Agent and Multi-Agent Systems:

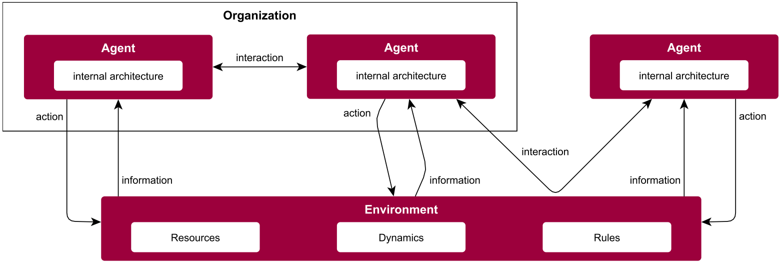
Automated Planning

Wassila Ouerdane - Emmanuel Hermellin

November 3 and 4, 2021

Recall

- In Multi-Agent Systems, the control is distributed.
  ⇝ Each agent takes decision autonomously.
- Agents are situated in the same dynamic environment.
  ⇝ Possible interactions between agents' actions, and not always possible to anticipate this during the design of the system.
- Agents may share limited resources.



Coordination

Coordination is used in the case where individual agents actions can **interfere** with each other:

- Positive interactions: Share of actions, joint realization of actions, . . .
- Negative interactions: Incompatibility of goals, access to limited resources, . . .

→ Taking into account behavioral interactions allows to improve the behavior of agents: Resolution of issues, emergence of behaviors, etc.

Coordination

Definition [Malon,1988]

"The set of supplementary activities which need to be carried out in a multi-agent environment, and which a single agent pursuing the same goals would not accomplish."

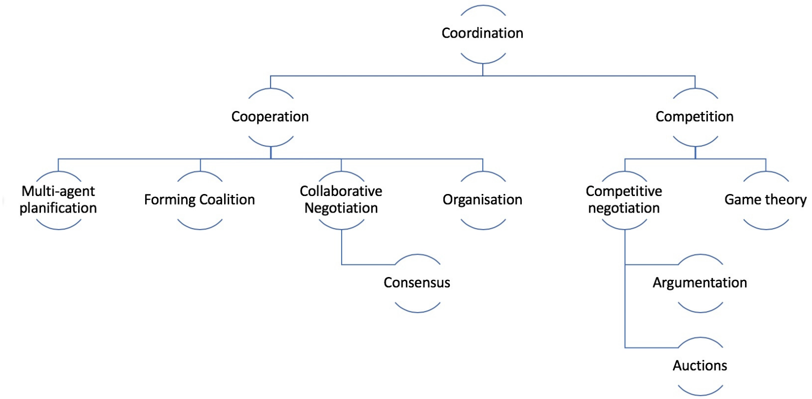
Definition [Ferber, 1995]

"The coordination of action can be defined as the articulation of the individual actions accomplished by each agents in such a way that the whole ends up being a coherent and high performance operation. It is a matter of arranging the behaviors of the agents in time and space in such a way that the group action is improved, either through better performance or through a reduction in conflict. So the coordination of actions is one of the main methods of ensuring cooperation between autonomous agents." [Ferber, 1995]

Why coordination?

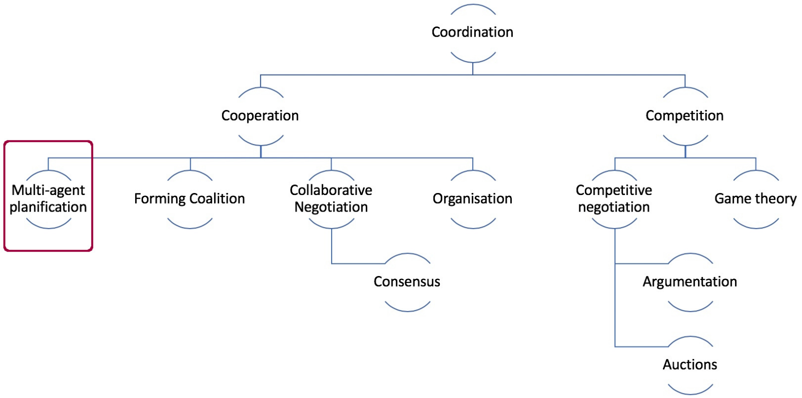
- The agents need information and results which only other agents can supply. (*e.g.* An agent supervising the activity of an industrial process at a given point will need information on the state of this process at other points).
- The resources in the environment may be limited. Resources have to be shared in such a way as to optimise the actions to be carried out (*e.g.* eliminating pointless actions, improving response times, reducing cost) while still trying to avoid any possible conflicts (*e.g.* contradictory actions).
- Optimize costs of actions (*e.g.* eliminating pointless actions and avoiding redundant actions).

Coordination mechanisms



Source: Borrowed from Aurélie Beynier's course.

Coordination mechanisms



Source: Borrowed from Aurélie Beynier's course.

Table of contents

First Intuition on Planning

Intuitive Planning Definition

What is Planning?

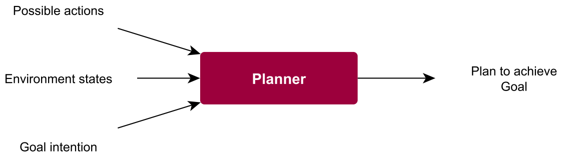
- Planning is the reasoning side of acting. It is an abstract, explicit deliberation process that chooses and organizes actions by anticipating their outcomes.
- This deliberation aims at achieving as best as possible some objectives.
- Automated planning is an area of Artificial Intelligence (AI) that studies this deliberation process computationally.

Book: *M. Ghallab, D. Nau and P. Traverso Automated Planning Theory and Practice Morgan Kaufmann, 2004*

Practical agent (recall from Basic Course)

Means-ends reasoning/planning

- Planning is the design of a course of actions that will achieve some desired goal.
- Planning system required:
    - (representation of) goal/intention to achieve,
    - (representation of) actions it can perform,
    - (representation of) the environment.
- Planning system generates a plan to achieve the goal.



Automated Planning Motivations

- Practical Motivations: Designing information processing tools that give access to affordable and efficient planning resources.
- Theoretical Motivations: Planning is an important component of rational behaviour.

Automated Planning Motivations

- Practical Motivations: Designing information processing tools that give access to affordable and efficient planning resources.
- Theoretical Motivations: Planning is an important component of rational behaviour.

Rescue operation after a natural disaster!

- It involves a large number of actors and requires transportation infrastructures.
- It relies on careful planning and assessment of several alternate plans.
- It is time constrained and demands immediate decisions that must be supported with a planning tool.

Planning

- Necessary when near-term choices of actions can enable, or prevent, later action choices required to achieve goals.
- Possible when agent possesses a sufficiently detailed and correct model of the environment, and how action affect the environment.
- Challenging because the space of possible plans grows exponentially with the plan duration.

Conceptual Model of Planning

State Transition System:
The conceptual model of planning can be represented as a state transition system. Formally, it is a 4-tuple $\Sigma = (S, A, E, \gamma)$:

- $S = \{s_0, s_1, \ldots, s_n\}$ is a finite set of states (*the possible states the world can be in*),
- $A = \{a_0, a_1, \ldots, a_n\}$ is a finite set of actions (*the actions that can be performed by some agent in the world*),
- $E = \{e_0, e_1, \ldots, e_n\}$ is a finite set of events (*the events that can occur in the world*),
- $\gamma : S \times A \times E \rightarrow 2^S$ is a state transition function (*the function describing how the world evolves when actions and events occur*).

A state transition system may be represented by a directed graph whose nodes are the state is S.

