Intelligent Systems: Reasoning and Recognition

James L. Crowley

ENSIMAG 2 Second Semester 2020/2021
Lesson 19 14 April 2021

Planning and Problem Solving

Background:

Simon, H. A. (1981). *The sciences of the artificial*. Cambridge, Massachusetts: MIT Press.

Nilsson, N. J., (1998). *Artificial intelligence: a new synthesis*. Morgan Kaufmann. (This is an updated version of Nilsson's classic 1980 textbook).

Korf R. E. (1987), Planning as search: A quantitative approach, *Artificial Intelligence*, Vol 33, Issue 1, pp65-88.

The Intelligent Agent

General Problem Solver and Means Ends Analysis.
In the early years of AI (1960s and 1970s), researchers posed the problem of intelligence as the ability to solve problems. In the late 1950s. Newell and Simon attempted to formalize problem solving with an algorithm known as the "General Problem Solver" (GPS) using "Means-Ends Analysis" (MEA).

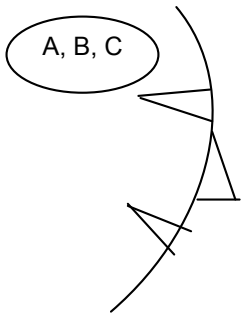
The General Problem Solver was a goal-based problem solving technique in which the solution to a problem can be described as finding a sequence of actions that lead to a desirable goal. Means-Ends Analysis (MEA) is a strategy to control search in problem-solving formulating goals (ends) as states, and possible actions as state transitions (means).

With Means-Ends Analysis, given a current state and a goal state, a set of possible actions are listed for the current state and an action is chosen which will reduce the difference (or distance) between the current state and the goal state. The selected action is performed on the current state to produce a new state, and the process is recursively applied to this new state and the goal state. GPS and MEA are still taught as part of curriculum in business management in some business schools.

This established planning and problem solving as an important area of artificial intelligent in the 1970s and 1980s, and lead to the notion of defining Intelligence as rationality.

The Rationality Principle

In his 1980 Turing Award, A. Newell proposed the Intelligent Agent as a fundamental concept for formalizing intelligence.



The Intelligent Agent is an abstract concept composed of 3 components: (A, B, C)
A) Actions: The ability to act; The ability to change the state of the universe.
B) Goals. Desired states.
C) Knowledge; The ability to choose actions to accomplish goals.

Newell claimed that to be considered as intelligent, the agent must have three abilities:
1) The agent must be able to act; able to change the state of the universe (embodied)
2) The agent must have goals, (desired states).
3) The agent must be able to choose its actions to accomplish goals (choose actions to bring and hold the current state to a goal state).

This was formulated as the principal of Rationality. The principle of rationality was widely used in Economics and only abandoned in favor or Behavioral Economics over the last 20 years.

The "Principal of Rationality states that an intelligent agent chooses its actions to accomplish its goals.

Knowledge is what enables the intelligent agent to choose actions.

Knowledge = Competence: The ability to act to bring the world to a desired state.

This paradigm leads to development of a series of frameworks and algorithms for planning and problems solving that remain widely used in robotics and other areas. The fundamental concept is the notion formalizing planning as search through a state space.

Planning as Search

Problem Spaces

Planning is formalized using a state space referred to as a "Problem Space".
A problem space is defined as

1) A set of states {U} (the Universe),
2) A set of operators for changing states {A} (Operations or Actions).

A state is defined using a conjunction of predicates (possibly negated - a negated predicate is a predicate). Disjunctions give rise to separate states, allowing for resolution by theorem proving, using, for example, Horn Clauses.

A problem is defined as a problem space ({U}, {A}) plus

an initial state i $\in$ {U}, and
a set of Goal States {G} $\subset$ {U}

A plan creates a sequence of actions A_1, A_2, A_3, A_4,... $\in$ {A} that lead from the initial state *i* to one of the states g $\in$ {G}

With this approach, planning is formalized as the search for a sequence of actions leading from an initial state to a goal state.

The core concept here is the notion of "State". A state is a "partial" description of the environment represented as a conjunction of predicates.

State: A conjunction of predicates (truth-valued functions) over entities.

A state is defined as a conjunction of predicates. Predicates are truth functions which represent relations between entities (perceived phenomena). Relations associate entities, typically based on properties.

Predicates express relations (spatial, temporal, part-of, category inclusion, etc) that associate entities. The basic principles are illustrated by Blocks world, formalized by Nils Nilsson in his 1980 text-book "Artificial Intelligence".

