Automated Planning and Scheduling

Lecture 2: Complexity of Planning

Tomáš Balyo, Dominik Schreiber | October 23, 2019
Planning: How hard can it be?

In a nutshell: Find a sequence of actions in order to get from a certain initial state to some state which satisfies certain objectives (goals).
Planning: How hard can it be?

In a nutshell: Find a sequence of actions in order to get from a certain initial state to some state which satisfies certain objectives (goals).

- How complex is this?
- What does the complexity depend on?
- How does the complexity translate to real-life planning tasks?

The following slides are partly based on *The Computational Complexity of Propositional STRIPS Planning* [Byl94].
More Definitions (1)

Atom, State, Action

- An *atom*, or *proposition*, $p$ is a variable which can be *true* or *false*. We write a false atom as $\bar{p}$.
- A *state* $s$ is a subset of all true atoms.
- In an *action* $a = (\text{pre}(a), \text{eff}(a))$, $\text{pre}(a)$ and $\text{eff}(a)$ are sets of atoms.
- For any set of atoms $M$, we write $M^+$ for all *true* atoms in $M$ and $M^-$ for all *false* atoms in $M$. 
Application of an Action

- An action \( a \) is *applicable* in a state \( s \) iff \( \forall p \in \text{pre}(a) : p \in s \) and \( \forall \bar{p} \in \text{pre}(a) : p \notin s \).
- If an action \( a \) is applicable in a state \( s \), we denote the *application* of \( a \) to \( s \) as
  \[
  s' = \gamma(s, a) := s \cup \text{eff}(a)^+ \setminus \text{eff}(a)^-.
  \]
- For a sequence of actions \( p = \langle a_1, \ldots, a_n \rangle \), we recursively define the application of \( p \) to \( s \) as
  \[
  \overset{n}{\gamma}(s, \langle \rangle) := s \tag{1}
  \\
  \overset{n}{\gamma}(s, \langle a_1, \ldots, a_n \rangle) := \overset{n}{\gamma}(\gamma(s, a_1), \langle a_2, \ldots, a_n \rangle) \tag{2}
  \\
  (\text{provided that } \gamma \text{ is well-defined for each } a_i).
Planning Problem (Goal as set of *atoms*)

Given
- a set of states \( S \),
- a set of actions \( A \),
- a state \( s_I \in S \), and
- a set of positive atoms \( G \),

\[ \pi = (S, A, s_I, G) \] is a planning problem.

Plan (Goal as set of *atoms*)

A sequence of actions \( p \in A^* \) is a plan (or solution) for \( \pi \) iff \( G \subseteq \hat{\gamma}(s_I, p) \).
Complexity Preliminaries

Important complexity classes

- **P**: Problems deterministically solvable in polynomial time
- **NP**: Problems non-deterministically solvable in polynomial time
- **PSPACE**: Problems deterministically solvable in polynomial space

A problem $P$ is **$X$-hard** iff every problem in $X$ can be reduced to $P$.
A problem $P$ is **$X$-complete** iff $P$ is $X$-hard and $P \in X$.

Decidability problems for planning

- **PLANSAT**: Given a planning problem, is there a solution?
- **PLANMIN**: Given a planning problem and $k \geq 0$, is there a solution with at most $k$ steps?
The Complexity of Classical Planning (1)

Theorem.

PLANSAT is PSPACE-complete.
The Complexity of Classical Planning (1)

Theorem.

\textbf{PLANSAT} is \textbf{PSPACE}-complete.

A proof has two parts:

1. Show that \textbf{PLANSAT} \in \textbf{PSPACE}
   - Propose an algorithm deciding \textbf{PLANSAT} which “only” takes polynomial space

2. Show that every problem in \textbf{PSPACE} can be reduced to \textbf{PLANSAT}
   - If a \textit{Turing} machine on polynomial space can solve a problem, show that we can solve the problem with classical planning
Proof for $\text{PLANSAT} \in \text{PSPACE}$:

- Theorem of Savitch: $\text{PSPACE} = \text{NPSPACE}$ [Sav70]
Proof for $\text{PLANSAT} \in \text{PSPACE}$:

- Theorem of Savitch: $\text{PSPACE} = \text{NPSPACE}$ [Sav70]
  - We can just prove $\text{PLANSAT} \in \text{NPSPACE}$

Why is a maximum plan length of $2^{|P|}$ always enough to check?

