



EEL5840: Elements of Machine Intelligence

SubjuGator

Announcements



- Reading Assignment:
 - > Nilsson chapters 11-12
- Announcements:
 - > Tentative 2nd Exam Dates:
 - 12/1/09 (Tuesday)
 - 12/3/09 (Thursday)
- Today's Handouts in WWW:
 - > LISP Project due 12/1/09
 - > Outline Class 25
 - > www.mil.ufl.edu/eel5840
 - > Software and Notes

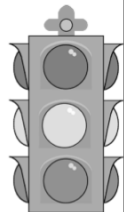
University of Florida
EEL 5840 - Class #25 - Fall 2009
© Dr. A. Armitage Armitage

1

EEL5840: Elements of Machine Intelligence

SubjuGator

Today's Menu



- Sense/Plan/Act Architecture
 - > Hierarchical Search
 - > Limited-Horizon Search (Branch-and-Bound)
 - > Search Cycles
 - > Building Reactive Procedures
- Learning Heuristic Functions
 - > Explicit Graphs
 - > Implicit Graphs
- Rewards instead of Goals - motivation

University of Florida
EEL 5840 - Class #25 - Fall 2009
© Dr. A. Armitage Armitage

2

EEL5840: Elements of Machine Intelligence

SubjuGator

Planning, Acting & Learning

- Sense/Plan/Act Cycle
 - > Search methods often fail to meet expectations in SR agents because many of the strong assumptions flounder due to:
 - Perceptual processes might not always provide the necessary information about the state of the environment (the sensors may be noisy or are often not able to sense important features). Environmental aliasing occurs when two different situations evoke the same sensory input
 - Actions might not always have their modeled effect (models may not be precise enough or the action executive may make errors when executing actions)
 - There may be adversaries present which cause interference
 - During the time it takes to plan the situation may change enough to make the plan irrelevant

University of Florida
EEL 5840 - Class #25 - Fall 2009
© Dr. A. Armitage Armitage

3

EEL5840: Elements of Machine Intelligence

SubjuGator

Planning, Acting & Learning

- Sense/Plan/Act Cycle
 - Actions may be required before a search reaches a goal state
 - The agent's computational memory may be limited
 - > There are two ways to deal with this problem
 - Use Probabilistic methods to deal formally with uncertainty in perception, environment or effect or systems
 - Briefly, a Markov Decision Process (MDP) is one where we assume that for every action executable in a certain state, the resulting state is given by a *pdf*
 - When you have imperfect perception we have a Partially Observable Markov Decision Process (POMDP) where we assume that the agent's sensory apparatus provides a *pdf* over the set of states in the world

University of Florida
EEL 5840 - Class #25 - Fall 2009
© Dr. A. Armitage Armitage

4

EEL5840: Elements of Machine Intelligence
SubjuGator
Planning, Acting & Learning

- **Sense/Plan/Act Cycle**
 - > Nilsson proposes another paradigm, the sense/plan/act architecture
 - Even if actions occasionally produce unanticipated effects and if agents sometimes cannot decide which world state it is in, these difficulties can be adequately dealt with by ensuring that the agent gets continuous feedback from its environment while it is executing its plan
 - A sense/plan/act agent is one which plans a sequence of actions, executes the 1st action, senses the environment, recalculates the start node and repeats the process.
 - We assume that the time required to recalculate the plan \ll time allowed to execute an action
 - In benign environments (one that is tolerant to a few missteps) errors in sensing and acting “average out” over a sequence of sense/plan/act cycles.

University of Florida
EEL 5840 - Class #27 - Fall 2009
© Dr. A. Armitage Armitage

EEL5840: Elements of Machine Intelligence
SubjuGator
Planning, Acting & Learning

Sense/Plan/Act Architecture
 It is expected that environmental feedback in these agents allow us to resolve environmental, perceptual or effector uncertainties by assuming that on the average, sensing and acting are accurate

University of Florida
EEL 5840 - Class #27 - Fall 2009
© Dr. A. Armitage Armitage