Blocks World

Blocks world is an abstract, toy world for exploring problems of reasoning and intelligence. We will use Blocks world to illustrate different principals and techniques concerning knowledge representation.

Blocks world is composed of:

- A table
- A set of blocks
- An agent (robot hand) that can act on (move) the blocks

The blocks, tables and hand are the primitive concepts that make up a blocks world. They are primitive because they are directly perceivable.

Classic Definition:
1) A universe composed of a set of cubic blocks and a table
2) Blocks are mobile, the table is immobile
3) The agent is a mobile hand,
4) A block can sit on a table, on another block, or in the hand.
5) There cannot be more than one block on another block
6) The table is large enough for all blocks to be on the table.
7) The hand can move only one block at a time.

The state of the universe is formalized using predicates.
Blocks are represented by Capital Letters {A, B, C, …}
Variables (lower case letters) represent sets of blocks
This are specified by Quantifiers: for-all x (∀x), There-exists y: (∃y)

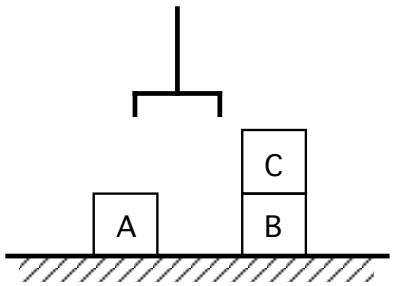
Predicates

Typical predicates used to define states in blocks world are

On(y, x)	O(y,x)	Block x is on Block y
OnTable(x)	OT(x)	Block x is on the table.
Held(x)	H(x)	Block x is in the hand.
Free(x)	F(x)	No block is On x : ¬∃y: (On(x, y)) or ∀y: (¬On(x, y))
HandFree()	HF()	The hand is empty, or ¬∃x (H(x))

Free() and HandFree() are complex concepts that build on the primitive concepts.

For example: HF ∧ OT(A) ∧ OT(B) ∧ On(B,C) ∧ F(C) ∧ F(A)



This is an example of a situation and can be expressed with Situation Modeling.

Actions

The system can move from one state to another by performing actions.
Actions, {A}, are represented as rules: IF <condition> THEN DO <action>.

Both <condition> and <action> are partial state descriptions using the same predicates that we used to define the problem space.
Rules can be used in forward chaining manner, matching <condition> to the predicates in the current state, or backward chaining, by matching the predicates made true by the action to the goal state.

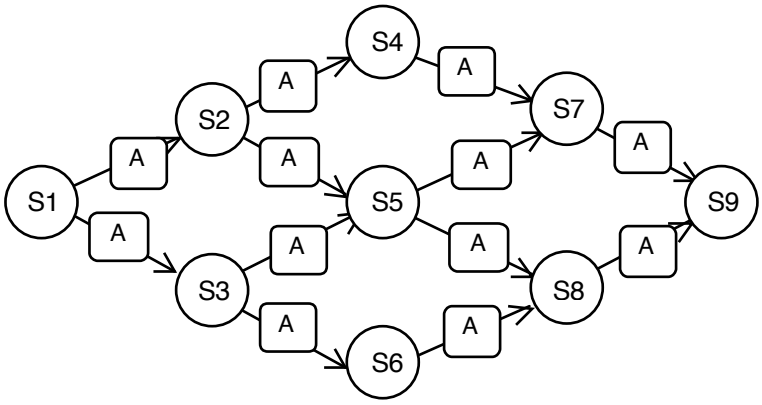
Actions are be defined by Arguments, Preconditions, and Post-conditions.

Action (<blocks>)

Preconditions: Predicates that must be true to execute the action

Post-conditions: Predicates that are made true or false by the action

To solve a problem we search for a path through the state space from an initial state to a target state.



We can use blocks world to illustrate various concepts and techniques. It is also useful for homework problems and exam questions!

Nilsson defined four actions for blocks world.

Grasp(x):
Precondition: HF() ∧ F(x) ∧ OT(x)
Postcondition: ¬HF() ∧ ¬OT(x) ∧ H(x)

Pose(x):
Precondition: H(x)
Postcondition: ¬H(x) ∧ HF() ∧ OT(x)

Stack(x, y): Stack block x on block y
Precondition: H(x) ∧ F(y)
Postcondition: ¬H(x) ∧ ¬F(y) ∧ F(x) ∧ O(y, x) ∧ HF()

Unstack (x, y)
Precondition: F(x) ∧ O(y, x) ∧ HF()
Postcondition: ¬O(y, x) ∧ ¬HF() ∧ H(x) ∧ F(y)

With some clever tricks, Nilsson was able to define a version of actions for Blocks World where all precondition predicates are rendered false by execution of an action. However, this is not true of most formulations, or in most problem domains.

