Multi-Unit Planning with HTN and A*

William van der Sterren
william@cgf-ai.com
Take Away

HTN planners are great for problems with hierarchy
- planning for multiple resources or units

HTN planners can be made to find a 'best' plan...
... when you make planning similar to A* path-finding
- Start with an initial top-level plan and expand it
- Expand until the plan solely consists of primitive tasks
- Methods generate multiple alternative successor plans
- Estimate costs of (partial) plans based on world state
- Expand least cost partial plan first

Combinatorial explosion can be avoided to some extent
Contents

- Take away
- 'PlannedAssault' application and multi-unit planner
  - example, requirements + scope
- Planning for multiple units with HTNs and A*
  - HTNs and Task Expansion
  - Plan Costs: Duration, Risk, Preference
  - Planner Methods and A*
  - Efficiency: Explicit Binding, Reverse Planning
- Summary and Other Work
- Questions?
PlannedAssault is a web-based mission generator creating large scale combined-arms battles for the ArmedAssault (ArmA) game.
The planner takes high-level objectives and units as input and generates the battle as a downloadable ready-to-play mission.

```json
{:objectives=>
    {
"Amy"=>
    {
:type="Hold",
:description="Hold this position at all cost.",
:position=>[5575.0, 11212.5]}}},

{:player_role=>'commander',
:player_side=>'WEST',
:duration=>1800,
:player_unit=>'Alpha',
...

"Golf"=>
    {
:type=>"rotary_wing",
:vehicles=>2,
:tags="armed,heavy_AT,light_AT,light_armor,rotary_wing",
:description="Section of two AH-1Z SuperCobra gunships AGM114/HE.",
:passenger_slots=>0,
:position=>[4750.0, 11981.2],
:size=>2,
:elements=>
    {
"Golf00"=>
    {
resource=>"ACE_AH1W_AGM_HE",
:comment="AH-1Z SuperCobra gunship AGM114/HE",
:rank="LIEUTENANT",
:leader=>true},
...}
PlannedAssault Example - 3/5

2 minute movie showing a synchronized attack resulting from PlannedAssault mission planning

Note:

- The movie primarily looks cool because the game (Bohemia Interactive’s ArmedAssault) does a great job rendering large scale battlefields
- However, ArmedAssault is lacking easy to use means to populate the battlefield with large scale synchronized combined-arms maneuvers
PlannedAssault Example - 4/5

Multi-Unit Planning with HTN and A*

June 11, 2009
AIGameDev Paris Game AI Conference
Planned Assault Example - 5/5

Multi-Unit Planning with HTN and A*

Russian_assault_of_Loliso

mission_A_B_C_D_E_F_G_H_I_L

objective_A_B_C_D_E_F_G_H_I_L

team_A_B_C_D_E

Move

Attack

FormationGroundAttack

tactics_A_B_C_D_E

PlannedFireSupport

tactics_G_H_I_L

unit_A

MountRideTransport

Move

Attack

unit_B

MountRideTransport

Move

Attack

unit_C

Ride

Move

Attack

unit_D

Attack

Move

unit_E

Move

Attack

unit_F

Move

Attack

unit_G

Move

Attack

unit_H

Attack

unit_I

Attack

unit_J

Attack

unit_L

Attack

ArtifactFireSupportMission

RotarywingCloseAirSupport

Mission

Clear

Regroup

TransportedMove

TransportedMove

PlannedFireSupport
Planner Requirements

- Plans
  - Good Solution, not just Any Solution

- Planning
  - Efficient, < 500 steps
  - Useful Mission Briefing and Planner Failure Messages
  - Make Method-writing Easy
  - Forward and (Military Style) Reverse Planning

- Scope
  - Off-line planning
  - No contingencies
Requirements: Any Plan vs a Good Plan 1/2

The 'Plan'

- Clear
- Move
  - FormUp
  - FormationAttack

objective area
Requirements: Any Plan vs a Good Plan 2/2

- Any

- Good

- Same 'plan'
- Different quality

- Objective area
- Form-up area
Planner Design

- Hierarchical Tasks Networks
  - Top-down planning reflects problem domain
- Search in Plan Space
  - Refine Partial Plans until Complete
- Assign Cost to (Partial) Plans
  - Using World-State $\Delta +$ risk $+$ preferences
- Planner Methods
  - Operating at different scopes
- Efficiency
  - Explicit Binding: DIY value assignment to variables
  - Task Input-output relations guide planning
Hierarchical Task Network Planning

Text books say:

- Planning: decomposing compound tasks into smaller tasks until the plan solely consists of primitive tasks
- Objective: formulate as a top level task rather than a set of states & conditions
- Methods do the decomposition if they apply to a task

HTN is often illustrated with down and forward, depth first planning in state space. However, HTN is not limited to that...
Task Refinement

Transported Move

A TransportedMove task is refined into a set of partially ordered tasks for transporter unit Alpha and passenger Bravo.
A multi-unit TeamMove partial plan is detailed into three alternative plans, involving combinations of TransportedMove and simple Moves. One plan is complete, two others remain partial plans.
Task Refinement - Heuristic

Heuristic:
Expand task at higher abstraction first! Doing so offers more insight in the plan's total utility or cost, and leads to more efficient search.
Plan Space: Keep track of World State

If we can define world states for the task start and task end, we’re able to:

- compute costs based on the difference in world state

World state = unit state, threat intel
Plan Cost

Cost of Task =
