Ordered Task Decomposition: Theory and Practice

Dana S. Nau
Dept. of Computer Science, and
Institute for Systems Research
University of Maryland, College Park, MD
Generating Plans of Action

- Programs to aid human planners
  - Project management (consumer software)
  - Plan storage and retrieval
    - (e.g., variant process planning)
  - Automatic schedule generation (various OR and AI techniques)

- For some problems, really want to generate plans automatically
  - Much more difficult
  - Very few successful programs for realistic problems

- AI planning research is starting to pay off
  - Of the few really good plan generation systems for practical problems, most involve AI planning techniques
  - However, …
AI Planning Is Different in Practice Than it Was in Theory

Unstack(x, y)
Pre: on(x, y), clear(x), handempty
Del: on(x, y), clear(x), handempty
Add: holding(x), clear(y)

Theory:
- Symbolic computations (STRIPS operators)
- Single agent (the planner)
- Perfect information

Practice:
- Complex numeric computations (geometry, images, probabilities)
- Multiple agents
- Imperfect information, external information sources
Goal

- Develop synergy between theory and applications

  - By understanding what works in practical planning situations, we can develop better theories of planning
  - Better theories of planning can lead to better real-world planners
Mainly I’ll use PowerPoint slides
At two points, I can run demos in Lisp
Please ask questions!
1. Principles of HTN Planning

**Theory**

1. Principles of HTN planning

2. Computer bridge

3. Electronic Design and Manufacturing

4. Ordered Task Decomposition

5. SHOP

**Applications**

6. Evacuation planning

- Joint work with Kutluhan Erol and Jim Hendler
What HTN Planning Is

- A type of problem reduction
  - Decompose tasks into subtasks
  - Handle constraints (e.g., taxi not good for long distances)
  - Resolve interactions (e.g., take taxi early enough to catch plane)
  - If necessary, backtrack and try other decompositions

![Diagram showing HTN planning process with tasks and methods]
Comparison with “Classical” AI Planning

- “Classical” AI planning is action-based
  - Declarative descriptions of actions
  - Specify declaratively what the operators are capable of doing
  - Don’t prescribe how to perform complex tasks
  - The planner must infer that, using trial-and-error search

- HTN decomposition
  - Can represent STRIPS-style declarative operators, but it’s clumsy
  - Easy to specify “recipes” for how to perform tasks

\[
\text{buy-ticket}(x,y) \\
\text{pre:} \ \text{airport}(x), \text{have-money}() \\
\text{del:} \ \text{have-money}() \\
\text{add:} \ \text{have-ticket}(x,y)
\]

\[
\text{fly}(x,y) \\
\text{pre:} \ \text{at}(x), \text{have-ticket}(x,y) \\
\text{del:} \ \text{at}(x), \text{have-ticket}(x,y) \\
\text{add:} \ \text{at}(y)
\]
History of HTN Planning

- Originally developed about 25 years ago
  - [Sacerdoti 1977; Tate 1977]
  - Long thought to have better potential for applicability to real-world planning problems than classical STRIPS-style planning
  - Progress delayed due to lack of theoretical understanding

- More recent work: theoretical basis for HTN planning
  - Formal semantics
    - HTN’s are strictly more expressive than STRIPS operators
      [Erol et al., 1994a]
  - Sound and complete algorithm [Erol et al., 1994b]
  - Complexity analysis [Erol et al., 1996]
  - This has helped to spread interest in HTN planning
What is Expressivity?

- **Expressivity of languages**
  - A language $L$ is as expressive as another language $M$ iff any expression in $L$ can be translated into an expression with the same meaning in $M$.

- **Possible ways to define “meaning”**
  1. based on computational complexity
  2. based on model theory
  3. based on operational semantics

- **HTN planning is more expressive than state-based planning according to all three of these definitions**
  - Will summarize all three
1. Complexity-Based Expressivity

- Transformation must preserve answer ("yes" or "no")
- Transformation must be computable/polynomial
- Affected by the conciseness of the language
HTN Language Versus STRIPS Language

- STRIPS-style planning is a special case of HTN planning
  - Erol [1995] presents two polynomial transformations from the STRIPS planning language to the HTN planning language
- There is no computable transformation from the HTN language to the STRIPS language, because HTNs can represent harder problems than the STRIPS language
- Example problem:
  - **Given two context-free languages L1 and L2, do they have a non-empty intersection?**
  - This problem is undecidable
  - It can’t be represented as a STRIPS-style planning problem (not unless you augment the STRIPS formalism to include function symbols!)
Given two context-free languages $L_1$ and $L_2$, do they have a non-empty intersection?