Comments on Blocks World and Search

Note that Nilsson's version of blocks world is a "Closed" world. It has a finite number of states. Real problems tend to be open, with a very large branching factor (possible actions) and an infinite number of states.

Nilsson's definition led to some very elegant solutions in rule based programming systems, such as Prolog, but was not generalizable. It was considered to be a "toy problem". Even this simple "toy" problem revealed some fundamental problems with formulating planning as search and looking for a least cost plan with a minimum effort for search.

Algorithms for Planning as Search

The general paradigm for planning, as shown with MEA, is "Generate and Test".
Planning is the generation of a sequence of actions to transform i to a state g ∈ {G}
Planning requires search for a path through a graph of states.

Nilsson defined a taxonomy of graph search algorithms includes the following
1) Breadth first search
2) Depth first search
3) Heuristic Search
4) Hierarchical Search
Nilsson unified Depth-First, Breadth First and Heuristic Search a single algorithm named GRAPHSEARCH.

The GRAPHSEARCH algorithm requires maintaining a list of "previously visited" states {C} (Closed list) and a list of available states to explore {O} (Open list).
As each state is tested, its neighbors are generated and added to the open list.
The state is then added to the closed list.

GRAPHSEARCH:
{O}←intial state
WHILE {O} ≠ empty DO
    Extract a new state s from {O}
    IF s ∈ {G} THEN halt ELSE add s to {C}
    Generate all neighbor states {N} of s.
    ∀ n ∈ {N} IF n ∉ {C} THEN add n to open states {O}.

Breadth-first, depth-first and heuristic search are all variations on the same GRAPHSEARCH algorithm, depending on whether the Open list is a stack, queue or sorted.

1) Breadth first search - The Open list {O} is a Queue (First In, First Out)
2) Depth first search - The Open list {O} is a Stack (First In, Last Out)
3) Heuristic Search - The open list {O} is sorted by a Cost f(s) = g(s) + h(s)

Algorithmic Complexity of Search

Algorithm complexity is expressed with the order operator O().
Order of complexity is equivalent for all linear functions. O(AN+B) = O(N)

The algorithm complexity of graph search depends on two factors: b and d.

b: The branching factor; The average number of actions {A} possible in a state.

$b = E\{card(\{A\})\}$ (E{} is expectation, or average)

d: Depth. The minimum number of actions from an initial state to a goal state.

Breadth First search: the open list, {O} is a queue (FIFO)

For breadth first search, finding the optimal path requires exhaustive search.

Computation Cost $O(b^d)$, memory $O(b^d)$.

This is a problem for humans, because human cognition has an important physiological limit: The size of working memory. To use breadth first planning a human must use a memory aid, such as a note pad. Human's use something more clever: Chunking, leading to hierarchical search discussed below.

Depth First search: The open list {O} is a stack (LIFO).

For depth first search, finding the optimal path requires exhaustive search, however

Computation Cost $O(b^d)$, memory $O(d)$.

This can be done by humans for limited depth problems. ($d \leq 7$)

However, limiting depth first requires setting a maximum depth d_{max}. which can cause the search to fail.

Most importantly, a cost of $O(b^d)$ is not acceptable for most real world problems.

Heuristic search: {O} is sorted based on the cost f(s) of a path through the state.

For Heuristic search, we reduce the order by reducing the branching factor:

This give computation and memory costs of $O(c^d)$ where $c \leq b$.

Heuristic Search is NOT exhaustive. The search avoids unnecessary branches.

For heuristic search, the cost of path through a state is $f(s) = g(s) + h(s)$ where where g(s) is the cost of a path from the start state to the state s (easily calculated) and h(s) is the cost of a path from the state to the goal state (unknown)

$$f(s) = g(s) + h(s)$$

Because h(s) is not known, it must be approximated with an estimate.

Optimality requires that the estimate for h(s) meet the "optimality conditions".

Nilsson's Conditions for Optimal Search

In 1968, Nilsson demonstrated the conditions under which a heuristic search algorithm is guaranteed to be optimal. That is: the first plan found is the cheapest plan to execute.

