Ordered Task Decomposition: Theory and Practice

Part 2

Dana S. Nau
Dept. of Computer Science, and
Institute for Systems Research
University of Maryland, College Park, MD
Approach (and Outline)

Theory

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

Applications

- Very quick review
- Comments
- Continue where I left off
What Activities Should a Planning System Plan?

- In AI planning, researchers traditionally have only allowed the planner to plan activities that will have a direct physical effect.
  - Examples:
    - picking up a block
    - moving a truck
  - In human planning, we also plan lots of other activities
What Activities Should a Planning System Plan?

- In AI planning, researchers traditionally have only allowed the planner to plan activities that will have a direct physical effect

- Examples:
  - picking up a block
  - moving a truck

- In human planning, we also plan lots of other activities

- Example 1:
  - Planning information-gathering operations

```
travel by air
airport(x,a)  airport(y,b)  ticket (a,b)  travel (x,a)  fly(a,b)  travel(b,y)
```
What Activities Should a Planning System Plan?

- In AI planning, researchers traditionally have only allowed the planner to plan activities that will have a direct physical effect
  - Examples:
    - picking up a block
    - moving a truck
- In human planning, we also plan lots of other activities
- Example 2:
  - Planning bookkeeping operations
What Activities Should a Planning System Plan?

- In AI planning, researchers traditionally have only allowed the planner to plan activities that will have a direct physical effect.

- Examples:
  - picking up a block
  - moving a truck

- In human planning, we also plan lots of other activities.

- Example 3:
  - Planning when/how to create and retrieve plans.

- travel by air

  - plan how to travel to Cyprus
  - finish work at the office
  - pack for the trip
  - execute the stored plan
  - store this plan into the current state
Homework Assignment

- Yesterday, I asked you to formulate and execute a plan for going to the beach
- Analyze that plan, to see if it contains any of the following activities:
  1. information-gathering operations
  2. bookkeeping operations
  3. planning when/how create and retrieve plans
Concrete Example

- Malik Ghallab and Piergiorgio Bertoli and I plan to go bicycling on Sunday (instead of going to the barbecue)

- One part of the plan:
  - Find a bicycle shop
  - Find out if they have suitable bicycles
  - If so, then plan how to go to it

- This involves information-gathering, bookkeeping, and planning when to do other planning
Concrete Example

- Malik Ghallab and Piergiorgio Bertoli and I plan to go bicycling on Sunday (instead of going to the barbecue)

- One part of the plan:
  - Find a bicycle shop
  - Find out if they have suitable bicycles
  - If so, then plan how to go to it

- This involves information-gathering, bookkeeping, and planning when to do other planning

- Another information-gathering part of the plan:
  - Would you like to join us?
  - If so, please let us know!
Review of 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
Answer to a Question from Yesterday

- **Question:** Suppose there’s an action $b$ that will come late in the plan and will cause a lot of backtracking.

- In this case we might like to plan for $b$ first. Won’t total-order planning preclude us from doing that?
Answer to a Question from Yesterday

**Question:** Suppose there’s an action $b$ that will come late in the plan and will cause a lot of backtracking.

In this case we might like to plan for $b$ first. Won’t total-order planning preclude us from doing that?

**Answer:** Not necessarily

One way to do it: Tell the planner to create a plan for $b$ and retrieve that plan later.
Answer to a Question from Yesterday

- **Question**: Suppose there’s an action $b$ that will come late in the plan and will cause a lot of backtracking.
- In this case we might like to plan for $b$ first. Won’t total-order planning preclude us from doing that?
- **Answer**: Not necessarily

- Another way to do it: Tell the planner to start by checking $b$’s preconditions, without actually planning for $b$
Answer to Another Question

**Question:** With total-order planning, can we interleave subgoals?

![Diagram](image.png)
**Question:** With total-order planning, can we interleave subgoals?

