Artificial Intelligence

Topics – Unit 4

- The Planning problem
- planning with state space search
- Planning graphs
- Planning with propositional logic
- Analysis of planning approaches
- Hierarchical planning
- Conditional planning
- Continuous and Multi Agent planning.

Planning problem

- An agent interacts with the world via perception and actions.
- Perception involves sensing the world and assessing the situation.
- Creating some internal representation of the world.
- Actions are what the agent does in the domain. Planning involves reasoning about actions that the agent intends to carry out.
- *Planning* is the reasoning side of acting.
- This reasoning involves the representation of the world that the agent has, as also the representation of its actions.
- Hard constraints where the objectives *have to* be achieved completely for success.
- The objectives could also be soft constraints, or *preferences*, to be achieved as much as possible.

Types of Planning

Projection into the future
The planner searches through the possible combination of actions to find the *plan* that will work

Memory based planning
looking into the past
The agent can retrieve a plan from its memory

Planning as a state-space search problem

Planning procedures are often state-space search problem, i.e. find a path on a state-space

- **Nodes = states of the world**
- **Transitions between nodes = actions**
- **Path on the state-space = plan**
- It is possible to explore the state-space in different ways
  - forward,
  - backward
- With different strategies (breath-first, depth-first, best-first, greedy)
- Using heuristics The **STanford Research Institute Problem Solver (STRIPS)** representation enables an efficient exploration

Continued…

The most straight forward approach is to use state-space search. Because the descriptions of actions in a planning problem specify both preconditions and effects, it is possible to search in either direction: forward from the initial state or backward from the goal

Forward State-Space Search

- Planning with forward state-space search is similar to the problem-solving approach. It is sometimes called progression planning, because it moves in the forward direction.
- We start with the problem's initial state, considering sequences of actions until we reach a goal state.

Formulation of planning problem

- The initial state of the search is the initial state from the planning problem. In general each state will be set of positive ground literals; literals not appearing are false.
- The actions which are applicable to a state are all those whose preconditions are satisfied. The successor state resulting from an action is generated by adding the positive effect literals and deleting the negative effect literals.
- The goal test checks whether the state satisfies the goal of the planning problem.
- The step cost of each action is typically 1. Although it would be easy to allow different costs for different actions, this was seldom done by STRIPS planners.

Backward State-Space Search

Backward search can be difficult to implement when the goal states are described by a set of constraints which are not listed explicitly. In particular, it is not always obvious how to generate a description of the possible predecessors of the set of goal states. The STRIPS representation makes this quite easy because sets of states can be described by the literals which must be true in those states.

Formulation of planning problem

- It is a search in the reverse direction: start with the goal state, expand the graph by computing parents.
- The parents are computed by regressing actions: given a ground goal description g and a ground action a, the regression from g over a is g′: g′ = (g− ADD (a))∪ PRECOND (a).

Forward Vs Backward

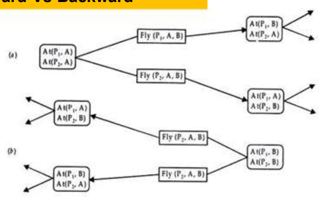


Fig. 8.5. Two approaches to searching for a plan, (a) Forward (Progression) state-space search, starting in the initial state and using the problem's actions to search forward for the goal state, (b) Backward (regression) state-space search: a belief-state search starting at the goal state(s) and using the inverse of the actions to search backward for the initial state.

Planning Graphs

- Planning graphs play a vital role in AI planning by visually representing possible states and actions that aid in decision-making. This article explores STRIP-like(STanford Research Institute Problem Solver) domains that construct and analyze the compact structure called graph planning.

Planning Graphs

Planning graphs consists of a sequence of levels that correspond to time steps in the plan.

- Level 0 is the initial state.
- Each level consists of a set of literals and a set of actions that represent what might be possible at that step in the plan.
- Records only a restricted subset of possible negative interactions among actions.

Continued…

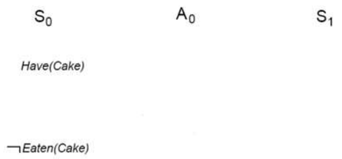
Each level consists of

- Literals = all those that could be true at that time step, depending upon the actions executed at preceding time steps.
- Actions = all those actions that could have their preconditions satisfied at that time step, depending on which of the literals actually hold.

