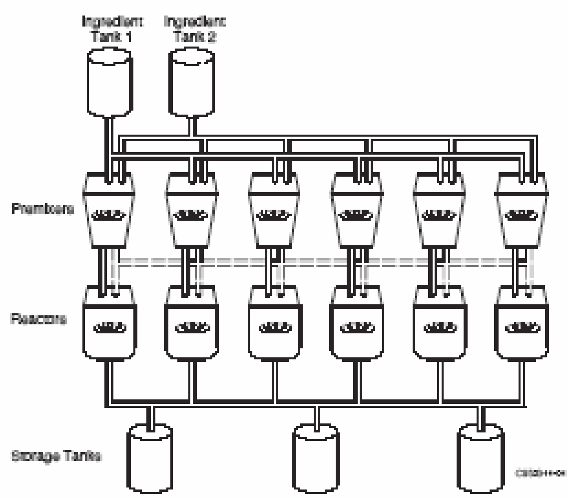
- Although a research project, this was a UAV control project, and *not* a planning project.
- Good features
  - The use of planning provided very rapid addition of new capabilities as they were added (e.g., as we got new vehicles and new control capabilities). New capabilities could be added with very few lines of code.
  - Provides a nice user interface to allow non-experts to tell these vehicles what to do.
    - HTN was very helpful.
    - Flying a UAV from a Ground Control Station is very difficult.
    - Control software is not at all user-friendly. E.g., control to airspeed….
- Bad features
  - It's still difficult to build planning domains.
  - Debugging them is *really* hard.
  - Handling overconstrained situations is still difficult. One needs to be able to explain planner failures, and guide users to reformulate their goals.
    - No planning algorithm I (RPG) know of is very good at this, or even at providing the raw materials for this.

Process Industry Domains

Types of Models

- Aggregate -- summed resource consumption and production over a period, or over multiple periods, out to some time horizon.
- Activity -- tracking activities (maintenance activities, sequencing batched production, dock usage, pipeline windows, etc.), resource usage (units/tanks/pipelines involved), inventory (feedstock and rundown tank levels), and timing requirements (delivery dates, crude shipment arrivals).
- Dynamic -- predicting and controlling transient responses in plant dynamics.
- Setpoint -- predicting and controlling steady-state plant operations.

Batch Processes

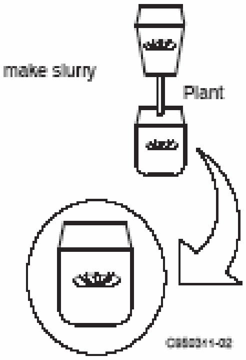


Recipes

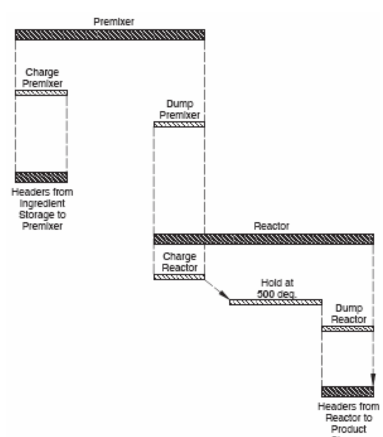
General Recipe
- Amounts: Ingredient 1: 500 gal; Ingredient 2: 1000 gal; catalyst: 10 lb; produces 1200 gal
- Mix ingredients 1 and 2
- Add catalyst
- Heat for 2 hr, agitating

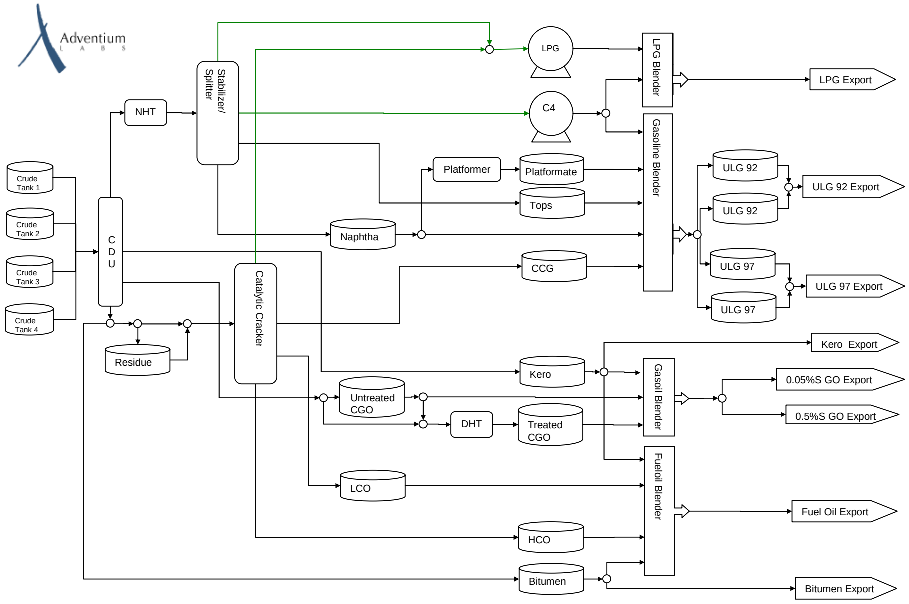
Site Recipe
- Mix ingredients 1 and 2 in Premixer to make slurry
- Transfer slurry to Reactor
- Add catalyst to Reactor
- Heat Reactor while agitating