This problem can be represented as an HTN planning problem:
- You don’t need to use function symbols
- You don’t even need to use any variable symbols!
- However, you need to make use of
  - cycles in methods
  - interleaving among subtasks
2. Model-Theoretic Expressivity

- Erol [1995] extended the work of Baader on knowledge representation languages to planning languages
  - The transformation must preserve meaning
    - The set of models satisfying a sentence and the set of models satisfying its transformation must be equivalent
  - No restrictions on the computational aspects of the translation
    - Not affected by the conciseness of the languages

- Result
  - Erol’s transformations from STRIPS to HTN (mentioned earlier) preserve the set of models
  - No transformations in the other direction, because HTN models have richer structure
  - Thus the HTN language is more expressive than the STRIPS language
3. Operational-Semantic Expressivity

- Transformation must preserve the set of solutions (plans)

- Result:
  - Solutions to STRIPS problems are regular sets
  - Solutions to HTN problems can be arbitrary context-free sets
  - Thus HTN’s are more expressive than STRIPS
Related Publications


2. Computer Bridge

Theory

1. Principles of HTN planning → 4. Ordered Task Decomposition → 5. SHOP

2. Computer bridge

Applications

3. Electronic Design and Manufacturing

6. Evacuation planning

- Joint work with Stephen J. Smith and Tom Throop
Computer Programs for Games of Strategy

- Fundamental technique: minimax game-tree search
  - Largely “brute force”
    - Can prune off portions of the tree
      - cutoff depth,
      - alpha-beta pruning,
      - hash tables, …
    - Even then, it still examines thousands of game positions

- This is very different from how humans think
  - Even the best human chess players examine at most a few dozen game positions to make their moves
Performance of Game-Playing Computer Programs

Connect Four: solved
Go-Moku: solved
Qubic: solved
Nine Men’s Morris: solved
Othello: better than humans
Checkers: better than any living human
Backgammon: better than all but about 10 humans
Chess: certainly better than all but about 250 humans; possibly even better than that

Bridge: probably worse than the best player at your local bridge club
How Bridge Works

- Four players; 52 playing cards dealt equally among them
- Bidding to determine the trump suit
  - *Declarer*: whoever makes highest bid
  - *Dummy*: declarer’s partner
- The basic unit of play is the *trick*
  - One player leads; the others must follow suit if possible
  - Trick won by highest card of the suit led, unless someone plays a trump
  - Keep playing tricks until all cards have been played
- Scoring based on how many tricks were bid and how many were taken
Why Bridge is Difficult for Computers

- Bridge is an *imperfect information* game
  - Don’t know what cards the others have (except the dummy)
  - Many possible card distributions, so many possible moves
- If we encode the additional moves as additional branches in the game tree, this increases the number of nodes exponentially
  - worst case: about $6 \times 10^{44}$ leaf nodes
  - average case: about $10^{24}$ leaf nodes

Not enough time to search the game tree
How to Reduce the Size of the Game Tree?

- Bridge is a game of planning
  - The declarer plans how to play the hand
  - The plan combines various strategies (ruffing, finessing, etc.)
  - If a move doesn’t fit into a sensible strategy, it probably doesn’t need to be considered

- Our approach for declarer play
  - Adaptation of an Hierarchical Task-Network (HTN) planning
  - Generate a game tree in which each move corresponds to a different strategy, not a different card

<table>
<thead>
<tr>
<th></th>
<th>Brute-force search</th>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst case</td>
<td>$6 \times 10^{44}$ leaf nodes</td>
<td>$305,000$ leaf nodes</td>
</tr>
<tr>
<td>Average case</td>
<td>$10^{24}$ leaf nodes</td>
<td>$26,000$ leaf nodes</td>
</tr>
</tbody>
</table>
Task Network for Finessing

Us: East declarer, West dummy
Opponents: defenders, South & North
Contract: East – 3NT
On lead: West at trick 3

East: ♣KJ74
West: ♣A2
Out: ♣QT98653

primitive action by us

primitive action by opponent
Game Tree Generated from the Task Network

...later strategies...