Example

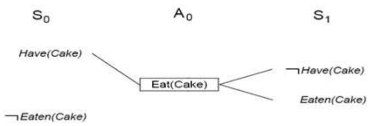
Init(Have(Cake))
Goal(Have(Cake) ∧ Eaten(Cake))
Action(Eat(Cake),
  PRECOND: Have(Cake)
  EFFECT: ¬Have(Cake) ∧ Eaten(Cake))
Action(Bake(Cake),
  PRECOND: ¬ Have(Cake)
  EFFECT: Have(Cake))

Initial Graph at L0



Create level 0 from initial problem state.

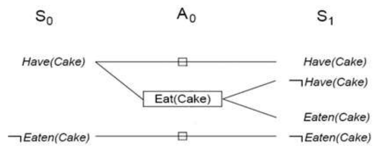
Graph at L1



Add all applicable actions.

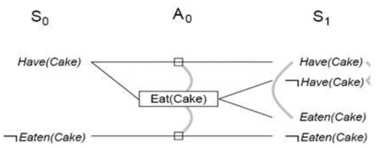
Add all effects to the next state.

Graph at L1 updated



Add *persistence actions* (inaction = no-ops) to map all literals in state S_i to state S_{i+1}.

Graph with Mutual Exclusion



Identify *mutual exclusions* between actions and literals based on potential conflicts.

Mutual exclusion

A mutex relation holds between two actions when:

- ***Inconsistent effects:*** one action negates the effect of another.
- ***Interference:*** one of the effects of one action is the negation of a precondition of the other.
- ***Competing needs:*** one of the preconditions of one action is mutually exclusive with the precondition of the other.

A mutex relation holds between two literals when:

- *One is the negation of the other.*
- *Each possible action pair that could achieve the literals is mutex (inconsistent support).*

Algorithm

function GRAPHPLAN (*problem*)
returns a solution, or failure

graph ← INITIAL-PLANNING-GRAPH (*problem*)
goals ← GOALS[*problem*]
loop do
  if *goals* all non-mutex in last level of *graph* **then do**
  solution ← EXTRACT-SOLUTION (*graph, goals,* LENGTH (*graph*))
  if *solution* ≠ *failure* **then return** *solution*
  else if NO-SOLUTION-POSSIBLE(*graph*) **then return** *failure*
graph ← EXPAND-GRAPH (*graph, problem*)

planning using propositional logic

We've seen that plans might be extracted from a knowledge base via *theorem proving*, using *first order logic (FOL)* and *situation calculus*.

BUT: this might be computationally infeasible for realistic problems.

Sophisticated techniques are available for testing *satisfiability* in *propositional logic*, and these have also been applied to planning.

The basic idea is to attempt to find a model of a sentence having the form

description of start state
∧ descriptions of the possible actions
∧ description of goal

We attempt to construct this sentence such that:

- If M is a model of the sentence then M assigns `true` to a proposition if and only if it is in the plan.
- Any assignment denoting an incorrect plan will not be a model as the goal description will not be `true`.
- The sentence is unsatisfiable if no plan exists.

1. **Theorem Proving in First-Order Logic (FOL):**
   - Early planning systems used **theorem proving** with **FOL** and **situation calculus**. This method encodes the planning problem into a logic-based system where a solution can be derived by proving that a sequence of actions leads to a goal.
   - **Drawback:**
     - While this method is expressive, it is computationally infeasible for large, realistic problems due to the complexity of theorem proving.
2. **Satisfiability (SAT) in Propositional Logic:**
   - More efficient methods are available using **propositional logic** (a less expressive but simpler form of logic) to test whether a set of logical statements can be satisfied simultaneously.
   - This method transforms the planning problem into a series of propositions and checks if there is a way to assign truth values that satisfy these propositions.

3. **Constructing the SAT Formula for Planning:**
   - The planning problem is formulated as a sentence combining:
     - **Start state description:** The initial conditions of the problem.
     - **Possible actions:** Representing the actions that can be taken.
     - **Goal description:** What needs to be achieved.
   - **How it works:**
     - A model (M) of this formula is constructed.
     - If M is a valid model (i.e., a set of assignments to variables), then the plan is feasible.
     - If no such model exists, it means no plan can satisfy the goal.

