Planning Techniques

Dra. Mª Dolores Rodríguez Moreno

Objectives

Specific Objectives

- Main structures used
- Main techniques

Source

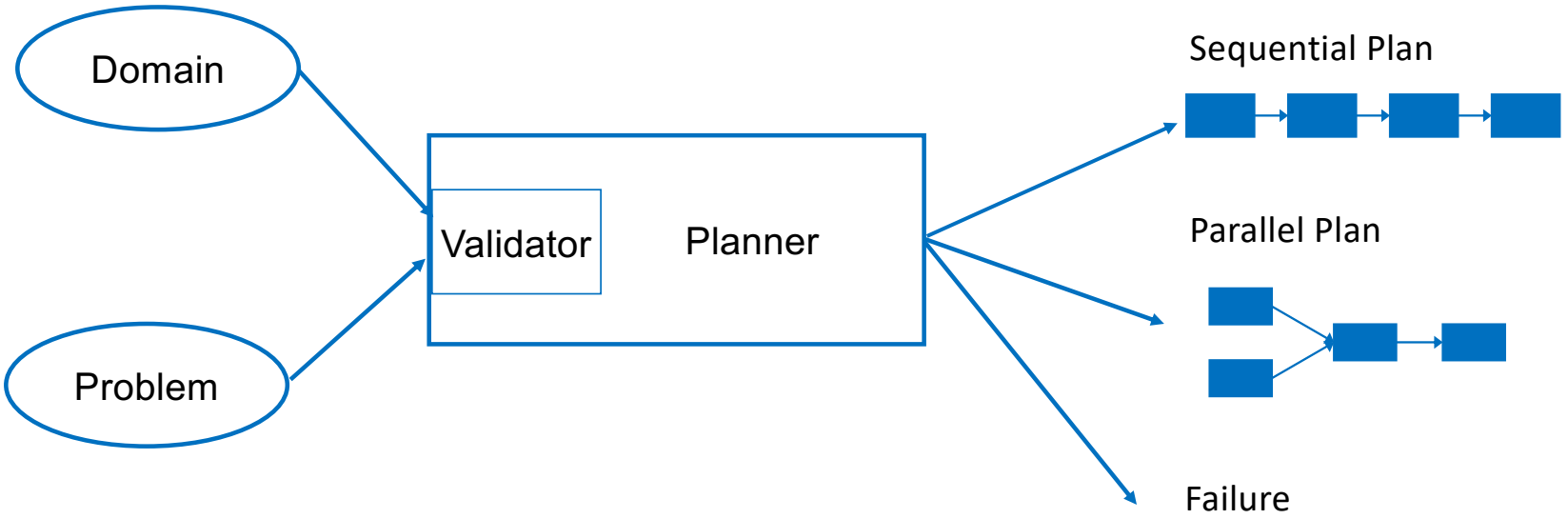
- Stuart Russell & Peter Norvig (2009). Artificial Intelligence: A Modern Approach. (3rd Edition). Ed. Pearsons
- Ghallab, Nau &Traverso (2004). Automated Planning: Theory & Practice. The Morgan Kaufmann Series in Artificial Intelligence
- Dana Nau's slides for Automated Planning. Licensed under License https://creativecommons.org/licenses/by-nc-sa/2.0/

Outline

- **Introduction**
- Classification of algorithms
- Planning techniques
- Conclusions

Introduction

- Nearly all planning procedures are search procedures



Outline

- Introduction
- **Classification of algorithms**
- Planning techniques
- Conclusions

Classification of algorithms

- Criteria for classifying algorithms:
  - How the search is performed (state/plan, progression/regression)
  - Goals ordering: (no)lineal
  - How plans are built: generative (no library plans) or case-based
  - Dealing with uncertainty: contingent and probabilistic
  - Using specific knowledge: domain (in)dependent

Classification of algorithms

- Criteria for classifying algorithms:
  - **How the search is performed (state/plan, progression/regression)**
  - Goals ordering: (no)lineal
  - How plans are built: generative (no library plans) or case-based
  - Dealing with uncertainty: contingent and probabilistic
  - Using specific knowledge: domain (in)dependent