Note: 2^S *means powerset of* S, *maps to a set of states.*

Planning Objectives

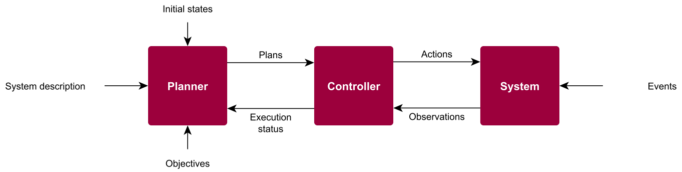
Given a state transition Σ, the purpose planning is to find which actions to apply to which states in order to achieve some objective when starting from a given situation. A plan is a structure that gives the appropriate actions. The objective can be expressed :

1. By simply specifying a goals state s_g or a set of goal states S_g. In this case, the objective is achieved by any sequence of state transition that ends at one of the goal states.
2. By the satisfaction of some conditions over the sequence of state followed by the system.
3. By a utility function attached to each state, with penalties and rewards. The goal is to optimize some compounded function of these utilities.
4. As tasks that the system should perform.

Graphical Representation of Planning Model

It is convenient to depict conceptual planning model through the interaction between three components:

- A state transition system Σ evolves as specified by its state transition function γ according to the events and actions that it receives.
- A controller, given as input the state s of the system, provides as output an action a according to some plan.
- A planner, given as input a description of the system Σ, an initial situation, and some objective, synthesizes a plan for the controller in order to achieve the objectives.



Source: Borrowed from Damien Pellier's course.

Multi-Agent Planning

Multi-Agent Planning

- Multi-Agent Planning (MAP) introduces a new perspective in the resolution of a planning task with the adoption of a distributed problem-solving scheme instead of the classical single-agent planning paradigm.
- Distributed planning is required *"when planning knowledge or responsibility is distributed among agents or when the execution capabilities that must be employed to successfully achieve objectives are inherently distributed"* [desJardins et al. 1999].

Article: *Weerdt, Mathijs & Clement, Bradley. (2009). Introduction to planning in multiagent systems. Multiagent and Grid Systems. 5. 345-355. 10.3233/MGS-2009-0133.*

Multi-Agent Planning

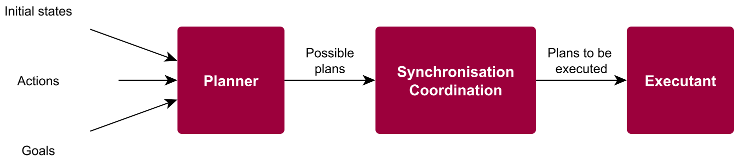
What is a plan?
A (partially) ordered sequence of actions such that the execution makes it possible to achieve the desired goals or to maximize a performance measure.

Challenges

- The actions of an agent have a more or less long-term consequences on the other agents (actions, objectives).
- It is not effective for agents to plan independently of one another.
- Challenging because the space of possible plans grows exponentially with the plan duration, and of multi-agent plans grows exponentially in the number of agents.
- Coordination required between agents (whether they are cooperatives or competitive).

Multi-Agent Planning Modes

Planning actions in MAS universes can be broken down into three distinct stages: making plans, synchronizing/coordinating plans, and executing plans.

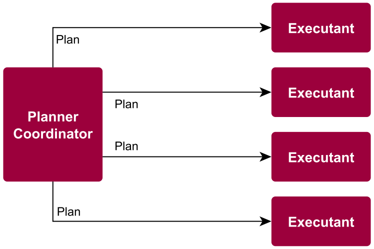


Different modes:

- Centralized Planning for multiple agents.
- Centralized Coordination for partial Plans.
- Distributed planning.

Centralized planning for multiple agents

- Assumes that only a single planner exists (a single agent capable of planning and organizing actions for all agents).
- Handles the synchronizing of plans together with the allocation of tasks to the other agents (executants).
- Assumes that the planner has an exact knowledge of the environment and is able to anticipate its evolution.
- The agents (executants) should send their private knowledge to the planner (preferences for instance).



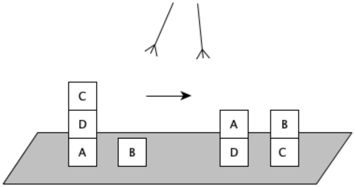
Centralized planning for multiple agents

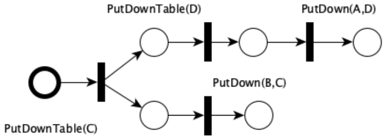
Three stages:

1. The first step is to look for a general plan, which can be expressed in the form of a cyclic diagram.
2. We then determine the branches which can be executed in parallel, and we introduce synchronisation points each time two computation branches join together.
3. Finally, we allocate the execution of the tasks to the various agents concerned.

Centralized planning for multiple agents

Example: Two robots manipulating the cubes

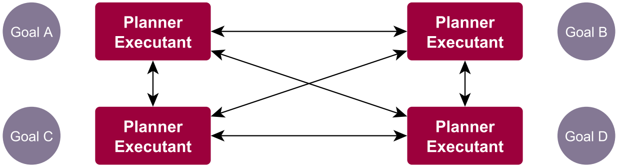




Distributed planning

- No centralizing agent either for drawing up overall plans or coordinating partial plans.
- Each agent individually plans the actions it intends to carry out, depending of its own goal.

✗ Difficulties: Resolution of potential conflicts, recognition of the synergetic situations



The problem of the agents consists of exchanging information relating to their plans and their goals, so that each of them may achieve its objectives.

Distributed planning

- Cooperative Distributed Planning (CDP): Focuses on **planning** and how it can be extended into a distributed environment, where the process of formulating or executing a plan could involve a number of participants.
- Negotiated distributed planning (NDP): Focuses on **coordinating and controlling the actions** of multiple agents in a shared environment, and has adopted planning representations and algorithms as a mean to do that end.

Cooperative Distributed Planning: Common questions.

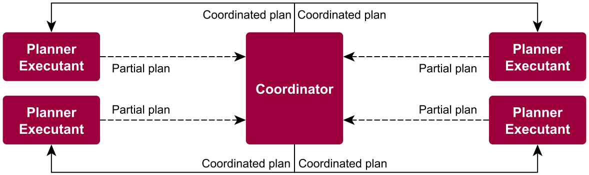
- How the plans or partial plans are represented?
- What are the methods for generating the plans?
- How the tasks are allocated to agents?
- How the agents communicate with each other during the planning?
- How the agents coordinate their actions?

$\rightsquigarrow$ **Centralized coordination** of distributed plans.

$\rightsquigarrow$ **Distributed coordination** of distributed plans.

CDP: Centralized coordination of distributed plans

- Each agent independently draws up it own partial plan, which is sends to the coordinator.
- The coordinator synthesize all the partial plans into a single coherent overall plan.
- The coordinator communicates to synchronize the agents actions.
- Objectives: Avoid conflicts and parallelize actions as much as possible.
  - Identification of unsafe (conflicting) situations.
  - Insertion of synchronization primitives to avoid such situations.



CDP: Centralized coordination of distributed plans

Limits

- Uncontrollable system in case of faulty coordinator.
- Bottleneck at coordinator level.
- Very high communication load: communications with the coordinator as soon as you reach a critical area.
- Loss of data confidentiality.
- Question of the neutrality of the central coordinator.

CDP: Distributed coordination of distributed plans

- Reduce the communication.
- Allows parallel resolution of coordination.
- Greater robustness.
- Possibility of keeping confidential data (privacy).

CDP: Distributed coordination of distributed plans

- Reduce the communication.
- Allows parallel resolution of coordination.
- Greater robustness.
- Possibility of keeping confidential data (privacy).