- Task duration +
- Task risk -
- Task preference

Task duration =
- Task duration, if primitive and inputs available
- Max (children’s end time) - Min(children’s start time), if compound and detailed
- Lower-bound estimate otherwise
Plan Cost Examples

Estimate costs of partial plan TransportedMove

- Move passenger from pA to pB with transport starting at tA and moving to tB

Cost estimate =

- path(tA,pA) + path(pA,pB) + path(pB,tB)

assuming movement at transport’s max speed

- preferences for transported move
# setup initial plan containing a
# MissionTask with objectives, units and threat intel.
...
# start planning
while !done
    current = most promising open node (lowest cost)
    done    = current.nil? || current.plan.complete?
    if !done
        t  = current.plan.task_to_expand
        ms = methods.select { |m| m.applies_to?(t) }
        neighbors = ms.collect do |m|
            m.generate(t).collect do |pars|
                m.plan(current.plan, t, pars)
            end
        end
        neighbors.flatten.compact
        neighbors.each { |n| nodes.push(Node.new(n, n.cost))}
        current.close
    end
end
## PlannerMethod examples

<table>
<thead>
<tr>
<th>scope</th>
<th>methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>mission</td>
<td>OneTaskForceSequentially, TwoTaskForcesParallel, ...</td>
</tr>
<tr>
<td>objective</td>
<td>Clear, Occupy, Defend, ...</td>
</tr>
<tr>
<td>team</td>
<td>Defend, DefendStatic, DefendWithCounter, Attack, Move, Regroup, Assemble, ...</td>
</tr>
<tr>
<td>tactics</td>
<td>FormationGroundAttack, PlannedFireSupport, OnCallFireSupport</td>
</tr>
<tr>
<td>units</td>
<td>DefendSector, HideInSector, TransportMove</td>
</tr>
<tr>
<td>unit</td>
<td>DefendSector, HideInSector, ArtilleryFireMission, RotaryWingCAS, FixedWingCAS</td>
</tr>
</tbody>
</table>

### task primitives

- Move, Attack, Hide, Wait, Mount, Unmount, Load, Unload, Ride
- ArtilleryFireSupportMission, RotaryWingCAS, FixedWingCAS
- ArtilleryFireSupportOnCall, RotaryWingCASOnCall, FixedWingCASOnCall
Binding

STRIPS planners typically have (implicit) binding

Implicit binding may become a problem when you:
- Allow a large range of values (16,000 waypoints)
- Write most of your planner code to constrain bindings
- Do lots of branching to find a good plan among all plans

Work-arounds
- Procedural pre-conditions (SHOP, GOAP)
- Abstract/reduce world state values (GOAP)
- Explicit binding (next slide)

Preconditions:
location(loc);
near_passenger(pas, loc);
accessible_to_transport(tr, loc);
accessible_to_dismount(pas, loc);
Explicit Binding

For free variables
- Compute best (or good enough) value, if possible
  - 1 plan
- Or generate a set of alternatives
  - n alternative plans

Example
- MoveTeam transport/passenger allocation
  - 3 transports, 5 passengers = 136 unique assignments
  - Explicit generation offers option to rate and select top N assignments for transport/passenger distance
PlannerMethod example

class TeamMoveMethod < PlannerMethod
  def initialize()
    # insert your code here
  end
  def test(o); o.kind_of?(MoveTeamTask) && o.ready?; end
  def plan(o, movement_combinations)
    # Generate subtasks and set the o MoveTeamTask's output.
    # A movement combination is a list of lists of units travelling together.
    # If the units' list contains more than 1 unit, the first units is
    # the transport and the remainder are cargo.
    unit_names = o.start_state.collect { |u| u.name }
    movement_combinations.each do |move_team|
      if move_team.size == 1
        ix = unit_names.index(move_team.first.name)
        um = MoveTask.new do |um|
          um.parent o
          um.start_state = o.start_state[ix]
          um.target_state = o.target_state[ix]
          um.end_state = um.target_state
        end
      else
        # first unit is transport, all other units are cargo
        ix = move_team.collect { |u| unit_names.index(u.name) }
        ix = ix.shift
        tm = TransportedMoveTask.new do |tm|
          tm.parent o
          tm.start_transport_state = o.start_state[ix]
          tm.target_transport_state = o.target_state[ix]
          tm.start_passengers_state = o.start_state.values_at(*ix)
          tm.target_passengers_state = o.target_state.values_at(*ix)
        end
      end
      o.end_state = o.target_state
    end
  end
  def generate(g, o)
    # Generate combinations of transport units, cargo units for combined moves, and
    # individual moves, taking into account transport's cargo_slots and passenger's size.
    generate_movement_combinations(o.start_state)
  end
private
  def generate_movement_combinations(units)
    transports, cargos = units.select { |u| u.transporter? }, units.select { |u| u.cargo? }
    others          = units - transports - cargos
    generate_move_combos(transports, cargos).collect { |lls| lls + others }
  end
  def generate_move_combos(transports, cargos)
    # insert your code here
  end
end

# Generate subtasks and set the o MoveTeamTask's output.
# A movement combination is a list of lists of units travelling together.
# If the units' list contains more than 1 unit, the first units is
# the transport and the remainder are cargo.

Note: Ruby code: 10x slower than C++, but great for prototyping