Why does the algorithm only take a polynomial amount of space?
Proof for $\text{PLANSAT} \in \text{PSPACE}$:

- Theorem of Savitch: $\text{PSPACE} = \text{NPSPACE}$ [Sav70]
  - $\Rightarrow$ We can just prove $\text{PLANSAT} \in \text{NPSPACE}$
  - $\Rightarrow$ a non-deterministic algorithm for $\text{PLANSAT}$ suffices
The Complexity of Classical Planning (2)

Proof for PLANSAT $\in$ PSPACE:

- Theorem of Savitch: PSPACE $=$ NPSPACE [Sav70]
  $\Rightarrow$ We can just prove PLANSAT $\in$ NPSPACE
  $\Rightarrow$ a non-deterministic algorithm for PLANSAT suffices

- Easy non-deterministic PLANSAT decision procedure:
  - Choose some plan $p$ with $|p| \leq 2^{|P|}$ ($P$: set of all atoms)
  - Process $p$ step by step, and if it is a valid solution, output YES
The Complexity of Classical Planning (2)

Proof for \textbf{PLANSAT} \in \textbf{PSPACE}:

- Theorem of Savitch: \textbf{PSPACE} = \textbf{NPSPACE} [Sav70]

  \Rightarrow \text{ We can just prove } \textbf{PLANSAT} \in \textbf{NPSPACE}

  \Rightarrow \text{ a non-deterministic algorithm for } \textbf{PLANSAT} \text{ suffices}

- Easy non-deterministic \textbf{PLANSAT} decision procedure:
  
  - Choose some plan \( p \) with \(|p| \leq 2^{|\mathcal{P}|} \) (\( \mathcal{P} \): set of all atoms)
  
  - Process \( p \) step by step, and if it is a valid solution, output \text{ YES}

- Why is a maximum plan length of \( 2^{|\mathcal{P}|} \) always enough to check?

- Why does the algorithm only take a polynomial amount of space?
The Complexity of Classical Planning (3)

Proof for $\textbf{PSPACE} \preceq_p \textbf{PLANSAT}$:

- Show that any Turing machine with polynomial space bounds can be simulated by a planning problem!
- Given a TM, construct a planning domain such that
  - the TM terminates iff the planning problem has a solution
  - the planning domain has polynomial size w.r.t. the TM program
The Complexity of Classical Planning (3)

Proof for $\text{PSPACE} \preceq_p \text{PLANSAT}$:

- Show that any Turing machine with polynomial space bounds can be simulated by a planning problem!
- Given a TM, construct a planning domain such that
  - the TM terminates iff the planning problem has a solution
  - the planning domain has polynomial size w.r.t. the TM program
- Any ideas?
Planning a Turing machine

- **Configuration** of TM (position, tape, state) \(\sim\) **atoms**
  - \(at(position, symbol), reader-at(position), state(s)\)
- **Transitions** of TM \(\sim\) **actions**
  - Example:

\[
\text{PRE: } \{ \text{reader-at}(pos), \ at(pos, x), \ state(s) \} \\
\text{transition}(pos, x, s) \\
\text{EFF: } \{ \text{reader-at}(pos+1), \ at(pos, y), \ state(s') \}
\]

- **Termination** / acceptance of TM \(\sim\) **goals**
Planning a Turing machine

- **Configuration** of TM (position, tape, state) $\leadsto$ atoms
  - `at(position, symbol)`, `reader-at(position)`, `state(s)`
- **Transitions** of TM $\leadsto$ actions
  - Example:
    
    PRE: \{\textit{reader-at(pos)}, \textit{at(pos, x)}, \textit{state(s)}\}
    
    transition(pos, x, s)

    EFF: \{\textit{reader-at(pos+1)}, \textit{at(pos, y)}, \textit{state(s')}\}

- **Termination** / acceptance of TM $\leadsto$ goals
- How many atoms and actions are needed?