Partial Global Planning [Durfee, 1988]:
Framework for the coordination of distributed systems in a dynamic and non predictable environment:

- Agents continually re-plan.
- Changes in an agent's plan influence others agents who replan in turn.
- Planning is continuous and agents need to revise and continue their plans despite the fact that they can be temporarily non-coordinated.

CDP: Distributed coordination of distributed plans

Partial Global Planning [Durfee, 1988]:
Cooperating agents exchange information in order to reach common conclusion about the problem-solving process.

- Partial: The system does not (indeed cannot) generate a plan for the entire problem.
- Global: Agents form non-local plans by exchanging local plans and cooperating to achieve a non-local view of problem solving.

Three iterated stages:

1. Each agent decides what its own goals are and generates short-term plans in order to achieve them.
2. Agents exchange information to determine where plans and goals interact.
3. Agent alter local plan in order to better coordinate their own activities.

CDP: Distributed coordination of distributed plans

- Communication: To prevent incoherence $\rightsquigarrow$ Meta-level structure to guide the cooperation.
- It indicates which agents an agent should exchange information with, and under what conditions they should do.
- Actions and interactions of a group of agents are incorporated into a data structure: partial Global plan (PGP):
    - Objective: The larger goal that the system is working towards.
    - Activity maps: Representation of what agents are actually doing, and what results will be generated by their activities.
    - Solution construction graph: Representation of how agents should interact, and what information should be exchanged, and when, in order for the system to successfully generate a result.

Negotiation Distributed Planning

Coordination through Join Intentions: Using the notion of "human teamwork models".

- The agents have a team model: Goals and plans of the team.
- Teams members may communicate to attain mutual beliefs while building joint intentions.
- Ease monitoring of team performance and recognition of special situations (e.g. failure), very useful in dynamic environments.

NDP: Join Intention Theory

- A team θ jointly intends a team action if team members are jointly committed to completing that team actions.
- A joint persistent goal (JPG): A joint commitment to achieve p (completion of a team action) is denoted, $JPG(\theta, p, q)$. q is an irrelevance clause $\rightsquigarrow$ t enables a team to drop the JPG should they mutually believe q to be false.
- Three conditions:
  1. All team members mutually believe that p is currently false.
  2. All team members have p as their mutual goal (they want p to be eventually true).
  3. All teal members believe that until p is mutually known to be achieved, unachievable or irrelevant they mutually believe that they each hold p as a weak achievement goal: $WAG(\mu, p, \theta, q)$ (μ: a team member).
- The commitment to attain mutual belief in the termination of p is a key aspect of a JPG.

NDP: Coordination through Join Intentions

Example: Coordinated attack of a fleet of helicopters

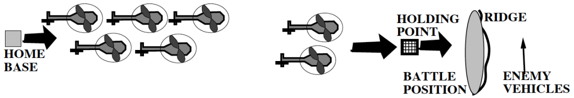


Figure 1: Attack domain: company flying in subteams

- Pilot agents for a company of (up to eight) synthetic attack helicopters.
- The company starts at the home-base, where the commander pilot agent first sends orders and instructions to the company members.
- The company processes these orders and then begins flying towards their specified battle position, i.e., the area from which the company will attack the enemy.
- While on the way to the battle position, depending on the orders, the company members may fly together or dynamically split into pre-determined subteams.

Going further !!

$\rightarrow$ **Planning under uncertainty** [Russel and Norvig, 2003]

- Markov Decision Processes (MDP)
- Partially Observable Markov Decision Processes (POMDP)
- Decentralized Partially Observable Markov Decision Processes (DEC-POMDP)
- ...

Automated Planning

Path and Motion Planning

Path and Motion Planning is concerned with the synthesis of a geometric path from a starting position in space to a goal and a control trajectory along that path that specifies the state variables in the configuration space of mobile systems, such as a truck, a mechanical arm, a robot,etc.

Motion planning takes into account:

- The model of the environment;
- The kinematic constraints;
- The dynamic constraints.

Perception Planning

Perception Planning is concerned with plans involving sensing actions forgathering informations. It arises in tasks such as modelling environments or objects, identifying objects, localizing through sensing a mobile system, or more generally identifying the current state of the environment.

- Perception planning addresses question such as information needed and when it is needed, where to look for it, which sensors are most adequate for a particular task and how to use them.
- It relies on decision theory for problems of which and when information is needed, on mathematical programming and constraint satisfaction for viewpoint selection and the sensor modalities.

Navigation Planning

Navigation Planning combines the two previous problems of motion and perception planning in order to reach a goal or to explore an area. The purpose of navigation planning is to synthesize a policy that combines localization primitives and sensor-based motion primitives.

Manipulation Planning

Manipulation Planning is concerned with handling objects, e.g. to build assemblies. It arises in dialog and in cooperation problems between several agents, human or artificial. It addresses issues such as when and how to query needed information and which feedback should be provided.

- The actions include sensory information.
- A plan might involve picking up an object from its marked sides, returning it if needed, inserting it into an assembly, and pushing lightly till it clips mechanically into position.

Domain Specific Approaches

Domain specific approaches to specific forms of planning are certainly well justified. However, they are frustrating for several reasons.

1. Some commonalities to all these forms of planning are not addressed in the domain specific approaches. The study of these commonalities is needed for understanding the process of planning.
2. It is more costly to address each planning problem anew instead of relying on and adapting some general tools.
3. Domain specific approaches are not satisfactory for studying and designing an autonomous intelligent machine. Its deliberative capabilities will be limited to areas for which it has a domain specific planner.

Domain Independent Approaches

Domain independent approaches relies on abstract, general models of actions.These models range from simple ones that allow only for limited forms of reasoning to models with richer prediction capabilities. There are in particular the following forms of models and planning capabilities.

1. Project Planning in which models of actions are reduced mainly to temporal and precedence constraints, e.g. the earliest and the latest start times of an action or its latency with respect to another action. Project planning is used for interactive plan edition and verification.
2. Scheduling and resources allocation in which the action models include the above types of constraints plus constraints on the resources to be used by each action.
3. Plan synthesis in which the action models enrich the precedent models with the conditions needed for applicability of an action and the effects of the action on the state of the world.

Conceptual Model of Planning

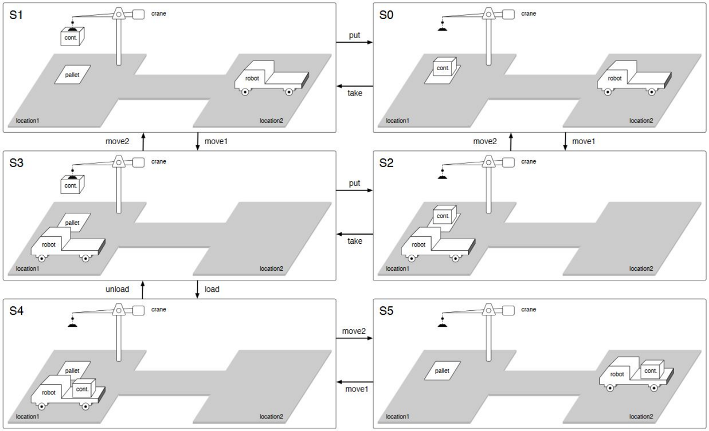
State Transition System:
The conceptual model of planning can be represented as a state transition system. Formally, it is a 4-tuple $\Sigma = (S, A, E, \gamma)$:

- $S = \{s_0, s_1, \ldots, s_n\}$ is a finite set of states
  (the possible states the world can be in),
- $A = \{a_0, a_1, \ldots, a_n\}$ is a finite set of actions
  (the actions that can be performed by some agent in the world),
- $E = \{e_0, e_1, \ldots, e_n\}$ is a finite set of events
  (the events that can occur in the world),
- $\gamma : S \times A \times E \rightarrow 2^S$ is a state transition function
  (the function describing how the world evolves when actions and events occur).