3-1. General.

a. Successful air assault execution is based on a careful analysis of METT-T and detailed, precise reverse planning. Five basic plans that comprise the reverse planning sequence are developed for each air assault operation. They are:

- The ground tactical plan.
- The landing plan.
- The air movement plan.
- The loading plan.
- The staging plan.

These plans should not be developed independently. They are coordinated and developed concurrently by the AATF staff to make best use of available time. The ground tactical plan is normally developed first and is the basis from which the other plans are derived.
When a Method rewrites a partial plan into a more detailed plan, it may link task inputs and outputs. Task readiness for detailing depends on the availability of inputs. These input relations enable ‘military-style reverse planning’ for part of the plan, which leads to more efficient planning and planner methods resembling real-world planning.

Pre-condition for this is world-state is communicated between tasks.
Task Refining w Input Dependencies 1/5
Task Refining w Input Dependencies 2/5

- ClearObjective
- TeamAttackAfterFormUp
- TeamAssemble
- TeamRegroup
- TeamMove
- FormationGroundAttack

Diagram showing task dependencies and flows in a game environment.
Task Refining w Input Dependencies 3/5
Task Refining w Input Dependencies 4/5
Task Refining w Input Dependencies 5/5

Multi-Unit Planning with HTN and A*
Summary

- Off-line Planning Combined Arms Attacks and Defense
  - Requires planning and a ‘best plan’

- Search in Plan Space using HTN for best plan
- Plans have a cost based on duration, preferences & risk
- A* ‘best-first’ search
- Plans are expanded at highest hierarchy first
- Explicit ‘binding’ to control performance and branching
- Explicit input/output binding allows forward and reverse planning
What are others doing?

- **Military Folks**

- **Academia**
  - Hierarchical Plan Representations for Encoding Strategic Game AI - Hoang, Lee-Urban, Munoz-Avila, AIIDE 2005
  - SquadSmart: Hierarchical Planning and Coordinated Plan Execution for Squads of Characters - Gorniak, Davis AIIDE 2007

- **Games**
  - The Creative Assembly’s Empire Total War
    Uses GOAP (multi-agent, multiple levels?)
  - Panther Games’ Airborne Assault / Command Operations series
    Very advanced, uses multi-agent hierarchical planning?

Note: multi-agent ‘distributed’ planning a must for on-line planners to efficiently repair plans
The Creative Assembly’s Empire Total War

Now we’re using a system of goal-oriented action planning, or GOAP, which basically means that the AI is constantly looking at the status of forces on the battlefield, which of course, is resources, and it’s got a list of jobs. Like, imagine you’ve got a list of post-its on your desk, and this is the number one thing I’ve got to do today, and this is the second. It moves those things around based on what’s happening. So let’s say its general is safely at the back of its army, right? And you’re fighting very well and suddenly you outflank him. You come right around from the back, and there’s a real chance that its general is in danger. The post-it note that says "Protected the general" goes from seventh to first, and then it looks at its resources and goes, "I’ve got artillery crews, I’ve got cavalry, I’ve got infantry. How do I achieve that objective now?" So what that means is you get a lot more dynamic backward and forward gameplay with the AI reacting to the things you’re doing in real time, rather than going "I’ve got to do A, then I’ve got to do B," and if you interrupt it anywhere in that cycle, it doesn’t get to the next step. Now, it’s constantly looking at what its priorities should be to achieve its objectives and basing its movements and its actions around that exact thing.

CA’s Kieran Brigden in an interview, Dec 01, 2008
Panther Games’ Airborne Assault series

- Screenshot: Highway to the Reich, 2003

- Game ‘Command Operations: Battles for the Bulge’ scheduled for 2009H2 with more advanced planning (bridge-building, plan scheduling)

Smart Organisations

- Command Structure
- Subordination
- Human Player must be able to TRUST that an AI controlled subordinates will do a reasonable job of managing itself and its subordinates
  - Hence, build AI from the bottom up
    - Basic mechanics
    - Simple tasks/plans
    - Complex tasks/plans
- Organic, Player and Current Structures
- Effect changes as part of order process

Spiral Plan Development

- Recursive plan development
- Decomposing complex tasks
- Eg. Objective = Secure X
  - Am I there yet? No, then Move to X
  - Am I threatened or blocked? Yes, then Attack.
  - Am I there yet? Yes, then Defend.

Presentation by Dave O’Connor, Canberra, May 2008?
Plan Cost Examples (2/2)

Mission

Clear B → Clear A

Cost Estimate =

- Min of Path(u, objective).length / speed, for all ground combat units u traveling at max speed
- + risk of traveling through hostile terrain
Multi-Unit Planning with HTN and A*