**Answer:**
- Not in SHOP, because all of the subtasks will be totally ordered.
- Sometimes we can circumvent this, by using methods that put the subtasks into a different order.
  - In some cases this is natural; in other cases it is clumsy.
- We have an extension to the SHOP algorithm called M-SHOP [Nau et al., TR, 2000], in which some of the subtasks can be unordered.
Basic idea for M-SHOP:

- When we have several task lists without an ordering between them, do this:
  
  ```
  loop
  if all of the task lists are empty, then exit
  remove the first item from one of the task lists
  decompose it, and put its subtasks at the front of the task list
  repeat
  ```

- Need a way to handle protected conditions
  
  - How we do this is described in [Nau et al., 2000]

- M-SHOP is available at http://www.cs.umd.edu/projects/shop as part of the latest software distribution for SHOP
  
  - Currently they are separate programs
  - In the next release, they'll be merged into a single program
M-SHOP doesn’t fully implement partially ordered task lists
- It can take multiple task lists as part of its input
- It doesn’t allow methods to have subtasks that are partially ordered

However
- To allow methods to have partially ordered subtasks is an easy generalization
- We’ve written down the algorithm, but haven’t implemented it yet
**Review of the SHOP Algorithm**

**procedure** SHOP (state S, task-list T, domain D)

1. **if** \( T = \text{nil} \) **then** return nil
2. \( t_1 = \text{the first task in } T \)
3. \( U = \text{the remaining tasks in } T \)
4. **if** \( t \) is primitive \& an operator instance \( o \) matches \( t_1 \) **then**
5. \( P = \text{SHOP } (o(S), U, D) \)
6. **if** \( P = \text{FAIL} \) **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 \( (S, \text{append}(m(t_1), U), D) \)
10. **else**
11. return FAIL
12. **end if**

**end SHOP**
Encoding the Blocks World Algorithm into SHOP

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 x 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
How to Encode
"
x or a block beneath it is in a location inconsistent with the goal"

(:- (need-to-move ?x)
    ;; need to move x if x needs to go from one block to another
    ((on ?x ?y) (goal (on ?x ?z)) (different ?y ?z))
    ;; need to move x if x needs to go from table to block
    ((on-table ?x) (goal (on ?x ?z)))
    ;; need to move x if x needs to go from block to table
    ((on ?x ?y) (goal (on-table ?x)))
    ;; need to move x if x is on y and y needs to be clear
    ((on ?x ?y) (goal (clear ?y)))
    ;; need to move x if x is on z and something else needs to be on z
    ((on ?x ?z) (goal (on ?y ?z)) (different ?x ?y))
    ;; need to move x if x is on something else that needs to be moved
    ((on ?x ?w) (need-to-move ?w)))
Example (Continued)

```
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
```
Operators for Moving Blocks

(:operator (!pickup ?x)
    ((clear ?x) (on-table ?x))
    ((holding ?x)))

(:operator (!putdown ?x)
    ((holding ?x))
    ((on-table ?x) (clear ?x)))

(:operator (!stack ?x ?y)
    ((holding ?x) (clear ?y))
    ((on ?x ?y) (clear ?x)))

(:operator (!unstack ?x ?y)
    ((clear ?x) (on ?x ?y))
    ((holding ?x) (clear ?y)))
Example (Continued)

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
Suppose we use the task list

- `((achieve (on a b)) (achieve (on b c)))`

This won’t do what we want

- First it will perform the task `(achieve (on a b))`
- Then it will perform the task `(achieve (on b c))`

Instead, we need a task list something like

- `((achieve (on a b) (on b c)))`

Need to keep track of

- which goals we have achieved and need to preserve
- which goals remain to be achieved

How to do this?
Bookkeeping

- Keeping track of what the goals are
  - Insert information about this into the current state
  - This is convenient because it lets us use precondition-matching to match methods to goals

- Keeping track of the goals that we have achieved and need to preserve
  - Insert information about this into the task list
  - Could also have put this into the current state instead
    - That’s what we do in other domains, such as the logistics domain [Nau et al., 2000]
Bookkeeping Operator and Methods

(:operator (!assert ?atoms) ; ?atoms is a list of atoms to assert
 () ; no preconditions
 ?atoms ; put the list of atoms into the current state
 0) ; this operator has no cost

(:method (assert-goals (?first . ?rest) ?atoms) ; recursively build a list
 () ; of atoms to assert into
 '((assert-goals ?rest ((goal ?first) . ?atoms)))) ; the current state