A state transition system may be represented by a directed graph whose nodes are the state is S.

Note: 2^S *means powerset of* S, *maps to a set of states.*

Crane and robot transportation example



Crane and robot transportation example shows a state transition system involving a container in a pile, a crane that pick up and put down the container and a robot that can carry the container and move it from one location to another.

Crane and robot transportation example

In this example:

- The set of states is $S = \{s_0, s_1, s_2, s_3, s_4, s_5\}$
- The set of actions is $A = \{take, put, load, unload, move1, move2\}$
- The set of events E is empty
- the transition function γ is defined by : if a is an action and $\gamma(s, a)$ is not empty then a is applicable to state s.

Restricted Model

Planning model puts forward various restrictive assumptions, particularly the following ones.

- **Finite Σ.** The system Σ has a finite set of states.
- **Fully Observable Σ.** The system Σ is fully observable, i.e. one has complete knowledge about the state of Σ.
- **Deterministic Σ.** The system Σ is deterministic, i.e. for every states s and for every event of action u, if an action is applicable to a state, its application brings a deterministic system to a single other state.
- **Static Σ.** The system Σ is static, i.e. the set of event E is empty. Σ has no internal dynamics.

Restricted Model

- **Restricted Goals.** The planner handles only restricted goals that a respecified as an explicit goal state s_g or a set of goal states S_g.
- **Sequential Plans.** A solution plan to a planning problem is a linearly ordered finite sequence of actions.
- **Implicit time.** Actions and events have no duration. They are instantaneous, state transitions. This assumptions is embedded in state transition systems, a model that does not represent time explicitly.
- **Offline Planning.** The planner is not concerned with any change that may occur in Σ while it is planning. It plans for a given initial and goal states regardless of the current dynamics, if any.

Classical Representation for Planning

There are three different ways to represent classical planning problems. Each of them is equivalent in expressive power.

- **Set theoric representation.** Each state of the world is a set of propositions and each action is a syntactic expression specifying which propositions belong to the state in order for the action to be applicable and which propositions the action will add or remove to change the state of the world.
- **Classical representation.** The states and the actions are like the ones described for set theoric representation except that first order literals and logical connectives are used instead propositions.
- **State variable representation.** Each state is represented by a tuple of value n state variables $\{x_1, \dots, x_n\}$ and each action is represented by a partial function that map this tuple into some other tuple of values of the n states.

Classical Representation

The classical representation scheme generalize the set theoric representation scheme using notation derived from first order logic.

- States are represented as set of logical atoms that are true or false within some interpretation.
- Actions are represented by planning operators that change the truth values of theses atoms.

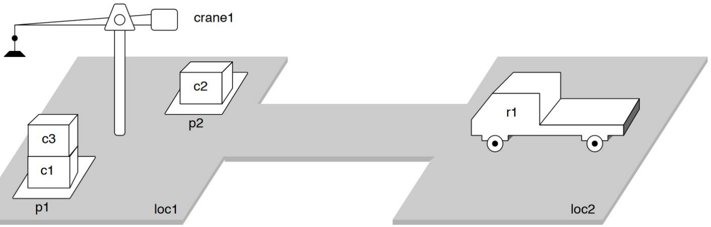
States Representation

The classical planning language is built on a first order language L.

Definition (State)

A state is a set of ground atoms of $\mathcal{L}$. $\mathcal{L}$ has no function symbols. Thus the set S of all possible states is guaranteed to be finite. As in the set of theoric representation scheme, an atom p holds in s iff $p \in s$. If g is a set of literals, we will say that s satisfies g (denoted $s \models g$) when there is a substitution σ such that every positive literal of $\sigma(g)$ is in s and no negated literal of $\sigma(g)$ is in s.

States Representation Example



Initial state s_0 = attached(p1, loc1), attached(p2, loc1) ; in(c1, p1, in(c3,p1), top(c3, p1), on(c3, c1), on(c1, pallet) in(c2, p2), top(c2, p2), on(c2,pallet),belong(crane1, loc1), empty(crane1), adjacent(loc1, loc2), adjacent(loc2, loc1), at(r1,loc2), occupied(loc2), unloaded(r1)

Planning Operator Definition

The planning operators define the transition function γ of the state transition system.

Definition (Planning Operator)

A planning operator is a triple $o = (\text{name}(o), \text{precond}(o), \text{effects}(o))$ whose elements are follows :

- name(o), the name of the operator, is a syntactic expression of the form $n(x_1, \ldots, x_k)$ where n is a symbol called an operator symbol (n is unique in $\mathcal{L}$) and $x_1, \ldots, x_k$ are all variable symbols that appear anywhere in o.
- precond(o) and effects(o), the preconditions and effects of o, respectively are generalizations of the preconditions and the effects of the set theory action, *i.e.*, instead of being sets of proposition they are sets of literals.

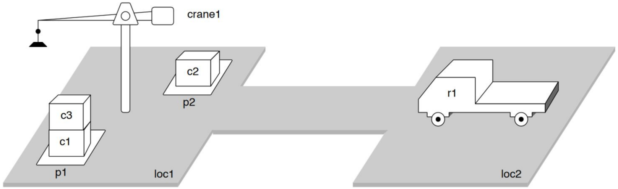
Planning Action Definition

Definition (Action)

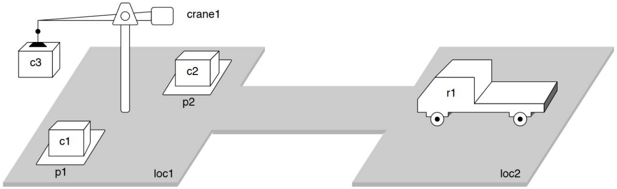
An action is any ground instance of planning operator. If a is an action and s is a state such that $\text{precond}^+(a) \subseteq s$ and $\text{precond}^-(a) \cap s = \emptyset$, then a is applicable to s, and the result of applying a to s is the state :

$$\gamma(s, a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$$

Action Example



The action $take(crane1, loc1, c3, c1, p1)$ is applicable to the state s_0 of the previous figure. The result is the state $s_5 = \gamma(s0, take(crane1, loc1, c3, c1, p1))$ shown by the figure below.



Classical Planning Domains Definition

Definition (Classical Planning Domain)

Let $\mathcal{L}$ be a first order language that has finitely many predicate symbols and constraint symbols. A classical planning domain in $\mathcal{L}$ is a restricted state transition system $\Sigma = (S, A, \gamma)$ such that :

- $S \subseteq 2^{\text{all ground atoms of}\mathcal{L}}$
- $A = \{\text{all ground instances of the operators in } \mathcal{O}\}$ where $\mathcal{O}$ is a set of operators as defined earlier
- $\gamma(s, a) = (s - \text{effects}^{-}(a)) \cup \text{effects}^{+}(a)$ if $a \in A$ is applicable to $s \in S$ and otherwise $\gamma(s, a)$ is undefined
- S is closed under γ, *i.e.*, if $s \in S$, then for every action a that is applicable to s, $\gamma(s, a) \in S$.