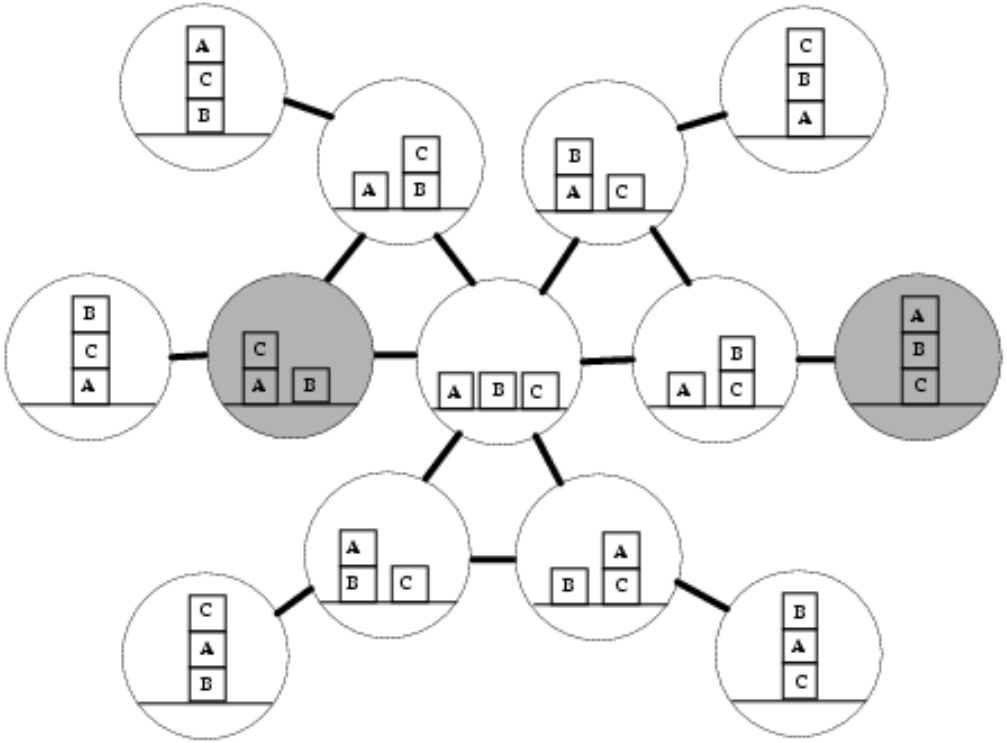
Classification of algorithms: search

- **State-space search in planning**
  - Each node represents a state of the world
  - A plan is a path through the space
- Plan-space search in planning
  - Each node is a set of partially-instantiated operators, plus some constraints
  - Impose more and more constraints, until we get a plan

State Space Search (SSS) (I)

- The easiest way to build a planner is to convert it in a search problem through the state space (SSS)
  - Each node in the tree/graph represents a state of the world
  - Each arc connects worlds that are achieved by executing an action
- Once the planning problem is converted → we can apply any algorithm studied

State Space Search (SSS) (II)



State Space Search (SSS) (III)

- You can incrementally generate the set of reachable states from the initial state by a sequence of actions: *progression*
- The states that are achieved in these sequences can be calculated
  - Checking if the goal has been achieved
  - Checking if the preconditions are satisfied
- Instead of looking forward from the initial state, you can search backward from the goals: *regression*

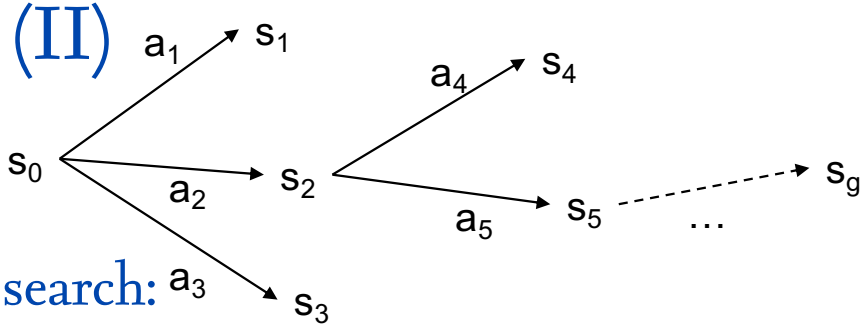
State Space Search (SSS) (IV)

- The algorithm is:
  - Robust? (the plan that returns, works)
  - Complete? (if there is a solution plan, can find it)
- What algorithm is faster?
  - They have the same complexity, both will do approx. ”n" decisions before finding the solution
  - The number of decisions at each branch point suppose to be "b”. Complexity??
  - By experience, the branching factor (b) is smaller in one. What????