Unit Recipe
- Charge from Premixer
- Charge from catalyst conveyor
- Start agitator
- Heat to 500°F
- Hold at 500°F for 2 hr
- Stop agitator
- Dump



Constraints





Refinery Planning and Scheduling

Planning -- generating activities to support goals:

- Accept receipts of crude oil (make sure there is a place to put them)
- Satisfy product shipments (make sure material is available, generate shipment activities)

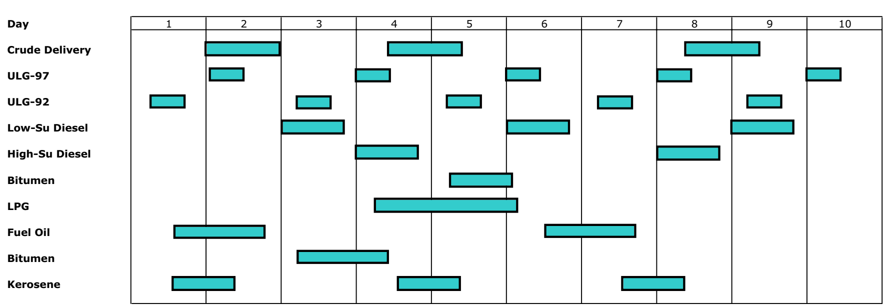
Scheduling

- Constraints on tank volumes, flow rates, unit operating parameters
- Mutex constraints on CDU mode, gasoline blender usage, filling and emptying certain tanks (“standing gauge”).

Problem statement

- Specified
  - Crude deliveries
  - Product liftings (shipments)
  - Initial tank contents, volume and qualities
- Constrained
  - Product specifications (in terms of qualities)
  - Tank volume min/max
  - Unit constraints (operating ranges)
  - Material balances (Hydrogen, RFG)
- Objective function
  - Product “giveaway”
  - Inventory value, by component
  - Ending inventory targets

Input “Constraints”



Problem Features

Continuous dynamics:
- Continuous time
- Metric resources (power, weight, process heat)
- Consumable resources (energy, volume/capacity, fuel)
- Continuous action effects
- “Rate” is a free variable
- Objective functions over continuous parameters.

Actions:
- Concurrent/asynchronous actions
- Action choice
- “State” (action sequences, state resources, action preconditions)

Solution Must Specify

- Material movements
  - Crude charges
  - Shipments
  - Blends
- Unit modes
  - Crude Distillation Unit
  - Distillate Hydrotreater (desulphurizer)
- Unit controls
  - Split fractions (e.g., CDU)
  - Conversion (e.g., desulphurization, platformer)

Model Elements

- Movements
  - Shipments
  - Rundowns
  - Blends
- Tanks
- Pumps, Berths, Transfer lines
- Paths
- Ships
- Properties
- Pipelines

Moving Material

- Simple movements:
  - Source tank
  - Destination tank
  - Volume
  - Bounds on rate (meaning duration).

- Multiple *concurrent* sources and destinations can be handled using multiple resource requirements (that's where volume effects are encoded), or by creating multiple activities (e.g., inline blends)
- Asynchronous multiple movements, as well
- Matching on time *and rate*

Internal pipelines

- Pipeline volume must be modelled explicitly (not like shipping and receiving).
- Model slugs of material with associated volume, etc., plus a *position* in the pipeline.
- Material movement in the pipeline requires both input and output flows.
- Rate is independent of individual movements
- It seems to simplify things to break up movements into volumes <= pipeline volume.

IPC-4 Pipesworld

- Fixed batch sizes
- Material does not enter or leave the system
- “Cycles” in pipeline movement
- Tank volume constraints, in some versions

Tanks

- Two kinds of resources
  - consumable (volume)
  - unary (busy)

Ships

Both a resource *and* an activity

- Activity: ship's presence
- Resource: ship's hold(s)
- One interesting complication is that the contents of the hold are only available during a specified time window.
- There are several ways to model this:
  - Constrain the movement activity directly
  - Use max/min limits to force movements into desired window
  - Make resource availability time-varying

Shipments

Represent a movement of material out of the system.