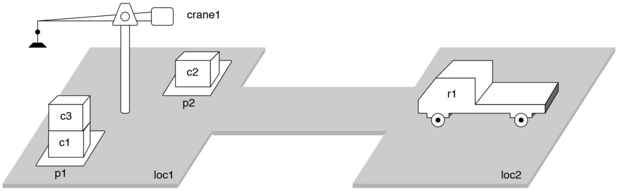
Classical Planning Problems Definition

Definition (Classical Planning Problem)

A classical planning problem is a triple $\mathcal{P} = (\mathcal{O}, s_0, g)$ where :

- $\mathcal{O}$ is the set of planning operators
- s_0, the initial state, is any state in S
- g, the goal, is any set of ground literals
- $S_g = \{s \in S \mid s \text{ satisfies } g\}$

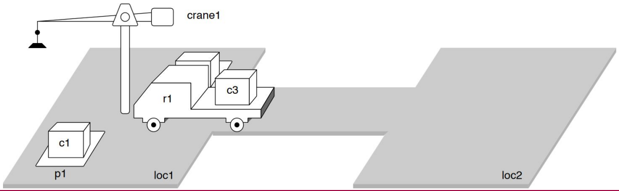
Plan Example



$\pi = \{take(crane1, loc1, c3, c1, p1), move(r1, loc2, loc1), load(crane1, loc1, c3, r1)\}$

This plan is applicable to the state s_0 shown in previous figure producing the state s_6.

$g_1 = \{loaded(r1, c3), at(r1, loc1)\}$



Extending the Classical Representation

Classical planning formalism is very restricted, extensions to it are needed in order to describe interesting domains. The most important extensions are:

- Typing variables;
- Conditional Planning Operators;
- Quantified expression;
- Disjunctive preconditions;
- Axiomatic Inference;
- etc.

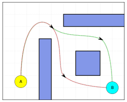
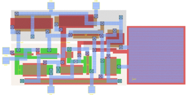
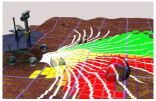
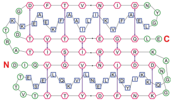
A planning language, called PDDL, has been developed to express all these extensions (PDDL stands for Planning Domain Description Language).

It's up to you !! You can do the first two practical works.

Solving Automated Planning Problems

Problems in AI

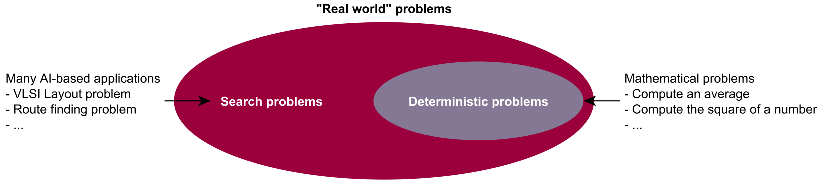
There are a lot of problems to solve in AI-based applications:

- **Automatic assembly sequencing**: Find an order in which to assemble parts of an object.
- **VLSI Layout problem**: Position million of components and connections on a chip to minimize area, shorten delays.
- **Route finding problem**: Include tools for driving directions in websites, in-car systems, etc.
- **...**

→ All these kinds of problems are called **search problems**!

Search problem



A problem is a **search problem** if there's an **algorithmic way** to find the solution.

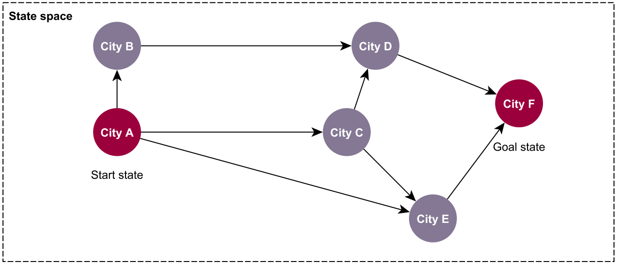
- It cannot be solve by deterministic procedure.
- It can only be solved by searching for a solution and the success is not guarenteed.

Search problem

A **search problem** is defined by:

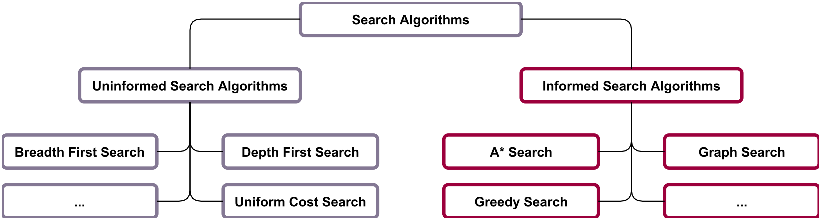
- A **state space**: Set of all possible states of the problem.
- A **start state**: The initial state of the problem.
- A **goal state**: The goal state to reach to solve the problem.
- A **solution plan**: The sequence of actions to reach the goal state.

Note: Search problems can be often represented using graphs.



Informed Search Algorithms

To solve search problems, one must use **search algorithms**!



No additional information

Receive additional information

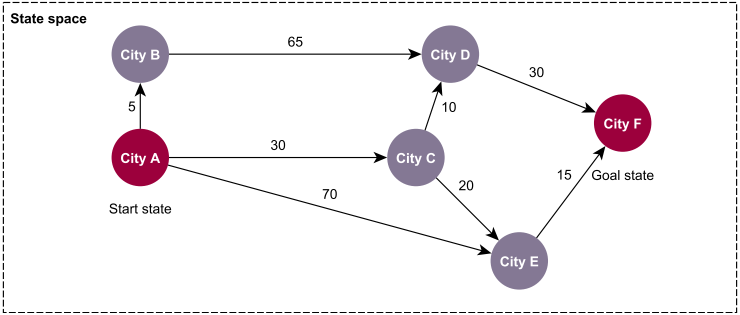
→ To **solve large problems** with large number of possible states, **problem-specific knowledge** needs to be added to increase the efficiency of search algorithms. This knowledge is obtained by a **heuristic** function.

What is a heuristic ?

Definition

A **heuristic function**, also called simply a heuristic, is a function that ranks alternatives in search algorithms: It estimates how close a state is to the goal state [Pearl, 1984].

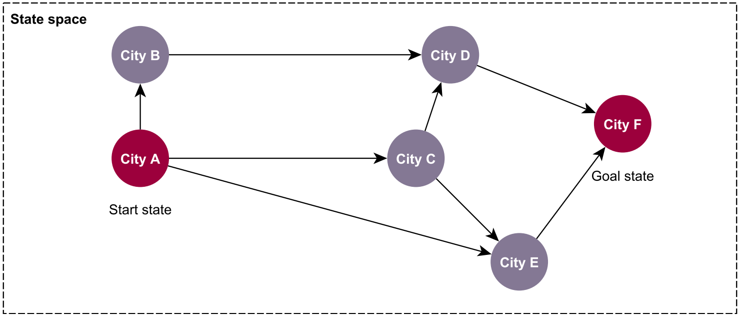
Example: `SUM` heuristic (*sum the cost of all distance to the goal*)



What is a heuristic ?

Definition
A **heuristic function**, also called simply a heuristic, is a function that ranks alternatives in search algorithms: It estimates how close a state is to the goal state [Pearl, 1984].

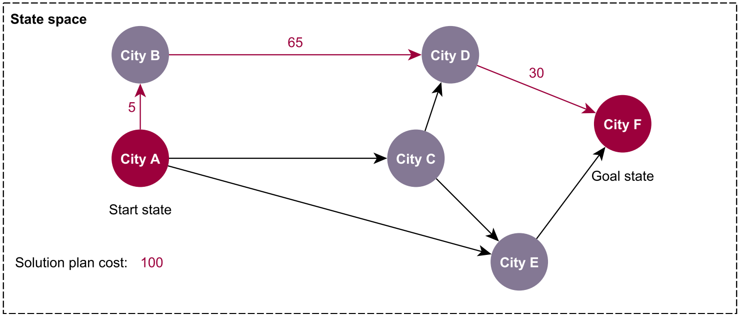
Example: `SUM` heuristic (*sum the cost of all distance to the goal*)



What is a heuristic ?

Definition
A **heuristic function**, also called simply a heuristic, is a function that ranks alternatives in search algorithms: It estimates how close a state is to the goal state [Pearl, 1984].