State Space Search (SSS): Progression (I)

- Forward-search is *sound*
  - for any plan returned by any of its nondeterministic traces, this plan is guaranteed to be a solution
- Forward-search also is *complete*
  - if a solution exists then at least one of Forward-search's nondeterministic traces will return a solution

State Space Search (SSS): Progression (II)



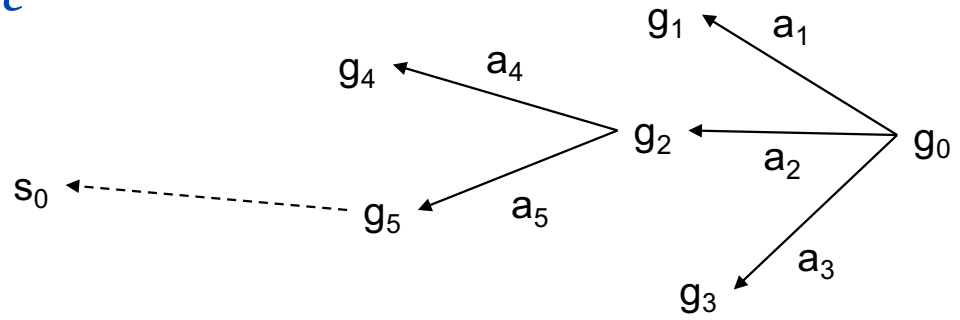
- Some deterministic implementations of forward search:
  - breadth-first, depth-first, best-first (e.g., A*) or greedy search
- Breadth-first and best-first search are sound and complete
  - But they usually aren't practical because they require too much memory
  - Memory requirement is exponential in the length of the solution
- In practice, more likely to use depth-first or greedy search
  - Worst-case memory requirement is linear in the length of the solution
  - In general, sound but not complete

State Space Search (SSS): Progression (III)

- Forward search can have a very large branching factor
  - E.g., many applicable actions that don't progress toward goal
- Why this is bad:
  - Deterministic implementations can waste time trying lots of irrelevant actions
- Need a good heuristic function and/or pruning procedure

State Space Search (SSS): Regression (I)

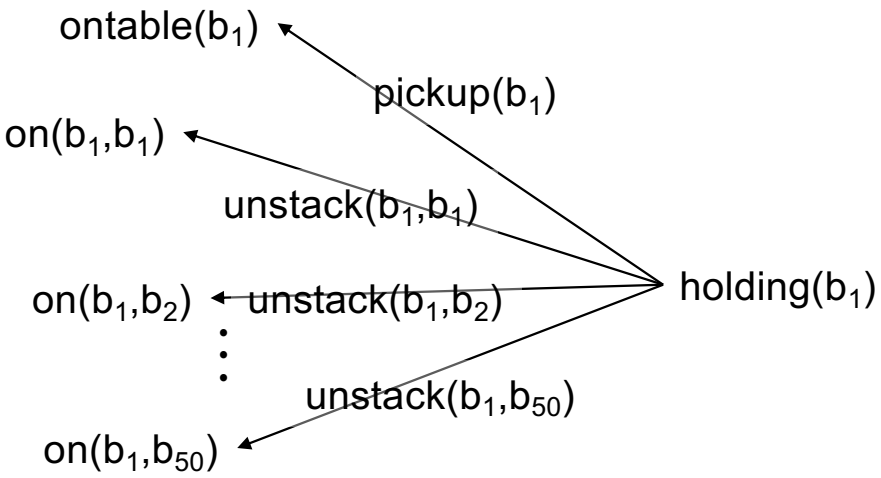
- For forward search, we started at the initial state and computed state transitions
  - new state = $\gamma(s,a)$
- For backward search, we start at the goal and compute inverse state transitions
  - new set of subgoals = $\gamma^{-1}(g,a)$
- To define $\gamma^{-1}(g,a)$, must first define *relevance*
  - An action a is relevant for a goal g if
    - a makes at least one of g's literals true
      - $g \cap \text{effects}(a) \neq \emptyset$
    - a does not make any of g's literals false
      - $g^+ \cap \text{effects}^-(a) = \emptyset$ and $g^- \cap \text{effects}^+(a) = \emptyset$



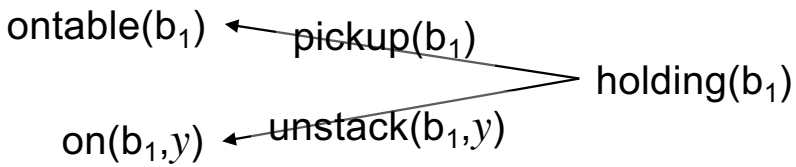
State Space Search (SSS): Regression (II)