Planning in the propositional logic

- Early work on deductive planning viewed plans as proofs that lead to a desired goal (theorem).
- Planning as satisfiability testing was proposed in 1992.
  1. A propositional formula represents all length n action sequences from the initial state to a goal state.
  2. If the formula is satisfiable then a plan of length n exists.
- Satisfiability planning is the best approach to solve difficult planning problems.
  Heuristic search is often more efficient on very big but easy problems.

---

1. **Deductive Planning (Theorem Proving):**
   - In early AI, planning was viewed as **deductive reasoning**, where the plan is like a proof that leads to the goal state. However, this approach is slow and unsuitable for complex problems.
2. **Planning as Satisfiability (SAT):**
   - Introduced in 1992, this method encodes the entire planning problem as a **propositional formula**:
     - It represents all possible action sequences from the initial state to the goal state.
     - The goal is to check if this propositional formula is **satisfiable.**
     - If satisfiable, then there exists a sequence of actions (a plan) that leads to the goal.
3. **SAT-Based Planning:**
   - SAT-based planning is highly effective for complex, difficult planning problems.
   - It's often more efficient than heuristic search methods for large but structured problems because it systematically checks the satisfiability of all possible action sequences.

Planning in the propositional logic

1. Represent actions (= binary relations) as propositional formulae.
2. Construct a formula saying "execute one of the actions".
3. Construct a formula saying "execute a sequence of n actions, starting from the initial state, ending in a goal state."
4. Test the satisfiability of this formula by a satisfiability algorithm.
5. If the formula is satisfiable, construct a plan from a satisfying valuation.

Example

Propositional logic for planning

Two roof-climbers want to *swap places*:

Start state:

$$S = \mathtt{At}^0(\mathtt{a},\mathtt{spire}) \wedge \mathtt{At}^0(\mathtt{b},\mathtt{ground}) \wedge \neg\mathtt{At}^0(\mathtt{a},\mathtt{ground}) \wedge \neg\mathtt{At}^0(\mathtt{b},\mathtt{spire})$$

Remember that an expression such as $\mathtt{At}^0(\mathtt{a},\mathtt{spire})$ is a *proposition*. The super-scripted number now denotes time.

Example continued…

Propositional logic for planning

Goal:

$$G = \mathtt{At}^1(\mathtt{a},\mathtt{ground}) \wedge \mathtt{At}^1(\mathtt{b},\mathtt{spire}) \wedge \neg\mathtt{At}^1(\mathtt{a},\mathtt{spire}) \wedge \neg\mathtt{At}^1(\mathtt{b},\mathtt{ground})$$

Actions: can be introduced using the equivalent of successor-state axioms

$$\mathtt{At}^1(\mathtt{a},\mathtt{ground}) \leftrightarrow (\mathtt{At}^0(\mathtt{a},\mathtt{ground}) \wedge \neg\mathtt{Move}^0(\mathtt{a},\mathtt{ground},\mathtt{spire})) \vee (\mathtt{At}^0(\mathtt{a},\mathtt{spire}) \wedge \mathtt{Move}^0(\mathtt{a},\mathtt{spire},\mathtt{ground})) \quad (1)$$

Denote by A the collection of all such axioms.

Example continued…

Propositional logic for planning

We will now find that $S \wedge A \wedge G$ has a model in which $\mathtt{Move}^0(\mathtt{a},\mathtt{spire},\mathtt{ground})$ and $\mathtt{Move}^0(\mathtt{b},\mathtt{ground},\mathtt{spire})$ are $\mathtt{true}$ while all remaining actions are $\mathtt{false}$.

In more realistic planning problems we will clearly not know in advance at what time the goal might expect to be achieved.

We therefore:

- Loop through possible final times T.
- Generate a goal for time T and actions up to time T.
- Try to find a model and extract a plan.
- Until a plan is obtained or we hit some maximum time.

Example continued…

Propositional logic for planning

Life becomes more complicated still if a third location is added: hospital.

$$Move^0(a, spire, ground) \wedge Move^0(a, spire, hospital)$$

is perfectly valid and so we need to specify that he can't move to two places simultaneously

$$\neg(Move^1(a, spire, ground) \wedge Move^1(a, spire, hospital))$$
$$\neg(Move^1(a, ground, spire) \wedge Move^1(a, ground, hospital))$$
$$\vdots$$

and so on.

These are *action-exclusion* axioms.