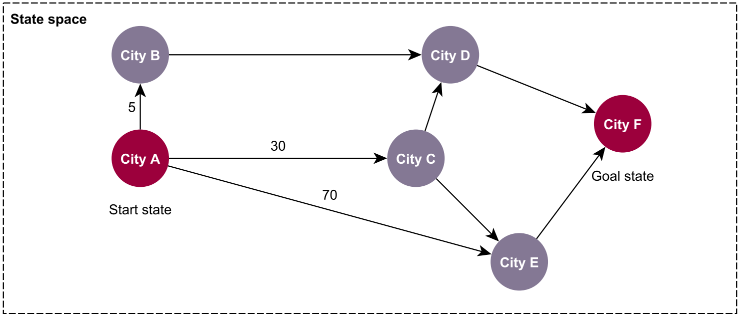
Example: `SUM` heuristic (*sum the cost of all distance to the goal*)



What is a heuristic ?

Definition
A **heuristic function**, also called simply a heuristic, is a function that ranks alternatives in search algorithms: It estimates how close a state is to the goal state [Pearl, 1984].

Example: `SUM` heuristic (*sum the cost of all distance to the goal*)

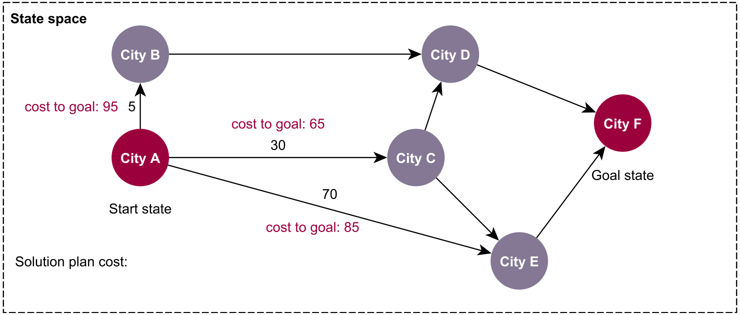


What is a heuristic ?

Definition

A **heuristic function**, also called simply a heuristic, is a function that ranks alternatives in search algorithms: It estimates how close a state is to the goal state [Pearl, 1984].

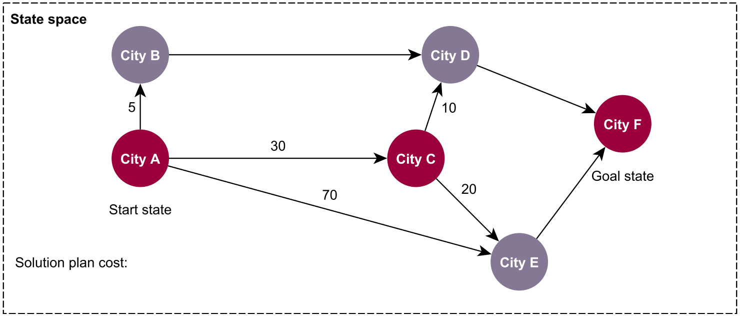
Example: `SUM` heuristic (*sum the cost of all distance to the goal*)



What is a heuristic ?

Definition
A **heuristic function**, also called simply a heuristic, is a function that ranks alternatives in search algorithms: It estimates how close a state is to the goal state [Pearl, 1984].

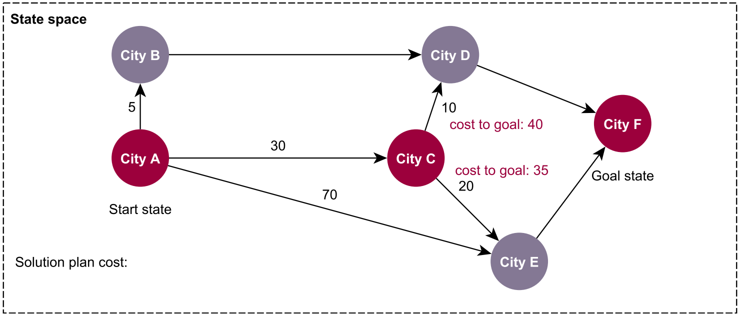
Example: `SUM` heuristic (*sum the cost of all distance to the goal*)



What is a heuristic ?

Definition
A **heuristic function**, also called simply a heuristic, is a function that ranks alternatives in search algorithms: It estimates how close a state is to the goal state [Pearl, 1984].

Example: `SUM` heuristic (*sum the cost of all distance to the goal*)

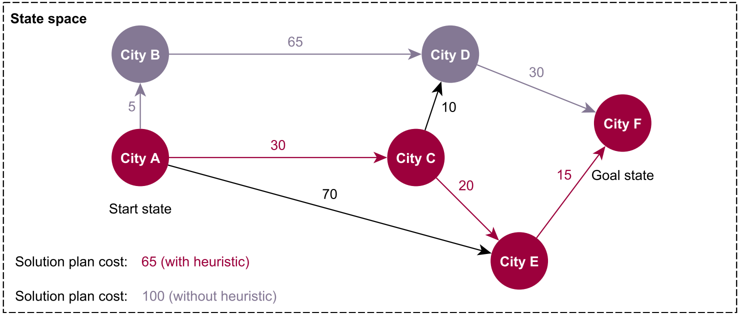


What is a heuristic ?

Definition

A **heuristic function**, also called simply a heuristic, is a function that ranks alternatives in search algorithms: It estimates how close a state is to the goal state [Pearl, 1984].

Example: `SUM` heuristic (*sum the cost of all distance to the goal*)



Admissible heuristic

Admissible heuristic definition [Norvig and Russell, 2002]
A heuristic function is said to be **admissible** if it never overestimates the cost of reaching the goal. The cost it estimates to reach the goal is not higher than the lowest possible cost from the current state: It is **optimistic**.

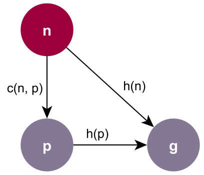
Formulation
n is a node wich represents a state
h is a heuristic function
$h(n)$ is cost indicated by h to reach a goal from n
$h'(n)$ is the optimal cost to reach a goal from n

$$h(n) \text{ is admissible if, } \forall n \; h(n) \leq h'(n)$$

Consistent heuristic

Consistent heuristic definition [Norvig and Russell, 2002]
A heuristic function is said to be **consistent**, if its estimate is always less than or equal to the estimated distance from any neighbouring state to the goal, plus the cost of reaching that neighbour state.

Formulation
n is a node wich represents a state
h is a heuristic function
p is any descendant of n
g is any goal node
$c(n, p)$ is the cost of reaching node p from n



$$h(n) \leq c(n, p) + h(p) \text{ and } h(g) = 0$$

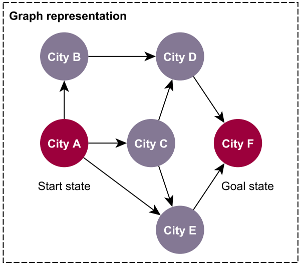
Note: A consistent heuristic is also admissible (the reverse is not always true) [Edelkamp and Schroedl, 2011].

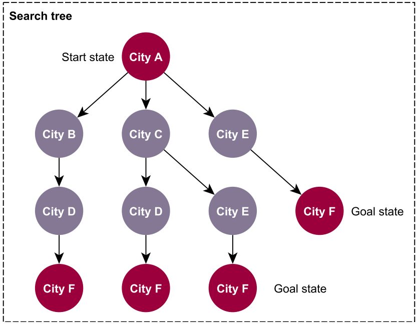
Search process in search algorithms

The **search process** in a search algorithms consists in **exploring the state space** from the initial state to the goal state.

Search algorithms uses a **search tree** which represents the exploration trace:

- **Search tree** is built during the search process.
- **Branches** correspond to explored paths and **nodes** to the exploration fringe.