- Backward search can *also* have a very large branching factor
  - E.g., an operator o that is relevant for g may have many ground instances a_1, a_2, …, a_n such that each a_i's input state might be unreachable from the initial state
- As before, deterministic implementations can waste lots of time trying all of them
- STRIPS uses this search

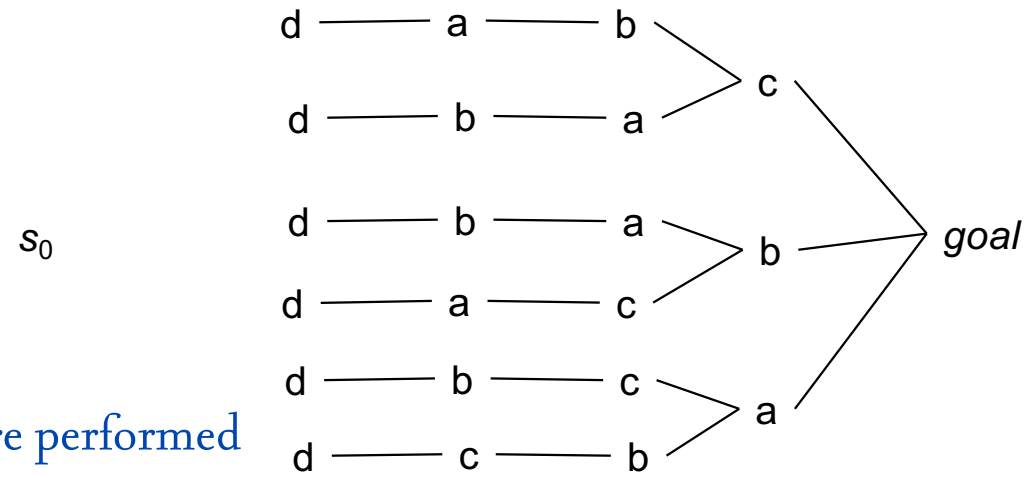
State Space Search (SSS): Lifting



- Can reduce the branching factor of backward search if we *partially* instantiate the operators = is called *lifting*



State Space Search (SSS): Lifting



- More complicated than Backward-search
  - Have to keep track of what substitutions were performed
- But it has a much smaller branching factor, but still can be quite large
- Suppose actions a, b, and c are independent, action d must precede all of them, and there's no path from s_o to d's input state
- We'll try all possible orderings of a, b, and c before realizing there is no solution