$S$ internal states, tape of length $L$, symbols each

$O(S + L \cdot \Sigma)$ atoms

$O(S \cdot L \cdot \Sigma)$ actions
Planning a Turing machine

- **Configuration of TM (position, tape, state)** → **atoms**
  - `at(position, symbol)`, `reader-at(position)`, `state(s)`

- **Transitions of TM** → **actions**
  - Example:
    
    PRE: `\{ reader-at(pos), at(pos, x), state(s) \}`
    
    `transition(pos, x, s)`
    
    EFF: `\{ reader-at(pos+1), at(pos, y), state(s') \}`

- **Termination / acceptance of TM** → **goals**

- How many atoms and actions are needed?
  - S internal states, tape of length L, Σ symbols each
  - $O(S + L \cdot Σ)$ atoms
  - $O(S \cdot L \cdot Σ)$ actions
A Turing machine in PDDL: Domain (1)

(define (domain TM)
  (:requirements :strips :typing :conditional-effects :equality)
  (:types position state symbol direction - object)
  (:constants left right stop - direction)
  (:predicates
    (reader-at ?p - position)
    (symbol-at ?p - position ?x - symbol)
    (state ?s - state)
    (program ?x - symbol ?s - state
      ?x2 - symbol ?s2 - state ?d - direction)
    (halted)
  )
  ...)
)
A Turing machine in PDDL: Domain (2)

(:action transition
 :parameters (?p ?p2 - position ?s ?s2 - state
 ?x ?x2 - symbol ?d - direction)
 :precondition (and
 )
 :effect (and
 (when (not (= ?p ?p2))
   (and (not (reader-at ?p)) (reader-at ?p2)))
 (when (not (= ?s ?s2))
   (and (not (state ?s)) (state ?s2)))
 (when (not (= ?x ?x2))
   (and (not (symbol-at ?p ?x)) (symbol-at ?p ?x2)))
 )
)
A Turing machine in PDDL: Domain (2)

(:action transition
  :parameters (?p ?p2 - position ?s ?s2 - state
  ?x ?x2 - symbol ?d - direction)
  :precondition (and
  )
  :effect (and
  (when (not (= ?p ?p2)) ← Conditional effect: Prerequisites, Effects
  (and (not (reader-at ?p)) (reader-at ?p2)))
  (when (not (= ?s ?s2))
   (and (not (state ?s)) (state ?s2)))
  (when (not (= ?x ?x2))
   (and (not (symbol-at ?p ?x)) (symbol-at ?p ?x2))))
  )
)
(:action halt
   :parameters (?p - position ?s - state ?x - symbol)
   :precondition (and
      (program ?x ?s ?x ?s stop)
   )
   :effect (and
      (halted)
   )
)
(define (problem TM-1)
  (:domain TM)
  (:objects
    p1 p2 p3 p4 p5 p6 pb - position
    s0 s1 s2 - state
    zero one blank - symbol
  )
  (:init
    (next-position p1 stop p1)
    (next-position p1 right p2)
    (next-position p2 left p1)
    (next-position p2 stop p2)
    (next-position p2 right p3)
    ...
A Turing machine in PDDL: Problem (2)

... ; symbol state | symbol state move
(program zero s0 zero s0 right)  \textit{Read from left to right until the first “1”;}  
(program one s0 one s1 right)  \textit{then write “1” until the right border;}  
(program blank s0 blank s2 left )  \textit{then stop.}  
(program zero s1 one s1 right)  
(program one s1 one s1 right)  
(program blank s1 blank s2 left )  
(program zero s2 zero s2 stop )  
(program one s2 one s2 stop )  

(:goal  
  (and (halted))  
)  
)
A Turing machine in PDDL: Plan

- Found plan:
  1: transition(p1 p2 s0 s0 zero zero right)
  2: transition(p2 p3 s0 s0 zero zero right)
  3: transition(p3 p4 s0 s0 zero zero right)
  4: transition(p4 p5 s0 s1 one one right)
  5: transition(p5 p6 s1 s1 zero one right)
  6: transition(p6 pb s1 s1 zero one right)
  7: transition(pb p6 s1 s2 blank blank left)
  8: halt(p6 s2 one)

- How can a non-deterministic TM be realized?
A Turing machine in PDDL: Plan

- Found plan:
  1: transition( p1 p2 s0 s0 zero zero right )
  2: transition( p2 p3 s0 s0 zero zero right )
  3: transition( p3 p4 s0 s0 zero zero right )
  4: transition( p4 p5 s0 s1 one one right )
  5: transition( p5 p6 s1 s1 zero one right )
  6: transition( p6 pb s1 s1 zero one right )
  7: transition( pb p6 s1 s2 blank blank left )
  8: halt( p6 s2 one )

- How can a non-deterministic TM be realized?
  - Just add multiple transition atoms for some of the state-symbol-combinations to the initial state
  - Nice analogy:
    TM is deterministic ⇔ A forward search through the problem’s state space is a linear path without branches (no decisions necessary!)
Now what?