Search tree expanding rules

Search algorithms differ in the way they determine the next node to expand in the search tree:

- **Uninformed Search**: Rigid procedure with no knowledge of the cost of a given node to the goal.
- **Informed Search**: Knowledge of the worth of expanding a node n is given in the form of an evaluation function $f(n)$, which assigns a real number to each node. Mostly, $f(n)$ includes as a component a heuristic function $h(n)$, which estimates the costs of the cheapest path from n to the goal.

Best-First Search

Best-first search is an instance of the general `Tree-Search` algorithm in which *frontier* is a priority queue ordered by an evaluation function f.

$\rightarrow$ Best-first search expands the node with the **"best"** f-value first. It could allow revising previous nodes.

```
function TREE-SEARCH(problem):
   Data: frontier, an ordered priority queue
1  initialize the frontier using the initial state of problem;
2  while frontier is not empty do
3      choose a leaf node with the best f-value and remove it from
         the frontier;
4      if the node contains a goal state then
5          return the corresponding solution;
6      end
7      expand the chosen node, adding the resulting nodes to the
         frontier;
8  end
9  return failure;
```

Greedy Best First Search

A **best-first search** using $f(n) = h(n)$ as the evaluation function is called a **greedy search**.

$\rightarrow$ Greedy best-first search expands the nodes estimated to be closest to goal.

Greedy Best First Search properties

- A good heuristic might reduce search time drastically.
- Non-optimal (*it not guarantees to return the best solution*).
- Incomplete (*it not guarantees to return a correct answer for any arbitrary input*); Can be stuck in a loop.
- Keep all nodes in memory.

Can we do better?

A*: Minimization of the estimated path costs

A* combines the **greedy search** with the **uniform-search strategy**: Always expand node with lowest $f(n)$ first.

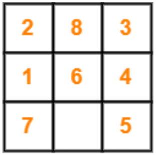
$g(n) =$ actual cost from the initial state to n
$h(n) =$ estimated cost from n to the next goal
$f(n) = g(n) + h(n)$ the estimated cost of the cheapest solution through n

A* requires that h is admissible to be **complete** and **optimal**.

A* Search

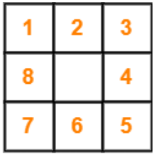
```
function AStar(problem):
    Data: OPEN and CLOSED, two lists
1   Place the starting node into OPEN using the initial state of problem;
2   while OPEN is not empty do
3       Select the node p from OPEN which has the smallest value of
          evaluation function $f(n) = g(n) + h(n)$;
4       if p is a goal state then
5           return the corresponding solution;
6       end
7       Expand node n and generate all of its p successors, and put n into
          CLOSED;
8       foreach p successors do
9           Check whether p is already in the OPEN or CLOSED, if not
              then compute evaluation function for p and place into OPEN;
10      end
11  end
12  return failure;
```

A* Search example (8-puzzle problem)



2	8	3
1	6	4
7		5

Initial State



1	2	3
8		4
7	6	5

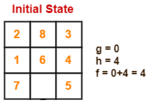
Final State

Find the most cost-effective path to reach the final state from initial state using A* Algorithm.

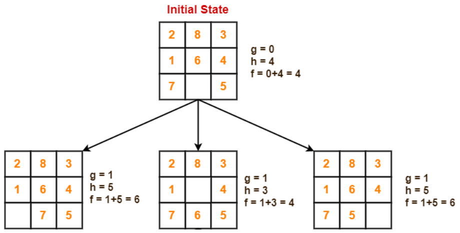
Consider:

- $g(n) =$ Depth of node
- $h(n) =$ Number of misplaced tiles.

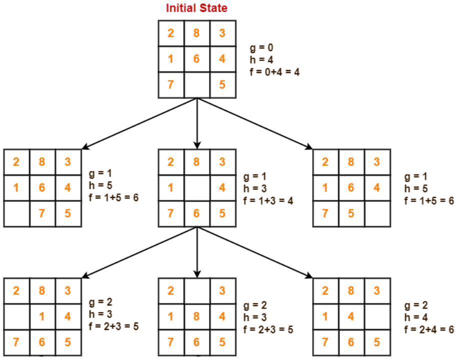
A* Search example (8-puzzle problem)



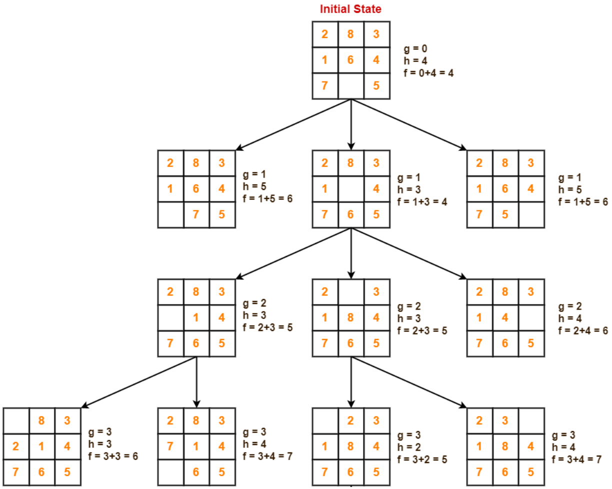
A* Search example (8-puzzle problem)



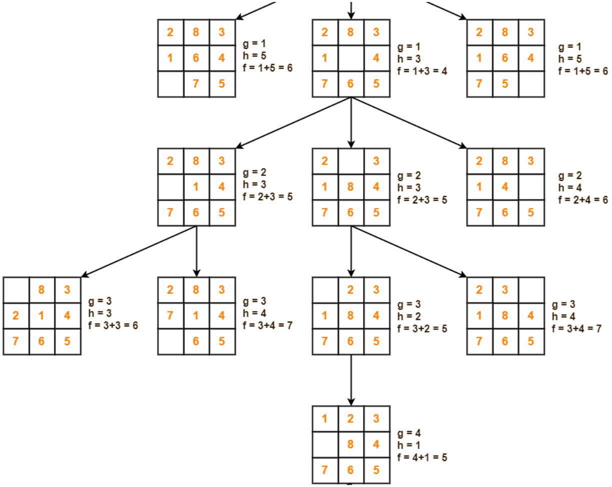
A* Search example (8-puzzle problem)



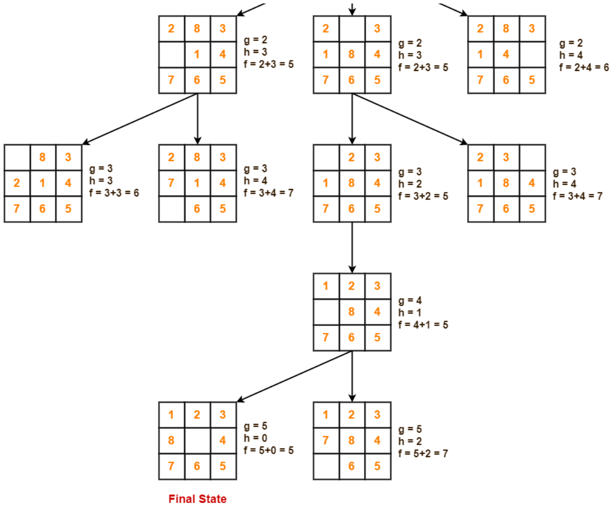
A* Search example (8-puzzle problem)



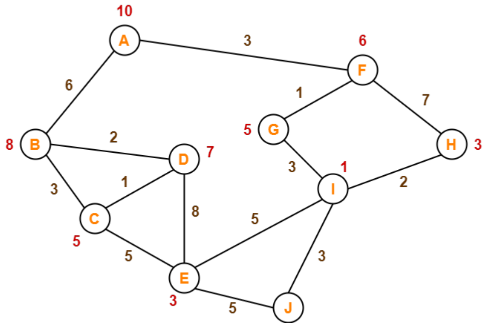
A* Search example (8-puzzle problem)



A* Search example (8-puzzle problem)

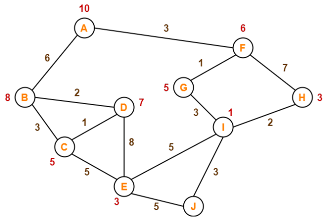


A* Search example (graph problem)



Find the most cost-effective path to reach from start state A to final state J using A* Algorithm.