Unfortunately they will tend to produce *totally-ordered* rather than *partially-ordered* plans.

Hierarchical planning

Hierarchical Planning is an Artificial Intelligence (AI) problem solving approach for a certain kind of *planning problems* -- the kind focusing on *problem decomposition*, where problems are step-wise refined into smaller and smaller ones until the problem is finally solved. A solution hereby is a sequence of actions that's executable in a given initial state. This form of hierarchical planning is usually referred to as Hierarchical Task Network (HTN) planning.

Hierarchical planning

Principle

- hierarchical organization of 'actions'
- complex and less complex (or: abstract) actions
- lowest level reflects directly executable actions

Procedure

- planning starts with complex action on top
- plan constructed through action decomposition
- substitute complex action with plan of less complex actions (pre-defined plan schemata; or learning of plans/plan abstraction)
- overall plan must generate effect of complex action

Hierarchical planning

Hierarchical Planning / Plan Decomposition

Plans are organized in a hierarchy. Links between nodes at different levels in the hierarchy denote a decomposition of a "complex action" into more primitive actions (operator expansion).

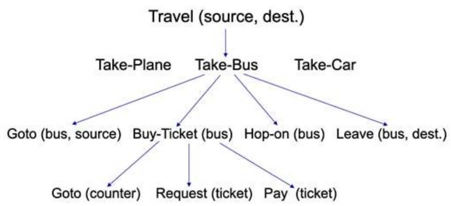
Example:

move (x, y, z)

operator expansion: pickup (x, y) putdown (x, z)

The lowest level corresponds to executable actions of the agent.

Hierarchical planning- Example



Components of Hierarchical Planning

Artificial intelligence (AI) hierarchical planning usually entails the following essential elements:

- **High-Level Goals:** High-level goals provide the initial direction for the planning process and guide the decomposition of tasks into smaller sub-goals.
- **Tasks:** Tasks are actions that need to be performed to accomplish the high-level goals.
- **Sub-Goals:** Sub-goals are intermediate objectives that contribute to the accomplishment of higher-level goals. Sub-goals are derived from

Components of Hierarchical Planning

Hierarchical Structure: Hierarchical planning organizes tasks and goals into a hierarchical structure, where higher-level goals are decomposed into sub-goals, and sub-goals are further decomposed until reaching primitive actions that can be directly executed.

Task Dependencies and Constraints: Hierarchical planning considers dependencies and constraints between tasks and sub-goals. These dependencies determine the order in which tasks need to be executed and any preconditions that must be satisfied before a task can be performed.

Components of Hierarchical Planning

Plan Representation: Plans in hierarchical planning are represented as hierarchical structures that capture the sequence of tasks and sub-goals required to achieve the high-level goals. This representation facilitates efficient plan generation, execution, and monitoring.

Plan Evaluation and Optimization: Hierarchical planning involves evaluating and optimizing plans to ensure they meet the desired criteria, such as efficiency, feasibility, and resource utilization. This may involve iteratively refining the plan structure or adjusting task priorities to improve performance.

Hierarchical Planning Techniques in AI

Hierarchical Task Networks are used for representing and reasoning about hierarchical task decomposition. HTNs consist of a set of tasks organized into a hierarchy, where higher-level tasks are decomposed into sequences of lower-level tasks. HTNs provide a structured framework for planning and execution, allowing for the efficient generation of plans that satisfy complex goals and constraints.

Hierarchical Planning Techniques in AI

Hierarchical Reinforcement Learning is extension of reinforcement learning, it leverages hierarchical structures to facilitate learning and decision-making in complex environments. In HRL, tasks are organized into a hierarchy of sub-goals, and the agent learns policies for achieving these sub-goals at different levels of abstraction. By learning hierarchies of policies, HRL enables more efficient exploration and exploitation of the environment, leading to faster learning and improved performance.

Hierarchical Planning Techniques in AI

Hierarchical state space search is a planning technique that involves exploring the state space of a problem in a hierarchical manner. Instead of directly exploring individual states, hierarchical state space search organizes states into hierarchical structures, where higher-level states represent abstract representations of the problem space. This hierarchical exploration allows for more efficient search and pruning of the state space, leading to faster convergence and improved scalability.