- By default, modelled as a movement with no destination
- Only works if shipment always comes out of a tank.

With inline blending:

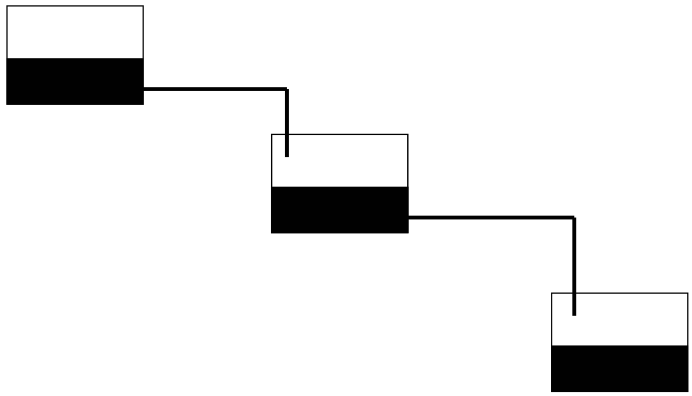
- Sometimes the “source” of a movement is another movement!

Paths

Paths through a factory are generally limited. Their indirect effects include:

- Choice of movement source constrains possible destinations
- Rate limits (which become duration bounds)
- Connection equipment resource requirements

Stretch Goal: Tracking Material Properties



What Kind of Planning is Needed?

- Multiple asynchronous activities in different parts of the plant.
- Single discrete choices can lead to multiple (synchronous or asynchronous!) activities.
- Current model uses durative actions (intervals), which might be hard to move away from (many continuous constraints are dependent on the extent of an interval).
- *Some* of the inter-activity relationships can be described using propositions.

What Kind of Scheduling is Needed?

- Multiple asynchronous activities in different parts of the plant
- Many and varied continuous constraints among activities
- Global resource constraints across the plant (e.g., hydrogen balance, RFG)
- Resource constraints integrated across the extent of the schedule (e.g., volume, properties)
- Constraints imposed by (and altered by) external agency
- Many of these are in the form of activity "requirements."

Planning and Scheduling as a Hybrid DCSP/DCOP

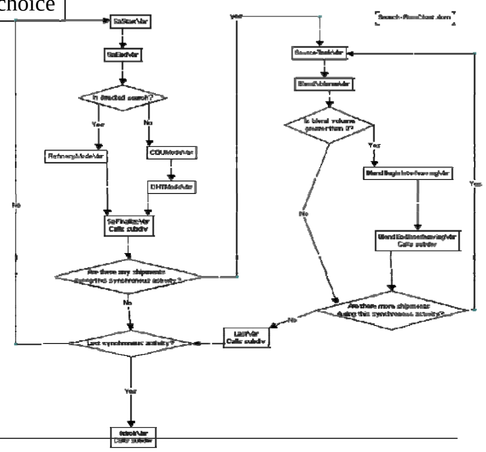
- Refinement search over discrete decisions
  - Activity generation
  - Discrete search within resulting CSP (resource choice)
- Continuous feasibility checking:
  - Constraint reduction
  - Propagation
  - Linear solve
  - "Subdivision search" over full nonlinear model.
- Objective function used as a heuristic during search, with a final optimizing solve on full set of continuous constraints.

Discrete search

- 3-5 crude charges in a 10-day schedule.
- 22 shipments
- 2-5 gasoline blends
  - blend volume is currently a discrete choice

Summary:

1) Create a crude-charge activity
   - start, end time
   - CDU mode
   - DHT mode
2) Create shipment activities
3) Create gasoline blend activities
4) Repeat, to end of schedule



Process-Industry Planning

1. Process-industry “scheduling” has a lot of planning in it.
2. Constraint-based models are useful or required, because the bulk of the constraints aren’t propositional.
3. A single thread of control is a fiction.
4. Externally-imposed (and modified!) constraints on operations are crucial, and complicated.
5. 18,000 continuous constraints (2700 quadratic) on 14,000 variables

Summary and conclusions

- Takeaway messages
- Things we didn't have time to cover
  - Golog-style planning
  - Planning and Execution
  - Uncertainty
  - Domain analysis
- For further information