(:method (assert-goals nil ?atoms) ; we’ve built the entire list, so assert it
 ()
 '(!assert ?atoms))

(:method (achieve-goals ?goals) ; assert all the goals into the current state,
 () ; then call move-block to achieve them
 '((assert-goals ?goals nil) (move-block nil)))
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

(:method (move-block ?nomove)

;;; method for moving x from y to z
(:first (clear ?x) (eval (not (member '?x '?nomove))) (on ?x ?y)
 (goal (on ?x ?z)) (different ?x ?z) (clear ?z) (not (need-to-move ?z)))
'((!unstack ?x ?y) (!stack ?x ?z) (move-block (?x . ?nomove)))

;;; method for moving x from y to table
(:first (clear ?x) (eval (not (member '?x '?nomove)))
 (on ?x ?y) (goal (on-table ?x)))
'((!unstack ?x ?y) (!putdown ?x) (move-block (?x . ?nomove)))

;;; method for moving x from table to y
(:first (clear ?x) (eval (not (member '?x '?nomove)))
 (on-table ?x) (goal (on ?x ?y)) (clear ?y) (not (need-to-move ?y)))
'((!pickup ?x) (!stack ?x ?y) (move-block (?x . ?nomove)))

...
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

(:method (move-block ?nomove)
  ...
  ;; method for moving x out of the way
  ((clear ?x) (eval (not (member '?x '?nomove)))
   (on ?x ?y) (need-to-move ?x))

  ;; if none of the above preconditions were satisfied, then we're done
  nil
  nil)
Question

- I can run SHOP on the blocks world for you right now.
- Would you like me to do so?
Experimental Comparison

- Experimental comparison
  - Planning domains:
    - Blocks world
    - Logistics
    - UM Translog
  - Planning systems compared:
    - Blackbox - Graphplan plus satisfiability
    - IPP - Graphplan plus ADL
    - TLplan - forward search with modal-logic pruning rules
    - UMCP - “classical” HTN planning
    - SHOP - Ordered Task Decomposition
Blocks World

- 100 randomly generated problems
- 167-MHz Sun Ultra with 64 MB of RAM
- Blackbox and IPP could not solve any of these problems
- TLPlan’s running time was only slightly worse than SHOP’s
  - TLPlan’s pruning rules [Bacchus et al., 2000] have expressive power similar to SHOP’s
  - Using its pruning rules, they encoded a block-stacking algorithm similar to ours
Logistics

- 110 randomly generated problems
- Same machine as before
- As before, Blackbox and IPP could not solve any of these problems
- TLplan ran somewhat slower than SHOP (about an order of magnitude on large problems)
Logistics (Continued)

- 30 problems from the Blackbox distribution
- SHOP and TLplan on the same machine as before
- Blackbox on a faster machine, with 8GB of RAM
- SHOP was about an order of magnitude faster than TLplan
- TLplan was about two orders of magnitude faster than Blackbox
UM Translog

- HTN planning domain [Andrews et al., 1995]
  - Inspired by the logistics domain, but about an order of magnitude larger
  - Several different kinds of packages, vehicles, and procedures for handling the packages
    - e.g., hazardous materials need special handling
  - Several dozen primitive operators
  - Several dozen methods
  - Typical problems: dozens of vehicles, dozens of packages

- SHOP versus UMCP
  - UMCP [Erol, 1994] is our “classical” HTN planner

- Couldn’t test the other planners
  - Difficult to translate this domain into an action-based format
    - Will say more about this later
100 randomly generated problems
- Same machine as before
- UMCP did much worse than SHOP
Digression: Translating HTN Problems into Action-Based Planning Problems

- Can we compare these results with those of action-based planners (IPP, Blackbox, etc.)?