Notation :

i : initial state

g : goal state $g \in \{G\}$

$k(s_i, s_j)$: the minimal theoretical cost between states s_i and s_j

$g^*(s) = k(i, s)$: The true cost of the shortest path from i to s.

$h^*(s) = k(s, g)$: The true cost of the shortest path from s to $g \in \{G\}$.

$f^*(s) = g^*(s) + h^*(s)$ The true cost of the shortest path from i to g passing by s. This is the cost of the path that we would like to discover.

Problem: If we do not know the shortest path, how can we know $h^*(s)$?
Solution: Approximate costs with an estimation function.

Define:

g(s) : estimated cost from i et s.

h(s) : estimated cost from s to g

$f(s) = g(s) + h(s)$ estimated total cost of a path through s

Nilsson showed that whenever the estimated cost $f(s) \leq f^*(s)$,
the first path that is found from s_1 to s_2 will always be the shortest.
Thus $g(s)=g^*(s)$ because the first path from i to s has the least cost.

$h(s) \leq h^*(s)$ requires two conditions:

Condition 1: that the heuristic UNDER-ESTIMATES the true cost.

$h(s) \leq h^*(s)$

Condition 2: that estimate h(s) is "monotonic". That is :

$h(s_i) - h(s_j) \leq k(s_i, s_j)$

Condition 2 is almost always true whenever $h(s) \leq h^*(s)$!!

Nilsson called this the A* condition.
A* is "optimal" because the first path found is the path with least-cost.

Cost and Optimality of Heuristic Search

A key problem is defining heuristic search is the concept of numerical "cost" for executing an action.

Cost, $k(s_i, s_j)$, can be any numerical value, but must respect the optimality conditions for GRAPHSEARCH to be optimal.

A* is widely used for robot path planning, where cost is a formulated as a linear function of Euclidean distance. Examples of cost: distance, time, Euros, risk, number of actions.

Whenever the cost metric is proportional to the length of the path, then Euclidean distance to the goal provides an "optimal" heuristic!

This is true for scalar multiples of distance, for example, time traveled or risk or cost or energy expended. (assuming constant speed, time = distance x 1/speed)

Cost of Search vs Optimality of Result

Note that there are TWO notions of cost!

1) Algorithmic complexity of the search (computational cost).
2) Cost of executing the resulting plan.

“Optimal” search means that the resulting plan is the cheapest possible plan.

Nilsson's A* algorithm provided both. Nilsson demonstrated that when the conditions for optimality of A* are met, the first plan discovered with A* (lowest computational cost) would be the least expensive plan (least cost execution.)

Note that for $h(s) = 0$, $h(s)$ meets the optimality condition because $h(s) \leq h^*(s)$!!

In this case A* reduces to Dijkstra's algorithm (1956), used for network routing.

For Blocks world, the usual cost is the number of actions. However, there is no admissible heuristic. The optimal solution is found by simply setting h(s) to 0. Even in this case, the algorithm reverts to a form of Dijkstra's algorithm, and the first solution found is always the best solution.

Lesson: Having an admissible heuristic for A* search reduces the cost of search over Dijkstra, but does not change the cost of the final, least-cost path.

Hierarchical Planning, Subgoals and Chunking

For many real problem domains, the cost of search is MORE EXPENSIVE than the cost of executing the resulting plan. H. Simon won the Nobel Prize in economics for demonstrating that in solving problems, humans sacrifice optimality to reduce the effort required for planning. He called this "satisficing".

A key technique for satisficing is hierarchical planning using known subgoals. Hierarchical planning sacrifices optimality in order to greatly extend the space of possible problems that can be solved by planning.

Subgoals

Subgoals can strongly reduce the algorithmic complexity search for a path through a network. A "Subgoal" is an abstract state that represents a set of states.

For example, to plan a route from campus to the train station.
1) Plan a route from Campus to the Quai d'Isere
2) Plan a route from the Quai d'Isere to the train station.

Define:

d : the minimal number for actions from the start state, i, to a goal state, g.
d_{is}: the minimal number for actions from the start state, i, to a subgoal s.
d_{sg}: the minimal number for actions from the subgoal s, to a goal state.