Challenges and Limitations of Hierarchical Planning

Although hierarchical planning has many benefits, there are some challenges and limitations as well:

- **Planning Complexity:** As the number of tasks rises, both the initial decomposition and the following planning may become computationally demanding.
- **Adaptability:** Modifications to the environment or the main objectives may call for a thorough re-planning process that may demand a large amount of resources.

Conditional planning

It works regardless of the outcome of an action. It deals with uncertainty by inspecting what is happening in the environment at predetermined points in the plan. It can take place in fully observable and non-deterministic environments. It will take actions and must be able to handle every outcome for the action taken.

Continued…

What's Conditional Planning?

- It's a planning method for handling bounded indeterminacy.
  - Bounded Indeterminacy – actions can have unpredictable effects, but the possible effects can be determined.
    Ex: flip a coin (outcome will be head or tail)
- It constructs a conditional plan with different branches for the different contingencies that could arise.
- It's a way to deal with uncertainty by checking what is actually happening in the environment at predetermined points in the plan. (Conditional Steps)
- Example:
  - Check whether SFO airport is operational. If so, fly there; otherwise, fly to Oakland.

Continued…

Three kind of Environments

- Fully Observable
  - The agent always knows the current state
- Partially Observable
  - The agent knows only a certain amount about the actual state. (much more common in real world)
- Unknown
  - The agent knows nothing about the current state

Continued…

Conditional Planning in Fully Observable Environments

- Agent used conditional steps to check the state of the environment to decide what to do next.
- Plan information stores in a library
  Ex: Action(Left) → Clean v Right
- Syntax:
  If <test> **then** plan_A **else** plan_B

Continued…

AND-OR-Graph-Search

- Modify Minimax Algorithm
  - MAX node → OR node
    - It returns a single move/plan for an action
  - MIN node → AND node
    - It returns a series of plans for each action

Continued…

Example of AND-OR-Graph

Plan:	Take her out		
Action:	Acquire information	Ask her	
AcquireInfo Action:	Ask her friend	Borrow her diary	Borrow her PDA
AskHerFriend Action:	Go to Friend	Send Message	

Continued…

```
Function AND-OR-GRAPH-SEARCH (problem) returns a conditional
                                              plan, or failure
   OR-SEARCH (INITIAL-STATE[problem], problem,[])
________________________________________________________________
Function OR-SEARCH(state,problem, path) returns a conditional plan,
                                                        or failure
   if GOAL-TEST[problem](state) then return the empty plan
   if state is on path then return failure
   for each action, state_set in SUCCESSORS[problem](state) do
        plan ← AND-SEARCH( state_set, problem, [state | path])
        if plan != failure then return [action | plan]
   return failure
________________________________________________________________
Function AND-SEARCH( state_set, problem, path) returns a
                                        conditional plan, or failure
   for each S(i) in state_set do
        plan(i) ← OR-SEARCH( S(i), problem, path)
        if plan = failure then return failure
   return [if s(1) then plan(1) else if s(2) then plan(2) else… if S(n-1)
   then plan(n-1) else plan(n)]
```

Continued…

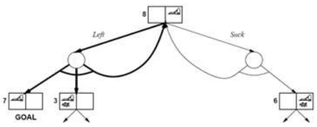
Good Thing About AND-OR Graph Search

- The way it deal with cycle
  - If the current state is identical to a state on the path from the root, then it returns with failure; it means that if there is a noncyclic solution, it must be reachable from the earlier incarnation of the current state, so the new incarnation can be discarded
  - Algorithm can terminate in every finite state space
- But it doesn't check whether the current state is a repetition of a state on some other path from the root

Continued…

Failure of AND-OR-Graph-Search

- "Triple Murphy" Vacuum World – there are no longer any acyclic solution, and this algorithm would return with failure



Continued…

Possible Solution For AND-OR-Graph-Search Failure

- Solution:
  - Cyclic solution – keep trying Left or Right until it is clean, but it doesn't guaranteed succeed.

[L1 : Left, **if** AtR **then** L1 **elseif** CleanL **then** [] **else** Suck]

Partially Observable Environments

- It used the same AND-OR-Graph-Search algorithm, but the belief states will defy differently.
- Three choices for belief states:
  1. Sets of full state descriptions
     Ex: {(AtR and CleanR and CleanL), (AtR and CleanR and not CleanL)}
     (not good, the size will become O(2^n))
  2. Logical sentences that capture exactly the set of possible worlds (Open-world Assumption)
     Ex: AtR and CleanR
     (not that good, it can't represent all domains)
  3. Knowledge Propositions – describe the agent's knowledge (Closed-world Assumption)
     Ex: K(P) → means the agent knows that P is true, if it doesn't appear, it's assumed false.