Problem:
- HTN planning is strictly more expressive than STRIPS-style planning (Erol et al. 1994)
- HTN problems from domains that include unbounded recursion among their methods cannot be expressed in STRIPS

However:
- UM-Translog does not include recursive methods at all
- For such cases, Amnon Lotem [Lotem & Nau, 2000] has defined an algorithm that translates an HTN domain into a STRIPS domain
HTN-to-STRIPS Algorithm

HTN Operator

Preconditions: p1, p2
Add Effects: e1

STRIPS Operator

Preconditions: p1, p2
Add Effects: e1, Load-completed

Artificial predicate
HTN-to-STRIPS Algorithm

HTN Method

Preconditions

Door-open-completed (?t)
Load-completed (?p, ?t, ?o)
Door-closed-completed (?t)

Add Effects

Artificial operator

Load-top-completed (?p, ?t, ?o)

STRIPS Operator

Load-top (?p, ?t, ?o)
Door-open (?t)
Load (?p, ?t, ?o)
Door-closed (?t)
Translating UM-Translog into STRIPS

Lotem used the algorithm to translate a subset of the UM-Translog domain into a STRIPS-style representation. There were several problems with that approach:

- Poor readability of the resulting domain (artificial operators and predicates)
- The artificial operators can be filtered out from the final solution, but we cannot guarantee minimal parallel length of the reported plan
- A huge number of instantiated actions. Only very simplified versions of the test problems could be solved using Blackbox and IPP.
Manual Encoding of UM-Translog

Guidelines:
- Do not create artificial operators
- Keep the domain compact (smaller number of instantiated actions)

Findings:
- It was difficult to express such a spec by using only operators.
- Many operators became context-specific to preserve order constraints.
- Domain was still large, although more compact than the previous one.
  - Could run IPP on some simple UM Translog problems, but not Blackbox.
AIPS-2000 Planning Competition

- Two tracks:
  - Track 1: STRIPS and PDDL planners
  - Track 2: “hand tailored” planners
    - many more problems, much harder problems

- We entered SHOP in Track 2

- We didn’t do much preparation for the competition
  - We felt very confident that SHOP would perform best

- We were wrong!
  - SHOP ran much faster than the Track-1 planners, but it wasn’t the best planner in Track 2
    - In two domains, we got incorrect results on a few problems
    - In the other problem domains, a new planner called TALplanner was faster
AIPS-2000 Planning Competition (Continued)

- Why we got the errors
  - We had to translate from PDDL to SHOP by hand
    - In two domains, we didn’t do the translation correctly
    - The errors were minor, but caused incorrect results
  - For next time, we are writing a PDDL-to-SHOP translator

- Why TALplanner was faster (we think)
  - Like TLplan, TALplanner does forward search guided by pruning rules written in modal logic
    - many of the same advantages as the SHOP algorithm
  - TALplanner uses highly optimized data structures
  - For next time, we intend to do the same for SHOP
    - Example: by making a simple modification to SHOP’s state representation, we made SHOP more than an order of magnitude faster
Related Publications


6. Evacuation Planning

Theory

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

Applications

- Joint work with David Aha, Leonard Breslow, and Héctor Muñoz-Avila
Noncombatant Evacuation Operations (NEOs)

Goal:
- Assist the US Dept. of State to evacuate people whose lives are in danger
  - noncombatants, nonessential military personnel, host-nation citizens, third-country nationals

Characteristics:
- Joint task force - geographically distributed, often multi-national
- Uncertainty
- Complexity (200+ tasks)
- US Ambassador is senior authority
- More than ten NEOs were conducted within the past decade
HICAP: Hierarchical Interactive Case-based Architecture for Planning

**Motivation**
- Interactive decision support tool - user in control
- Guided by military doctrine
- Provide access to previous experiences
  - Plan elicitation by reusing cases
  - Plan revision by reusing lessons learned
- Inferencing on standard procedures
- Perform bookkeeping
  - Reduce the chances of error
  - Help secure accountability
Why HTN Planning?