-100

+630

+630

+600

+600

CASH OUT

W—A

N—3

E—4

S—5

+600

+600

+600

FINESSE

W—2

N—Q

N—3

E—J

E—K

S—Q

S—3

S—3

+600

+630

+630

+600

+265

+265

0.5

0.5

0.0078

0.0078

0.9854

...later strategies...
Implementation

- We incorporated our code for declarer play into *Bridge Baron* (an existing commercial program)
- Using our code, *Bridge Baron* won the 1997 *World Bridge Computer Challenge*
- Our code has been used in all versions of *Bridge Baron* since October 1997
  - Has sold many thousands of copies
Related Publications

[Smith et al., 1996]

[Smith et al., 1998]
3. Electronic Design and Manufacturing

Theory

1. Principles of HTN planning
2. Computer bridge
3. Electronic Design and Manufacturing
4. Ordered Task Decomposition
5. SHOP

Applications

● Joint work with Stephen J. Smith, Kiran Hebbar, and Ioannis Minis
Augment the traditional “engineering design” loop

- Plan and evaluate what manufacturing processes will be needed
- Predict cost, time, quality, manufacturing problems
- Modify the design to improve its manufacturability
Microwave Transmit/Receive Modules

- 1-20 GHz frequency range (radars, satellite communications, etc.)
- Difficult and expensive to design and manufacture
EDAPS: Electro-mechanical Design And Planning System

Design, Constraints

Information about manufacturability

User Interface (Tcl/Tk)

Circuit Schematic, Component Selection

Substrate Design & 3-D Layout

Process Planning & Plan Evaluation

EEsof

Microstation

HTN Planner (C++)

AEL Routines

MDL Routines

- Commercial

Routines in C++

Product and Process Data Files

- Developed by us

Designer

Designer

Designer
EDAPS’s Process Planner

A portion of the task network:

- Make the artwork
  - (one possible method)
  - Preclean for artwork
  - Apply photoresist
  - Photolithography
  - Etching
  - Spindling
  - Spraying
  - Spreading
  - Painting

- Can express planning using “recipes” that fit well into HTN methods

  - Generate and evaluate multiple process plans
  - Estimate times and costs
  - Display results graphically
## Examples from EDAPS

### Substrate
- **Dimensions:** 7,4,1
- **Ground Material:** Aluminum
- **Material:** Teflon
- **Substrate thickness:** 30 mils
- **Metallized thickness:** 7 mils

### Processes:

<table>
<thead>
<tr>
<th>Opn A BC/WW</th>
<th>Setup</th>
<th>Runtime</th>
<th>LN</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>001 A VMC1</td>
<td>2.00</td>
<td>0.00</td>
<td></td>
<td>Orient board vertically</td>
</tr>
<tr>
<td>002 A</td>
<td></td>
<td></td>
<td></td>
<td>Clamp board at (1,1,1)</td>
</tr>
<tr>
<td>003 A</td>
<td></td>
<td></td>
<td></td>
<td>Establish datum point</td>
</tr>
<tr>
<td>001 B VMC1</td>
<td>0.10</td>
<td>0.43</td>
<td></td>
<td>Tool: 0.30-diameter drill bit</td>
</tr>
<tr>
<td>002 B</td>
<td></td>
<td></td>
<td></td>
<td>Drill at (1.25,-0.50)</td>
</tr>
<tr>
<td>003 B</td>
<td></td>
<td></td>
<td></td>
<td>Drill at (1.25,-0.50) d:1.00 f:50 s:30</td>
</tr>
<tr>
<td>001 C VMC1</td>
<td>0.10</td>
<td>0.77</td>
<td></td>
<td>Tool: 0.20-diameter slot miller</td>
</tr>
<tr>
<td>002 C</td>
<td></td>
<td></td>
<td></td>
<td>Mill start (0.044, 4.88)</td>
</tr>
<tr>
<td>003 C</td>
<td></td>
<td></td>
<td></td>
<td>l: 0.5, w: 0.5, d: 1.00, f: 50, s: 40</td>
</tr>
<tr>
<td>001 T VMC1</td>
<td>2.20</td>
<td>1.20</td>
<td></td>
<td>Total time on VMC1</td>
</tr>
</tbody>
</table>

| Opn A PLAT1 | 1.00  | 0.60    | 02  | Dip in bath for 2 minutes
|            |       |         |     | Temperature: 100°C, Conc: 1000 ppm             |
| Opn A ETR1  | 0.50  | 0.60    | 01  | Etch plate for 1 minute
|            |       |         |     | Temperature: 100°C, Conc: 1000 ppm             |
| Opn A ETC1  | 0.20  | 0.30    | 01  | Etch board for 2 minutes
|            |       |         |     | Temperature: 100°C, Conc: 1000 ppm             |
Nau: PLANET, 2000