A* Search example (graph problem)



Step 1:

We start with node A. Node B and Node F can be reached from node A.

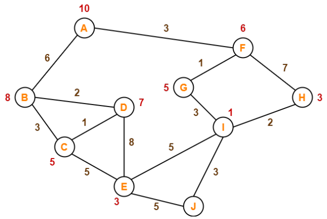
A* Algorithm calculates $f(B)$ and $f(F)$:

- $f(B) = 6 + 8 = 14$
- $f(F) = 3 + 6 = 9$

Since $f(F) < f(B)$, so it decides to go to node F.

Path: A $\rightarrow$ F

A* Search example (graph problem)



Step 2:

Node G and Node H can be reached from node F.

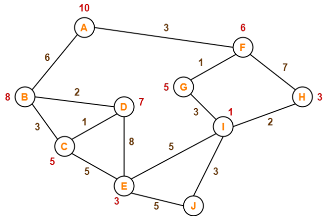
A* Algorithm calculates $f(G)$ and $f(H)$.

- $f(G) = (3 + 1) + 5 = 9$
- $f(H) = (3 + 7) + 3 = 13$

Since $f(G) < f(H)$, so it decides to go to node G.

Path: A $\rightarrow$ F $\rightarrow$ G

A* Search example (graph problem)



Step 3:

Node I can be reached from node G.

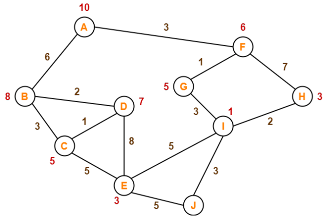
A* Algorithm calculates $f(I)$.

- $f(I) = (3 + 1 + 3) + 1 = 8$

It decides to go to node I.

Path: A $\rightarrow$ F $\rightarrow$ G $\rightarrow$ I

A* Search example (graph problem)



Step 4:

Node E, Node H and Node J can be reached from node I.

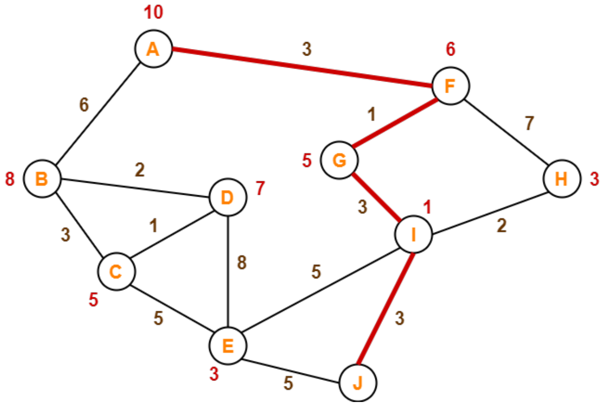
A* Algorithm calculates $f(E)$, $f(H)$ and $f(J)$.

- $f(E) = (3 + 1 + 3 + 5) + 3 = 15$
- $f(H) = (3 + 1 + 3 + 2) + 3 = 12$
- $f(J) = (3 + 1 + 3 + 3) + 0 = 10$

Since $f(J)$ is least, so it decides to go to node J.

Path: A $\rightarrow$ F $\rightarrow$ G $\rightarrow$ I $\rightarrow$ J

A* Search example (graph problem)



Properties of A*

- Optimal.
- Complete unless there are infinitely many nodes with $f \leq f(G)$.
- Keep all nodes in memory.
- A* search algorithm has some complexity issues.

Local Search Methods

In many optimization problems, the state space is the space of all possible complete solutions:

- We have an objective function that tells us how “good” a given state is, and we want to find the solution (goal) by minimizing or maximizing the value of this function.
- The start state may not be specified.
- The path to the goal doesn’t matter.

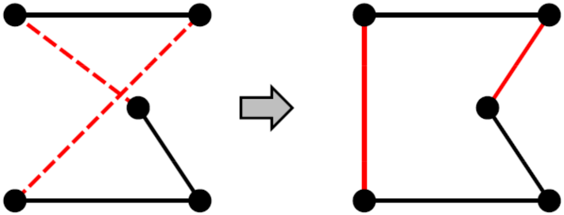
$\rightarrow$ In such cases, we can use local search algorithms that keep a single “current” state and gradually try to improve it.

Traveling Salesman problem example

Find the shortest tour connecting *n* cities !

- State space: All possible tours.
- Objective function: Length of tour.

What's a possible local improvement strategy? Start with any complete tour, perform pairwise exchanges, ...



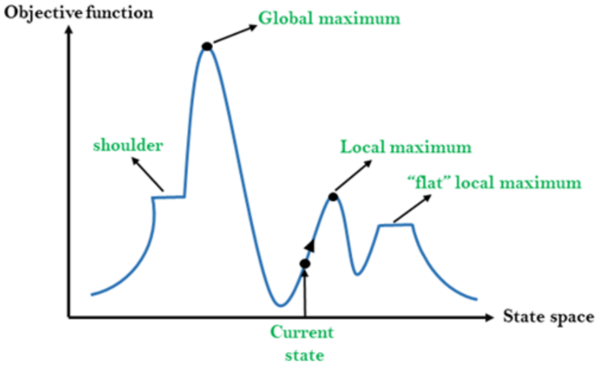
Hill climbing search

Hill climbing algorithm moves in the direction of increasing value to find the best solution. It terminates when it reaches a peak where no neighbor has a higher value.

```
function HillClimbing(startNode):
currentNode = startNode;
while currentNode is not null do
    neighborsNode = getNeighbors(currentNode);
    nextEval = -1;
    nextNode = null;
    foreach n node in neighborsNode do
        if getEval(n) > nextEval then
            nextNode = n;
            nextEval = getEval(n);
        end
    end
    if nextEval ≤ getEval(currentNode) then
        return currentNode;
    end
    currentNode = nextNode;
end
```

Hill climbing search

- Not optimal and not complete.
- Can get stuck in local optima
- Variants: Choose first better successor, randomly choose among better successors,...



Classes of search

Different classes of search [Norvig and Russell, 2002]:

Class	**Name**	**Operation**
Any Path Uninformed	Depth-First Breadth-First	Systematic exploration of whole tree until a goal nod is found
Any Path Informed	Best-First	Uses heuristic measure of goodness of a node
Optimal Uninformed	Uniform-Cost	Uses path "length" measure
Optimal Informed	A*	Uses path "length" measure and heuristic

Informed vs Uninformed searches?

Comparison between informed and uninformed searches [Norvig and Russell, 2002]:

	Informed search	**Uninformed search**
Knowledge	Uses knowledge	No use of knowledge (brute-force)
Cost	Low	Comparatively high
Speed	Fast	Slow
Efficiency	Hight (less time and cost)	Not so efficient

It’s up to you !! You can do the third practical work.

References I

Edelkamp, S. and Schroedl, S. (2011).
Heuristic search: theory and applications.
Elsevier.

Norvig, P. and Russell, S. (2002).
Artificial Intelligence: A modern approach.
Prentice Hall.

Pearl, J. (1984).
Heuristics: Intelligent search strategies for computer problem solving.
Addision Wesley.