- **PSPACE**-complete problems are hard. [citation needed]
Now what?

- **PSPACE**-complete problems are hard. \[\text{citation needed}\]
- What are possible responses to this (too) high complexity?
Now what?

- **PSPACE**-complete problems are hard.\[citation needed\]

What are possible responses to this (too) high complexity?

1. **Restrict our model** to make it easier (but still useful!)
   - We’ll try on the next slides
Now what?

- **PSPACE**-complete problems are hard.\(^{\text{[citation needed]}}\)

What are possible responses to this (too) high complexity?

1. **Restrict our model** to make it easier (but still useful!)
   - We’ll try on the next slides
2. **Work out heuristics and/or approximations** that work fine in practice
   - Upcoming lecture(s) on heuristics and search methods
Now what?

- **PSPACE**-complete problems are hard. \footnote{citation needed}

What are possible responses to this (too) high complexity?

1. **Restrict our model** to make it easier (but still useful!)
   - We’ll try on the next slides

2. **Work out heuristics and/or approximations** that work fine in practice
   - Upcoming lecture(s) on heuristics and search methods

3. **Drop our model’s generality** and develop planning algorithms which are **specialized** for the problem at hand
   - Partly being done in HTN planning (later lecture)
   - Feel free to take some robotics course covering motion planning
For now, consider planning problems where we restrict the amount of preconditions and effects per action

- Let $\text{PLANSAT}_p^q$ denote planning problems restricted to $p$ preconditions and $q$ effects
- Likewise, let $\text{PLANSAT}_{p^+}^q$ denote problems with $p$ positive preconditions (no negatives!) etc.
Model restrictions

- For now, consider planning problems where we restrict the amount of preconditions and effects per action
  - Let $\text{PLANSAT}_p^q$ denote planning problems restricted to $p$ preconditions and $q$ effects
  - Likewise, let $\text{PLANSAT}^q_{p+}$ denote problems with $p$ positive preconditions (no negatives!) etc.

- What about the complexity of deciding . . .
  - . . . $\text{PLANSAT}_1$, $\text{PLANSAT}^1$ (a single precondition/effect)?
  - . . . $\text{PLANSAT}^+$ (positive effects only)?
  - . . . $\text{PLANSAT}_0$ (no preconditions)?
Model restrictions (2)

- $\text{PLANSAT}_1$, $\text{PLANSAT}^1$: still $\text{PSPACE}$-complete
  - $\exists$ a transformation of a TM into a planning problem such that only a single precondition is needed per action
  - Same for a single effect (but not both restrictions at once!)
Model restrictions (2)

- $\text{PLANSAT}_1$, $\text{PLANSAT}^1$: still PSPACE-complete
  - $\exists$ a transformation of a TM into a planning problem such that only a single precondition is needed per action
  - Same for a single effect (but not both restrictions at once!)

- $\text{PLANSAT}^+$: NP-complete! (Yay ... I guess?)
  - When advancing in state space, the amount of true atoms will increase monotonously
  - A solution plan requires at most as many actions as there are atoms
Model restrictions (2)

- **PLANSAT\(_1\), PLANSAT\(^1\):** still PSPACE-complete
  - ∃ a transformation of a TM into a planning problem such that only a single precondition is needed per action
  - Same for a single effect (but not both restrictions at once!)

- **PLANSAT\(^+\):** NP-complete! (Yay ... I guess?)
  - When advancing in state space, the amount of true atoms will increase monotonously
  - A solution plan requires at most as many actions as there are atoms

- **PLANSAT\(_0\):** polynomial!
Model restrictions (2)

- **PLANSAT\(^1\), PLANSAT\(^1\)**: still PSPACE-complete
  - ∃ a transformation of a TM into a planning problem such that only a single precondition is needed per action
  - Same for a single effect (but not both restrictions at once!)