**Status**

- EDAPS completed under NSF funding
- Follow-up project with Northrop Grumman
  - Combine AI planning with Integer Programming optimization

### Electronic CAD
- Initial component selection

### Database lookup
- Alternative components

### HTN planning
- Alternative plans

### Multi-objective Integer Programming
- Pareto-optimal combinations

### Interactive display
- User exploration and selection
Related Publications


4. Ordered Task Decomposition

**Theory**

1. Principles of HTN planning
2. Computer bridge
3. Electronic Design and Manufacturing
4. Ordered Task Decomposition
5. SHOP

**Applications**

- Joint work with Kutluhan Erol, Naresh Gupta, and Stephen J. Smith
Discussion

- For both Bridge Baron and EDAPS
  - We used the same approach
  - Even some of the same code!

- Ordered Task Decomposition
  - Adaptation of HTN planning
    - Linear ordering on the subtasks of each method
    - Expand the subtasks in the same order in which they will be executed later on

- Compare and contrast with “classical” AI planning
  1. Search strategy
  2. Data representation
Search Strategy

- Ordered task decomposition
  - Adaptation of HTN planning
  - Require the subtasks of each method to be totally ordered
  - Decompose these tasks left-to-right
    - The same order that they’ll later be executed
    - Analogous to PROLOG’s search strategy

Make the artwork for a PC board

- Preclean for artwork
- Apply photoresist
- Photolithography
- Etching

Spindling
Spraying
Spreading
Painting

Photolithography
Apply photoresist
Preclean for artwork
Make the artwork for a PC board
With action-based planning, you have an either/or choice:
- goal-directed search (backward from the goal)
- forward search (forward from the initial state)

In contrast, ordered task decomposition is both forward and goal-directed at the same time.
STRIPS operator to stack a block

\[
\text{stack}(x,y) \\
\text{Pre: } \text{holding}(x), \text{clear}(y) \\
\text{Del: } \text{holding}(x), \text{clear}(y) \\
\text{Add: } \text{on}(x,y), \text{clear}(x), \text{handempty}
\]

Translate it into an HTN method

- If we expand the subtasks in left-to-right order, then we are searching forward from the initial state
  - Always know the current state
- However, it’s also a goal-directed search
  - The task is the goal
  - Invoke only those methods that match the task
The operators aren’t totally ordered

- How do we know what the states are?
- Represent states as sets of logical atoms
  - Partially instantiate them to represent what we know about them
    - protected conditions in POP, mutex conditions in Graphplan
- This works (it leads to sound and complete planners)
- Since the states aren’t totally instantiated, it’s hard to reason about them in all of the ways we might like
  - E.g., can’t call a CAD package to reason about the geometry of a machined part if some of the geometry is uninstantiated
State Representation for Ordered Task Decomposition

- Goal-directed, but generates actions in the same order in which they will later be executed
- Whenever we want to plan the next task
  - we’ve already planned everything that comes before it

Thus, we know the current state of the world
Increased Expressivity

- If we know what the current state is, then we can do complex reasoning about it
  - Logical inferences
  - Numeric and probabilistic computations
  - Interactions with external programs

- Example
  - In computer bridge, by knowing the current state, can decide which of nineteen finesse situations are applicable
  - With partial-order planning, it would be hard to decide which of them can be made applicable

- Can do lots of pruning
  - Often only one or two consistent “next steps”
    - Bridge declarer play: complete plans in about 1 1/2 minutes
    - Process plans for microwave modules: a few seconds
Example: Blocks-World Planning

- The blocks world

- On the next page is the best blocks-world planning algorithm I know of
  - [Gupta & Nau, 1992]
  - It finds near-optimal plans in low-order polynomial time

- In this section, I’ll describe the algorithm

- In the next section, I’ll explain how to implement it using Ordered Task Decomposition
The Algorithm

- loop
  - if there is a clear block \( x \) such that
    - \( x \) or a block beneath \( x \) is in a location inconsistent with the goal
    - and
    - \( x \) can be moved to a location such that \( x \) and all blocks beneath it will be in locations consistent with the goal
  - then move \( x \) to that location
  - else if there is a clear block \( x \) such that
    - \( x \) or a block beneath \( x \) is in a location inconsistent with the goal
  - then move \( x \) to the table
  - else exit
  - endif
- repeat
Running the Algorithm