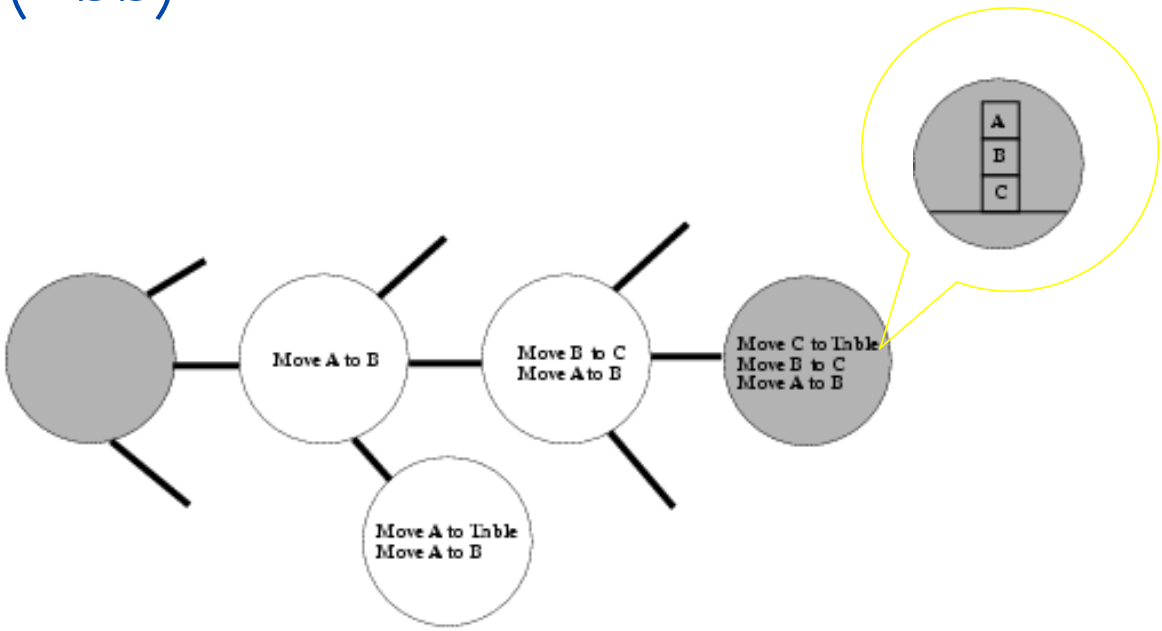
Classification of algorithms: search

- State-space search in planning
  - Each node represents a state of the world
  - A plan is a path through the space
- **Plan-space search in planning**
  - Each node is a set of partially-instantiated operators, plus some constraints
  - Impose more and more constraints, until we get a plan

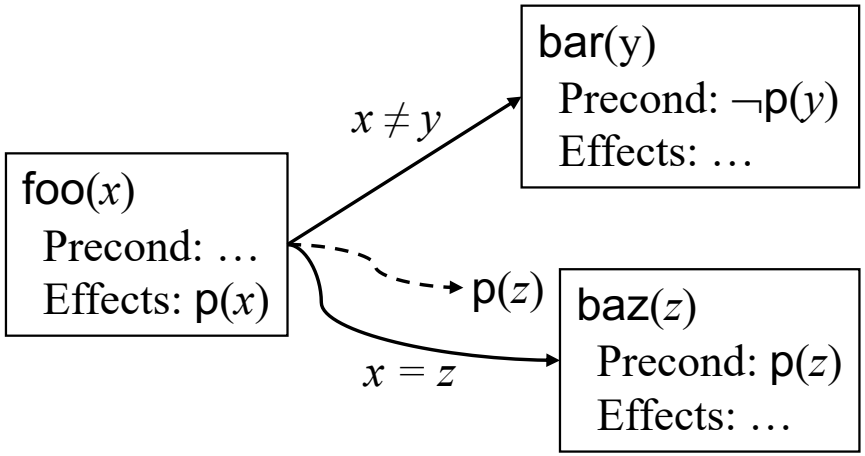
Plan Space Search (PSS)

- In 1974, the planner NOAH (Sacerdoti) was built, the search was conducted through the space of plans (PSS) :
  - Each node in the tree/graph represents a partial plan
  - Each arc represents refining plan operations (add an action to the plan)
  - A set of constraints
- The initial node is a NULL plan and the final node represents the solution plan for the goal
- While SSS has to return the solution path from the initial to the goal states, the goal state in PSS is the solution

Plan Space Search (PSS)



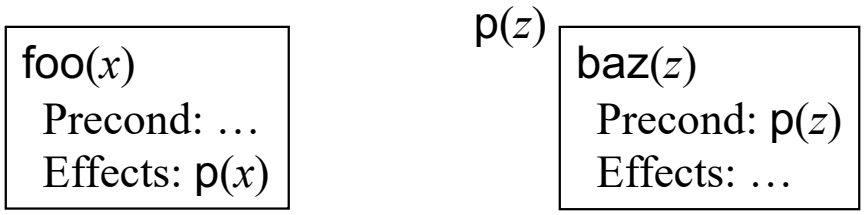
Plan Space Search (PSS)

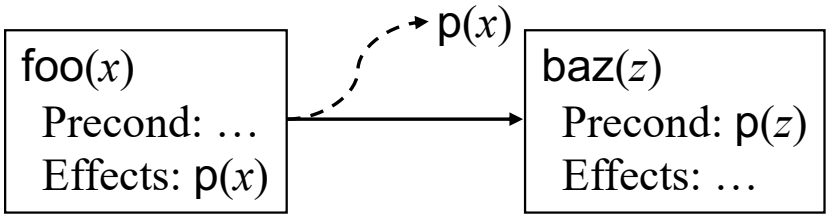


- Backward search from the goal
- Types of constraints:
  - *precedence constraint*: *a* must precede *b*
  - *binding constraints*:
    - inequality constraints, e.g., $v_1 \neq v_2$ or $v \neq c$
    - equality constraints (e.g., $v_1 = v_2$ or $v = c$) and/or substitutions
  - *causal link*: use action *a* to establish the precondition *p* needed by action *b*
- How to tell we have a solution: no more *flaws* in the plan