- **PLANSAT\(^+\)**: NP-complete! (Yay ... I guess?)
  - When advancing in state space, the amount of true atoms will increase monotonously
  - A solution plan requires at most as many actions as there are atoms

- **PLANSAT\(_0\)**: polynomial!
  - Pick an action \(a\) that does not counteract any goal atom
  - Put \(a\) at the end of the plan
  - Remove \(a\) from considered actions and \(\text{eff}(a)\) from goal atoms
  - Repeat until no more goals are left
Remember, **PLANMIN** ≡ “Given a planning problem and $k \geq 0$, is there a solution with at most $k$ steps?”
Remember, $\text{PLANMIN} \equiv \text{"Given a planning problem and } k \geq 0, \text{ is there a solution with at most } k \text{ steps?"}

We can interpret this as the \text{"optimal planning" problem}
Remember, **PLANMIN** ≡ “Given a planning problem and $k \geq 0$, is there a solution with at most $k$ steps?”

We can interpret this as the “optimal planning” problem

- Any decision algorithm must gain information on the minimum plan
- Beginning with $k = 0$ and then increasing $k$ one by one, we get the optimal plan length after solving **PLANMIN** repeatedly
Remember, PLANMIN ≡ “Given a planning problem and $k \geq 0$, is there a solution with at most $k$ steps?”

We can interpret this as the “optimal planning” problem

- Any decision algorithm must gain information on the minimum plan
- Beginning with $k = 0$ and then increasing $k$ one by one, we get the optimal plan length after solving PLANMIN repeatedly

Now what about the complexities of PLANMIN, and PLANMIN₁, and . . . ?
### Theorem

For each mentioned **PSPACE**-complete **PLANSAT** (sub)problem, the corresponding **PLANMIN** problem is **PSPACE**-complete, as well.
Theorem.

For each mentioned **PSPACE**-complete **PLANSAT** (sub)problem, the corresponding **PLANMIN** problem is **PSPACE**-complete, as well.

Idea of proof for such a **PLANMIN** problem being **PSPACE**-complete:
Complexity of Optimal Planning (2)

Theorem.

For each mentioned \textsc{PSPACE}-complete \textsc{PLANSAT} (sub)problem, the corresponding \textsc{PLANMIN} problem is \textsc{PSPACE}-complete, as well.

Idea of proof for such a \textsc{PLANMIN} problem being \textsc{PSPACE}-complete:

- Containment in \textsc{PSPACE}:
  - Again, non-deterministic choice of a plan
  - Plan length bounded by $k$
Theorem.

For each mentioned PSPACE-complete PLANSAT (sub)problem, the corresponding PLANMIN problem is PSPACE-complete, as well.

Idea of proof for such a PLANMIN problem being PSPACE-complete:

- **Containment in PSPACE:**
  - Again, non-deterministic choice of a plan
  - Plan length bounded by $k$

- **Hardness w.r.t. PSPACE:**
  - Solve any PLANSAT problem by using a PLANMIN solver and setting $k = 2^{|P|}$
  - Why does this work?
Theorem.

Take the previous theorem and replace each occurrence of PSPACE with NP. Then the theorem still holds.
Theorem.

Take the previous theorem and replace each occurrence of $\text{PSPACE}$ with $\text{NP}$. Then the theorem still holds.

Idea of proof:

- $\text{NP}$-complete $\text{PLANSAT}$ subproblems have clear, linear bound on needed plan length $\Rightarrow \text{PLANMIN}$ problems are solvable similarly.

- Reduce $\text{NP}$-complete $\text{PLANSAT}$ subproblems to $\text{PLANMIN}$ where $k$ is set to known upper bound on plan length.

One question remaining:

What about the polynomial $\text{PLANSAT}$ subproblems? Is optimal planning harder there?
Theorem.

Take the previous theorem and replace each occurrence of PSPACE with NP. Then the theorem still holds.

Idea of proof:

- NP-complete PLANSAT subproblems have clear, linear bound on needed plan length $\Rightarrow$ PLANMIN problems are solvable similarly
- Reduce NP-complete PLANSAT subproblems to PLANMIN where $k$ is set to known upper bound on plan length

One question remaining:

- What about the polynomial PLANSAT subproblems? Is optimal planning harder there?
Observation: For polynomial PLANSAT subproblems, the corresponding PLANMIN subproblem may still be NP-complete!