- Hierarchical planning is natural for military organizations
**HICAP**: Hierarchical Interactive Case-based Architecture for Planning

**Mixed-initiative planner**
- Interactive case retrieval: Select among alternative task decompositions
- Generative planning: Automated task component
- Resource conflict identification and resolution

**Input**: Doctrine & resources, cases, methods, lessons
- HTN representation

**Output**: Elicited plan
Hierarchical Task Editor

**HICAP : HTE**

- Task Hierarchy
- Resource Hierarchy

**HTE (i.e., manual)**
1. Apply Doctrine
2. Manual Edits

**NaCoDAE/HTN**
3. Apply Cases

**SHOP**
4. Apply Methods

**Task Decomposition**
Example SHOP Methods

- Decompose tasks into (more tactical) subtasks
- Consider restrictions (e.g., transport helicopters available)
- Resolve interactions (e.g., deploy security force first)
- If necessary, backtrack and try other methods

Select Helicopter Launching Base
- Select possible area (A)
  - Transport security force (F,A,H)
  - Embark security force (F,H)
  - Fly (H,A)
  - Disembark (F,H,A)
  - Position security force (F,A)
  - Transport fuel to (A)

Establish Base within Flying Distance
- Transport helicopters available (H)
- Security force available (F)

Launch from Carrier Battle Group
- Transport helicopters available (H)
- Helicopters have air refuel. capability (H)

alternative methods...
NaCoDAE/HTN: Navy Conversational Decision Aids Environment

- NaCoDAE/HTN implements Conversational Case Based Reasoning

- NaCoDAE/HTN allows interactive elicitation of situations from the user

Alternative decompositions (cases) for the task ranked according to the user’s answers

System’s Situation Knowledge

User

<Q,A> process

selects case
Methods Versus Cases

**SHOP**

*Methods* denote generic task decompositions and *conditions* for selecting those decompositions:

- **Task:** `travel(From,To)`
- **Decomposition:**
  - `travelC(From, Station1)`
  - `travelIC(Station1,Station2)`
  - `travelC(Station2, To)`
- **Conditions:**
  - `in(To,City1)`
  - `in(To,City2)`
  - `trainStation(Station1,City1)`
  - `trainStation(Station2,City2)`

**NaCoDAE/HTN**

*Cases* denote concrete task decompositions (taken from previous plans) and *preferences* for selecting those decompositions:

- **Task:** `travelC(Penn ST.,Downtn NYC)`
- **Decomposition:**
  - `take(taxi, Penn St., Downtn NYC)`
- **Questions:**
  - Is it raining hard in NYC? No
  - Is the passenger carrying luggage? No
Using NaCoDAE/HTN

Planning tasks

World State
In a mixed-initiative manner either SHOP or NaCoDAE/HTN has control over the decomposition process and cedes control to the other if it cannot proceed.
SiN: Algorithm

- Let $t$ be the task currently being decomposed
- If $t$ is primitive return a success
- Else if SHOP is in control
  - SHOP decomposes $t$ if possible. If SHOP cannot decompose $t$ but NaCoDAE/HTN can, then cede control to NaCoDAE/HTN.
- Else if NaCoDAE/HTN is in control
  - NaCoDAE/HTN decomposes $t$ if possible. If NaCODAE/HTN cannot decompose $t$ but SHOP can, then cede control to SHOP.
- Else if neither SHOP nor NaCoDAE/HTN can decompose $t$
  - Backtrack if possible. If it is not possible, then return a failure.
**SHOP**

*Methods* denote generic task decompositions and *conditions* for selecting those decompositions:

**Task:** travel(From, To)

**Decomposition:**
- travelC(From, Station1)
- travelIC(Station1, Station2)
- travelC(Station2, To)

**Conditions:**
- in(To, City1)
- in(To, City2)
- trainStation(Station1, City1)
- trainStation(Station2, City2)

---

**NaCoDAE/HTN**

*Cases* denote concrete task decompositions and *preferences* for selecting those decompositions:

**Task:** travelC(Penn ST., Downtn NYC)

**Decomposition:**
- take(taxi, Penn St., Downtn NYC)

**Questions:**
- Is it raining hard in NYC? No
- Is the passenger carrying luggage? No
SiN: Example – Mixed Initiative

In control

NaCoDAE/HTN

Travel(Greenbelt, NYC)

Travel(Greenbelt, Union St.)

Taxi(Greenbelt, GreenbeltM)

Metro(GreenbeltM, Union St.)

Train(Union St., Penn St.)