Readily Available Planners

- Forward, heuristic search
  - FF, Metric FF: http://www.mpi-sb.mpg.de/~hoffmann/metric-ff.html
  - SimPlan and TLPlan : http://planiart.usherbrooke.ca/
    These planners use Temporal logic to encode information to be used in controlling search.
- Plan graph
  - SGP: http://www.cs.washington.edu/ai/sgp.html (somewhat old)
- SAT
  - SATPLAN-2004: http://www.cs.washington.edu/homes/kautz/satplan
  - LPG.td: http://prometeo.ing.unibs.it/lpg
- HTN
  - SHOP2: http://sourceforge.net/projects/shop
- POCL
  - UCPOP: http://www.cs.washington.edu/ai/ucpop.html
  - RePOP: http://rakaposhi.eas.asu.edu/repop.html
  - VHPOP: http://www-2.cs.cmu.edu/~lorens/vhpop.html
- Worth a look at the International Planning Competition page: http://ipc.icaps-conference.org

Additional Resources, General

- *Automated Planning*, Malik Ghallab, Dana Nau, Paolo Traverso, Morgan Kaufmann, 2004.

POP/UnPOP

- D. McAllester and D. Rosenblitt. *Systematic nonlinear planning*. In Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91), volume 2, pages 634--639, Anaheim, California, USA, 1991. AAAI Press/MIT Press.
- Penberthy, J. S. and Weld, D., ``UCPOP: A Sound, Complete, Partial-Order Planner for ADL," Third International Conference on Knowledge Representation and Reasoning (KR-92), Cambridge, MA, October 1992.
- McDermott's unPOP: Drew McDermott Using regression-match graphs to control search in planning. *Artificial Intelligence*,**109** (1-2), pp. 111-159, 1999.
- Subbarao Kambhampati's RePOP page http://rakaposhi.eas.asu.edu/repop.html

HTN Planners

- Sipe: ***Practical planning: extending the classical AI planning paradigm***, David Wilkins, Morgan Kaufmann Publishers Inc., 1988.
- O-Plan: Ken Currie and Austin Tate, "O-Plan: the open planning architecture," *Artificial Intelligence*, Volume 52 , Issue 1 (November 1991).
- HTN semantics
  - Erol, K., Hendler, J., & Nau, D. (1994). "HTN planning: complexity and expressivity." In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI94)* . AAAI Press.
  - K. Erol, J. Hendler, and D. Nau. "UMCP: A sound and complete procedure for hierarchical task-network planning". In *Proc. 2nd Intl. Conf. on A.I. Planning Systems*, pages 249--254, June 1994.
- SHOP:
  - D. Nau, T.-C. Au, O. Ilghami, U. Kuter, H. Muñoz-Avila, J. W. Murdock, D. Wu, and F. Yaman. Applications of SHOP and SHOP2. *IEEE Intelligent Systems*, 2005. An earlier version is available as Tech. Rep. CS-TR-4604, UMIACS-TR-2004-46.
  - SHOP2 web page: http://www.cs.umd.edu/projects/shop/index.html

Cyber-Security

- *Course of Action Generation for Cyber Security Using Classical Planning* Mark Boddy, Johnathan Gohde, Thomas Haigh, and Steven Harp, ICAPS-05.
- Noel, S.; Jajodia, S.; O'Berry, B.; and Jacobs, M. 2003. Efficient minimum-cost network hardening via exploit dependency graphs. In *Proceedings of 19th Annual Computer Security Applications Conference*, 86–95. IEEE Computer Society.
- Ritchey, R. W., and Ammann, P. 2000. Using model checking to analyze network vulnerabilities. In *Proceedings 2000 IEEE Computer Society Symposium on Security and Privacy*, 156–165.
- Sheyner, O. 2004. *Scenario Graphs and Attack Graphs*. Ph.D. Dissertation, Computer Science Department, Carnegie Mellon.
- Zerkle, D., and Levitt, K. 1996. NetKuang–A multi-host configuration vulnerability checker. In *Proc. of the 6th USENIX Security Symposium*, 195–201.

Airport Planning

- Hatzack, W., & Nebel, B. (2001). The operational traffic control problem: Computational complexity
- and solutions. In Cesta, A., & Borrajo, D. (Eds.), *Recent Advances in AI Planning. 6th*
- *European Conference on Planning (ECP'01)*, pp. 49–60 Toledo, Spain. Springer-Verlag.
- ftp://ftp.informatik.uni-freiburg.de/documents/papers/ki/hatzack-nebel-ecp01.pdf
- IPC Domain
- Tr¨ug, S., Hoffmann, J., & Nebel, B. (2004). Applying automatic planning systems to airport groundtraffic
- control — a feasibility study. In *Proceedings of KI-04: Advances in Artificial Intelligence*.

Process Industries

- *Integrated Planning and Scheduling for Petroleum Refinery Operations*, Mark Boddy and Daniel P. Johnson, Workshop on Integrating Planning into Scheduling, ICAPS-04.
- Leffler, W. L. 1985. *Petroleum Refining for the Nontechnical Person*. John Wiley Pennwell Pub.