- Running the algorithm on the Sussman anomaly

Initial state:
- clear(c), on(c,a), ontable(a), clear(b), ontable(b), handempty

Goal:
- on(a,b), on(b,c)
Encoding the Algorithm

- loop
  - if there is a clear block $x$ such that
    - $x$ or a block beneath $x$ is in a location inconsistent with the goal
    and
    - $x$ can be moved to a location such that $x$ and all blocks beneath it will be in locations consistent with the goal
  - then move $x$ to that location
  - else if there is a clear block $x$ such that
    - $x$ or a block beneath $x$ is in a location inconsistent with the goal
  - then move $x$ to the table
  - else exit
  - endif

- repeat
  - Can’t write these preconditions using STRIPS-style operators
  - Can write them as Horn clauses
  - If we know the current state, we can infer whether they hold
  - Thus, we can write the algorithm using ordered task decomposition. In the next section, I’ll show how to do that
Related Publications

[Nau et al., 1998]

[Gupta & Nau, 1992]
5. SHOP

Theory

1. Principles of HTN planning → 4. Ordered Task Decomposition → 5. SHOP

Applications

- Joint work with Yue Cao, Amnon Lotem, and Héctor Muñoz-Avila
SHOP (Simple Hierarchical Ordered Planner)

- Domain-independent algorithm for Ordered Task Decomposition
  - Sound/complete across a large number of planning domains

- Implementation
  - Developing a Java implementation
Input and Output

- **Input:**
  - **State:** a set of ground atoms
  - **Task List:** a linear list of tasks
  - **Domain:** methods, operators, axioms

- **Output:** one or more plans
  - depending on what we tell SHOP to look for, it can return
    - the first plan it finds
    - all possible plans
    - a least-cost plan
    - all least-cost plans
    - etc.
Elements of the Input

- **State**: collection of ground atoms (in Lisp notation)
  - ((at home) (have-cash 50.43) (distance home downtown 10))

- **Task list**: linear list of tasks to perform
  - ((travel home downtown) (buy book))

- **Each method**: task, preconditions and decomposition
  - Preconditions to be established using logical inference
  - Decomposition is a task list

- **Each axiom**: Horn clause
  - Extensions: may contain negations and calls to the Lisp evaluator

- **Each primitive operator**: task, delete list, add list
  - Like a STRIPS operator, but without the preconditions
  - Performs a primitive task
Simple Example

- **Initial task list:** ((travel home park))
- **Initial state:** ((at home) (cash 20) (distance home park 8))
- **Methods** (task, preconditions, subtasks):
  - (:method (travel ?x ?y)
    ((at ?x) (walking-distance ?x ?y))' ' (!walk ?x ?y) 1)
  - (:method (travel ?x ?y)
    ((at ?x) (have-taxi-fare ?x ?y))
    ' (!call-taxi ?x) (!ride ?x ?y) (!pay-driver ?x ?y) 1)
- **Axioms:**
  - (:- (have-taxi-fare ?x ?y)
    ((have-cash ?c) (distance ?x ?y ?d) (eval (>= ?c (+ 1.50 ?d))))
- **Primitive operators** (task, delete list, add list)
  - (:operator (!walk ?x ?y) ((at ?x)) ((at ?y)))
  - …

Optional cost; default is 1
The SHOP Algorithm

procedure SHOP (state $S$, task-list $T$, domain $D$)
1. if $T = \text{nil}$ then return nil
2. $t_1 =$ the first task in $T$
3. $U =$ the remaining tasks in $T$
4. if $t$ is primitive & an operator instance $o$ matches $t_1$ then
5. $P =$ SHOP ($o(S)$, $U$, $D$)
6. if $P =$ FAIL then return FAIL
7. return cons($o, P$)
8. else if $t$ is non-primitive & a method instance $m$ matches $t_1$ in $S$
   & $m$’s preconditions can be inferred from $S$ then
9. return SHOP ($S$, append ($m(t_1)$, $U$), $D$)
10. else
11. return FAIL
12. end if
end SHOP
Initial state:

(at home)
(cash 20)
(distance home park 8)

Precond:

(at home)
(walking-distance Home park)

Succeed

Fail (distance > 5)

(!walk home park)

Precond:

(at home)
(have-taxi-fare home park)

Succeed

Succeed (we have $20, and the fare is only $9.50)

(!call-taxi home)
(!ride home park)
(!pay-driver home park)

Simple Example
(Continued)

Final state:

(at park)
(cash 10.50)
(distance home park 8)
Question

- I can run SHOP for you right now, on a slightly more complicated version of the example.
- Would you like me to do so?
Homework Assignment

- Formulate a plan for going to the beach
- Execute the plan