Travel(Penn St., Downtn NYC)

Taxi(Penn St., Downtn NYC)
Evaluating HICAP

- Use HICAP to plan a NEO subtasks
  - compare performance with other approaches
- How to evaluate? Can’t actually carry out a NEO
  - ModSAF - simulation program developed by the US Army
    - Simulates real-world military scenarios
    - Finite state simulation with modular components that represent individual entities and parts of entities
      - Example: simulated tank
        - Physical components (turret, etc.) and behavioral components (move, attack, target, react to fire, etc.)
    - Certain 3D aspects are also represented that can affect sensory and movement behavior
      - terrain elevation, trees, vegetation, rivers, oceans, atmospheric conditions, etc.
  - Realism is sufficient for training exercises
Snapshot of the MoDSAF Simulator
Evaluation: Empirical Methodology

Blind study: HICAP user did not know the simulated scenario

NEO Subtask: Transport 64 evacuees from meeting site to embassy
  • Meeting site at forested crossroads outside of city
  • Terrain database from Camp Lejeune

Transport by | Escorted?
--- | ---
Land | No
Land | Yes (tanks)
Air | No
Air | Yes (attack helicopters)

Selection Method:
1. HICAP
2. Most frequently used plan
3. Heuristic Choice (escort avg.)
4. Random

We used the same performance measures used by the US armed forces:
  • Casualties (Evacuees, friendlies, hostiles)
Two Scenarios and Average Results

(ModSAF is non-deterministic ⇒ 10 Runs)

Evacuees Saved:  Friendly Casualties:  Hostile Casualties:

Scenarios: 8 2-person dismounted hostile teams:
1. Anti-tank missiles, or
2. Anti-aircraft missiles
### Related Work

<table>
<thead>
<tr>
<th>System</th>
<th>Generative</th>
<th>Case-based</th>
<th>Mixed-initiative</th>
<th>Interleaved</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHOP</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHEF</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prodigy/Analogy</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIPE II</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>NaCoDAE/HTN</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>MI-CBP</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SiN</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Job Announcement

- We want to hire a full-time research scientist to work on extensions to HICAP
  - Research areas:
    - mixed-initiative planning, case-based reasoning, generative planning, knowledge management, and machine learning
  - The work will be joint between the University of Maryland and NRL's Intelligent Decision Aids Group
  - Ideal applicants will have some relevant research experience/interest, and Java software development expertise
    - Experience with simulators, military applications, and/or Smalltalk would also be welcome
  - The position is available now
- If you’re interested, please let me know!
Related Publications

[Muñoz-Avila et al., 1999a]

[Muñoz-Avila et al., 1999b]

[Muñoz-Avila et al., 2000]
Ordered task decomposition is an adaptation of HTN planning:
- Prolog-style left-to-right search; high degree of expressivity

Grew out of our work in two application domains:
- Manufacturing planning and the game of bridge

SHOP: domain-independent planning algorithm:
- Sound and complete over a wide class of problems
- Powerful enough for use in complex applications
  - Evacuation planning
Conclusions

- For a long time, AI planning researchers thought that forward search was not a good way to generate plans
  - Recent results (SHOP, TLplan, FF) strongly suggest otherwise

- AI planning is becoming capable enough to be used in real-world applications

- Synergy between theory and applications
  - Understanding what works in practical planning situations can produce better planning theories
  - This will lead to better real-world planners
Future Work

- Improvements to SHOP
  - optimize the data structures
  - PDDL-to-SHOP translator
- Planning with incomplete information
  - Contingency plans, evaluation of alternatives
    - Somewhat like the game-tree search we did in Bridge Baron
- Semi-automated knowledge acquisition
  - Case-based reasoning to synthesize HTN methods in HICAP, by observing what plans the human user creates in what situations
- Multi-agent planning
  - Integrate SHOP with the IMPACT multi-agent environment
Acknowledgements, and Where to Get More Information

- **Sponsors**
  - Government: NSF, NRL, ARL, DARPA
  - Industry: Great Game Products, Northrop Grumman

- **Where to get more information**
  - Computer Bridge
  - EDAPS
  - SHOP
  - HICAP