Plan Space Search (PSS): Flaws

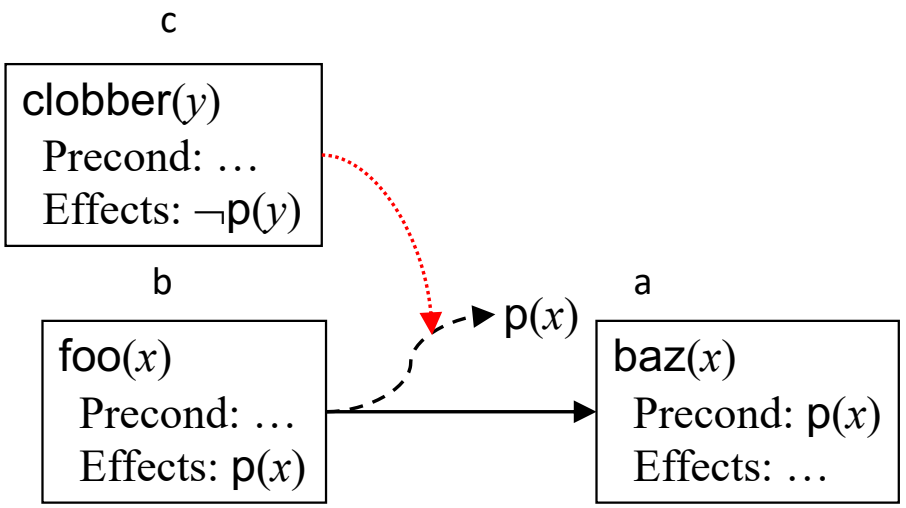
- Open goal:
  - An action a has a precondition p that we haven't decided how to establish
- Resolving the flaw:
  - Find an action b
    - (either already in the plan, or insert it)
  - that can be used to establish p
    - can precede a and produce p
  - Instantiate variables and/or constrain variable bindings
  - Create a causal link





Plan Space Search (PSS): Threat

- Threat: a deleted-condition interaction
  - Action *a* establishes a precondition (e.g., p(*x*)) of action *b*
  - Another action *c* is capable of deleting *p*
- Resolving the flaw:
  - impose a constraint to prevent *c* from deleting *p*
- Three possibilities:
  - Make *b* precede *c*
  - Make *c* precede *a*
  - Constrain variable(s) to prevent *c* from deleting *p*



Outline

- Introduction
- Classification of algorithms
- **Planning techniques**
  - Total Order Planners (TO)
  - Partial Order Planners (POP)
  - Hierarchical Task Network (HTN)
  - Graph-based Planners (GP)
  - SAT-based planners
  - Heuristic Search Planners
- Conclusions

Planning techniques

- TO: the solution is a totally ordered sequence of actions (States / Plans)
- PO: search at the space plans. Implements a "least Commitment approach": only the essential decisions orders are saved
- HTN: network of tasks and constraints
- Graph-based: the search structure is a Planning Graph
- SAT: takes as input a problem, guess the length of the plan and generates propositional clauses
- Heuristic Search Planners: transform planning problems in heuristic problems

Planning techniques

- **Total Order Planners (TO)**
- Partial Order Planners (POP)
- Hierarchical Task Network (HTN)
- Graph-based Planners (GP)
- SAT-based planners
- Heuristic Search Planners

Planning techniques

- Total Order Planners (TO)
- **Partial Order Planners (POP)**
- Hierarchical Task Network (HTN)
- Graph-based Planners (GP)
- SAT-based planners
- Heuristic Search Planners

Planning techniques

- Total Order Planners (TO)
- Partial Order Planners (POP)
- **Hierarchical Task Network (HTN)**
- Graph-based Planners (GP)
- SAT-based planners
- Heuristic Search Planners

Planning techniques

- Total Order Planners (TO)
- Partial Order Planners (POP)
- Hierarchical Task Network (HTN)
- **Graph-based Planners (GP)**
- SAT-based planners
- Heuristic Search Planners

Planning techniques

- Total Order Planners (TO)
- Partial Order Planners (POP)
- Hierarchical Temporal Planners (HTN)
- Graph-based Planners (GP)
- **SAT-based planners**
- Heuristic Search Planners

Planning techniques

- Total Order Planners (TO)
- Partial Order Planners (POP)
- Hierarchical Temporal Planners (HTN)
- Graph-based Planners (GP)
- SAT-based planners
- **Heuristic Search Planners**

Outline

- Introduction
- Classification of algorithms
- Planning techniques
- **Conclusions**

Conclusions

- Planning is an interesting area because it combines logic and search
- We have studied the main planners and the search algorithms that use
- There are actually no better strategies than others
- Competition between approaches and the intersection and combination of techniques has resulted in gains in efficiencies in syst. planning