Partially Observable Environments

- Any scheme capable of representing every possible belief state will require O(log2(2^2n)) = O(2^n) bit to represent each one in the worst case.
- Two kind of Sensing
  1. Automatic – auto. Check the state
  2. Active – agents must use sensory action to check the state of environment. ex: CheckDirt

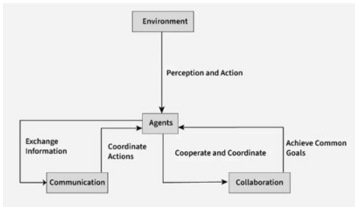
Disadvantages

- Agents are not capable of making tradeoffs between the probability of success and the cost of plan construction
- Conditional Planning is harder than NP
  - NP means that a candidate solution can be checked to see whether it really is a solution in polynomial time
- Use a lot of space O(2^n)

Continuous and Multi Agent planning

Multi Agent Systems (MAS)

- A multi-agent system is a system in which there are several agents in the same environment which co-operate at least part of the time.
- Complexity of the path planning systems for MAS (MASPP) increase exponentially with the number of moving agents.



Continued…

Problems with MASPP

- Possible problems of applying ordinary PP methods to MAS are,
- Collisions,
- Deadlock situations, etc.
- Problems with MASPP are,
- Computational overhead,
- Information exchange,
- Communication overhead, etc.

Multiagent Planning System Architecture

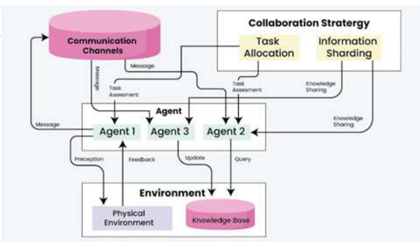
At its core, multiagent planning system involves:

Goal Specification: Agent grouping / coordination with a single objective or target on which they apply their efforts.

Knowledge Sharing: For instance, the missions may exchange important intelligence that can be an integral part of decision making.

Action Coordination: Enacting meticulous actions coherently among agents in the side stepping of conflicts and in the disease of synergy.

Adaptation: Strategy to include planning for overcoming the changing challenges or goal that may evoke on a constant basis and be capable to adapt.



Approaches in Multi Agent Environment

Centralized Planning: In the case of the centralized planning, one unit or the central controller decides what to do for all the agents from the whole system's state.

Decentralized Planning: Decentralized planning is the process where each agent makes its own decisions depending on the information available locally and the limited communication with other agents.

Distributed Planning: The so-called distributed planning is a mixed-up method where agents have to share some info and adjust their plans in order to obtain the common world objectives.

Multiagent Planning Techniques

- **Distributed Problem-Solving Algorithms:** The agents in these algorithms break down the complicated problems into the easy-to-handle sub-tasks and the agents then distribute these sub-tasks among themselves.
- **Game Theory:** It furnishes a tool for studying the strategic relationships among agents.
- **Multiagent Learning:** The multiagent learning process is based on the agents' enhancement of their performance by the means of their experience and interaction with other agents.
- **Communication Protocols:** The communication and coordination of the agents that are structured and have a clear protocol of the information exchange and synchronization amongst them, is a tool for the agents to exchange the information and be synchronized.

Advantages of Multiagent Planning in AI

- **Efficiency:** Dividing tasks between all the agents can accelerate already functioning methods and processes for solving problems and making decisions.
- **Robustness:** Shared intelligence increases the system reliability allowing seamless operation despite of one or few agents faults and/or a changing environment.
- **Scalability:** The decentralized design of multiagent systems brings scalability advantage as it is easy to add more agents or components without facing integration issues.
- **Flexibility:** Agents' smartness and communications system qualities facilitate instant changes to proper reaction to changing conditions.

Applications of Multi-Agent Planning in AI

- Robotics: Coordinating Multiple Robots
- Traffic Management: Traffic-flow-optimization
- Supply Chain Management: Planning Logistics
- Multiplayer Games: Smart Agents for Strategy in a Game
- Smart Grids: Energy supply reduction in worst-case scenarios.