- Example: PLANMIN$_0$ (no preconditions)
- Decision procedure for PLANSAT$_0$: iteratively add “non-destructive” action to the front of the plan
- No guarantee for a plan of minimum length!
- In general, finding the minimum sequence of actions is (NP-)hard
Complexity of Optimal Planning (4)

Observation: For polynomial PLANSAT subproblems, the corresponding PLANMIN subproblem may still be NP-complete!

- Example: PLANMIN₀ (no preconditions)
- Decision procedure for PLANSAT₀: iteratively add “non-destructive” action to the front of the plan
- No guarantee for a plan of minimum length!
- In general, finding the minimum sequence of actions is (NP-)hard
  - Needed amount of actions can depend on execution order
  - Combinatorial amount of decisions!
Planning Complexity: Further Examples

- Classical (STRIPS) planning: \textbf{PSPACE}-complete (as just seen)
  - (Very) easy subproblems may be in \textbf{NP} or \textbf{P}
Planning Complexity: Further Examples

- Classical (STRIPS) planning: \textbf{PSPACE}-complete (as just seen)
  - (Very) easy subproblems may be in \textbf{NP} or \textbf{P}
- Planning models with additional \textit{domain theories}:
Planning Complexity: Further Examples

- Classical (STRIPS) planning: **PSPACE**-complete (as just seen)
  - (Very) easy subproblems may be in **NP** or **P**
- Planning models with additional domain theories:
  - at least **PSPACE**-complete [Byl94]
Classical (STRIPS) planning: \textbf{PSPACE}-complete (as just seen)

- (Very) easy subproblems may be in \textbf{NP} or \textbf{P}

Planning models with additional domain theories:
- at least \textbf{PSPACE}-complete [Byl94]

(Concurrent) Temporal planning: \textbf{EXPSPACE}-complete [R+07]
Planning Complexity: Further Examples

- Classical (STRIPS) planning: \textbf{PSPACE}-complete (as just seen)
  - (Very) easy subproblems may be in \textbf{NP} or \textbf{P}
- Planning models with additional domain theories: at least \textbf{PSPACE}-complete [Byl94]
- (Concurrent) Temporal planning: \textbf{EXPSPACE}-complete [R$^+$07]
- Hierarchical Task Network (HTN) planning, an extension to STRIPS: undecidable! [EHN94]
Planning Complexity: Further Examples

- Classical (STRIPS) planning: **PSPACE**-complete (as just seen)
  - *(Very) easy subproblems* may be in **NP** or **P**
- Planning models with additional domain theories: at least **PSPACE**-complete [Byl94]
- *(Concurrent) Temporal planning:* **EXPSPACE**-complete [R⁺07]
- Hierarchical Task Network (HTN) planning, an extension to STRIPS: **undecidable**! [EHN94]
- But of course, there are always practically feasible instances . . .
Conclusion

- Classical planning, in its general form, is **highly** complex
- Very significant restrictions are necessary to even reach \(\text{NP}\)-completeness
Conclusion

- Classical planning, in its general form, is **highly** complex
- Very significant restrictions are necessary to even reach **NP**-completeness
  - At this point, the model has lost a lot of expressive power
Conclusion

- Classical planning, in its general form, is **highly** complex.
- Very significant restrictions are necessary to even reach NP-completeness.
  - At this point, the model has lost a lot of expressive power.

Practical consequences:

- Domain-agnostic planning is only feasible up to certain input sizes.
  - Are our real-world problems “small enough”?
Conclusion

- Classical planning, in its general form, is **highly** complex
- Very significant restrictions are necessary to even reach \textbf{NP}-completeness
  - At this point, the model has lost a lot of expressive power

Practical consequences:

- Domain-agnostic planning is only feasible up to certain input sizes
  - Are our real-world problems “small enough”?
- Tackle lots of real-world problems using clever heuristics
Conclusion

- Classical planning, in its general form, is **highly** complex
- Very significant restrictions are necessary to even reach \textbf{NP}-completeness
  - At this point, the model has lost a lot of expressive power

Practical consequences:

- Domain-agnostic planning is only feasible up to certain input sizes
  - Are our real-world problems “small enough”?
- Tackle lots of real-world problems using clever heuristics
- When problems become too hard, maybe switch to a domain-specific / specialized approach
Stay tuned!

Next lecture (Next week):
State space, and forward search planning