The subgoal divides the problem into two smaller subproblems: Plan a path from it to s, and plan a path from s to g.

$$O(b^{d_{is}} + b^{d_{sg}}) = O(b^{\max\{d_{is}, d_{sb}\}})$$

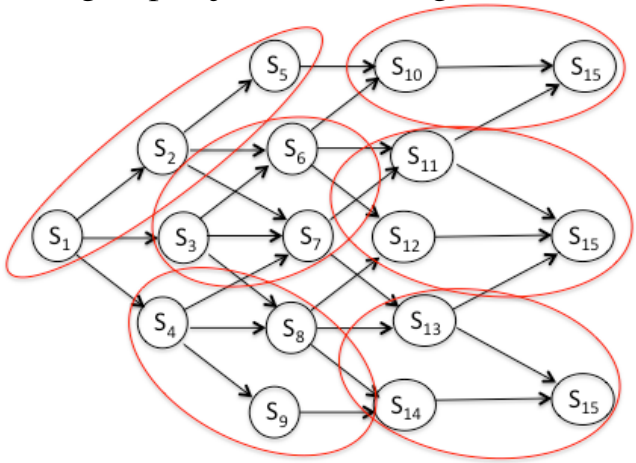
if $\max\{d_{is}, d_{sb}\} < d$, then the complexity is reduced.

Hierarchical search sacrifices autonomy for scope.
Hierarchical search provides solutions to more complex problems, but there is no guarantee that the resulting solution is "optimal".

Two possible approaches to hierarchical planning are
1) Build a hierarchy of super-states
2) Define a hierarchy of operators. (meta-operators)

Hierarchy of states

An obvious approach is to group adjacent states together to form Super states.



Korf set this up as an optimisation problem and found that an "optimal" size for super-states was "e" (2.71828...)

Possible ways to group states

1) Group sets of states connected by a single action to a privileges state
2) Define arbitrary connected sets of states
3) Group states by eliminative predicates.

A state is a conjunction of predicates $P_1() \wedge P_2() \wedge P_3() \wedge P_4()$

The state $P_1() \wedge P_2()$ represents the states $P_1() \wedge P_2() \wedge P_3()$ AND $P_1() \wedge P_2() \wedge \neg P_3()$

We can group states by deleting predicates and divide states into substates by adding predicates.

Sussman tried this with Blocks world and found that it was dependent on which states were eliminated. For example replacing

On(B,C) ∧ On(A,B) ∧ OT(A) with On(B,C) is not helpful.

Operators

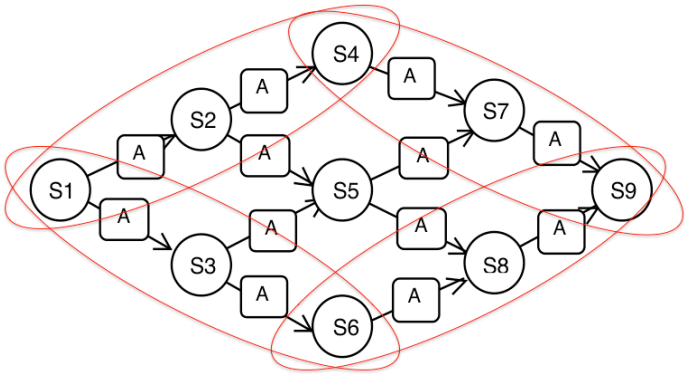
A more effective means is to group sequences of states and actions into "operators"

$S_1 - A_{12} \rightarrow S_2 - A_{23} \rightarrow S_3 - A_{34} \rightarrow S_4$

Is an operator to go from S_1 to S_4

The States translate to perceptual actions P() to verify the results of each action and to verify the pre-condition for the next action.

Operator $(S_{1,} S_4) = P(S_1) - A_{12} \rightarrow P(S_2) - A_{24} \rightarrow P(S_4)$



Human's do this to reduce the load on short term working memory. Simon called this "Chunking".

Operators are composed hierarchically, as needed, to accommodate the limits of human working memory. This is clearly demonstrated by chess experts solving chess problems and can be explained using Johnson Laird's situation modeling.

Chunking

To speed up search and to overcome limits to short term memory, humans use a technique called "chunking".

Chunking is a process by which individual pieces of information are bound together into a meaningful whole. A chunk is a concept that represents a collection of concepts.

Chunking states into sets of states enables hierarchical planning.
Chunking can be applied hierarchically, with groups of states at each level represented by a single at the next level.

Example: Travel Planning

Consider the problem of planning a trip to Oxford.
First we choose whether to go by plane, train or bus.
Choosing plane, we select company and flights : Air France LYS-CDG-LHR
Then we plan the trip to Lyon Airport: Bus, Train or car
Then we plan the trip from Heathrow: Bus, Train or car
Then we plan the trip to the train station to catch the bus to LYS, etc.