EEL5840: Elements of Machine Intelligence
SubjuGator
Planning, Acting & Learning

- **Sense/Plan/Act Cycle**
 - > The assumption that on the average sensing and acting are accurate is often realistic in many robotic applications
 - The designer’s task was to provide sensor, effector and perceptual modalities suitable to the task (the designer originally custom-tailored the agent for the task)
 - The agent can often enhance perceptual accuracy by comparing immediate sensory data with a stored model
 - > Example: In the grid world robot in chapter 5 (class 20)
 - If we assume that the sensory data is correct
 - and that the “no tight space” assumption holds
 - then the robot is able to alter its world model as shown in slide 7 of class 20, reproduced following this slide for convenience

University of Florida
EEL 5840 - Class #27 - Fall 2009
© Dr. A. Armitage Armitage

EEL5840: Elements of Machine Intelligence
SubjuGator
State Machines

- **Example**
 - It has two KSs to help correct errors, namely, a *gap filler* and a *sensory filter*.
 - The gap filler looks for tight spaces in the map and either fills them in with 1’s or expands them with additional 0’s. It fills the top ? with a 1 since no tight spaces are allowed.
 - The sensory filter looks at both the sensory data and the map and attempts to reconcile discrepancies. In the figure s_7 is ? on the BB but the sensors indicate a strong 1 \therefore change it to 1 on the BB. The map indicates 0 for s_4 but the sensors indicate $s_4=1$, it must be a bum sensor \therefore leave the s_4 in the BB alone.

Finally, the gap filler changes the left ? to 1.

University of Florida
EEL 5840 - Class #27 - Fall 2009
© Dr. A. Armitage Armitage

EEL5840: Elements of Machine Intelligence
SubjuGator
Planning, Acting & Learning

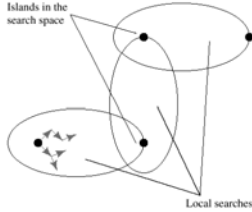
- **Approximate Search**
 - > How can we modify the search process in order to address the problem of limited computational and/or time resources at the expense of producing plans that might be sub-optimal or that might not always lead to a goal state?
 - > Notice that if our agent's 1st action has a tendency (on the average) to shorten the distance to the goal, then multiple iterations of the sense/plan/act cycle will eventually reach the goal
 - > Two ways used to reduce computational costs in searches are:
 - Search for a path to goal without requiring that the path be optimal. (We saw this when we used the Greedy Algorithm in the Traveling Salesman. This is a case analogous to using a non-admissible heuristic.)
 - Search for a partial path that does not take us all the way to a goal. Here we quit searching before reaching the goal regardless of whether the heuristic is or is not admissible. (Anytime Algorithms, Dean 1988)

University of Florida
 EEL 5840 - Class #27 - Fall 2009
 © Dr. A. Armitage Armitage

9

EEL5840: Elements of Machine Intelligence
SubjuGator
Planning, Acting & Learning

- **Approximate Search - Island-Driven Search**
 - > Heuristic knowledge from the problem domain is used to establish a sequence of "island nodes" in the search space through which it is suspected that good paths travel.



Suppose n_0 is the start node and n_g is the goal node and (n_1, n_2, \dots, n_k) is a sequence of such islands.

1. Initiate heuristic search with $s=n_0$ and $n'_g=n_1$, as the goal node
2. When search finds a path to n_1 terminate the search and start another search with $s=n_1$ & $n'_g=n_2$
3. Repeat until $n'_g=n_g$

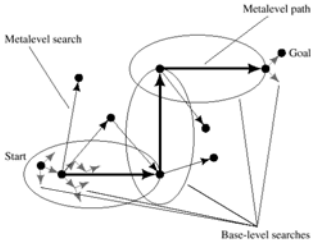
University of Florida
 EEL 5840 - Class #27 - Fall 2009
 © Dr. A. Armitage Armitage

Island-Driven Search

10

EEL5840: Elements of Machine Intelligence
SubjuGator
Planning, Acting & Learning

- **Approximate Search - Hierarchical Search**



Similar to island-driven search, but we do not have an explicit set of islands. We assume we have certain "macro operators" that can take large steps in implicit search space of islands. The *start island* and the macro operators form an implicit super graph of islands.

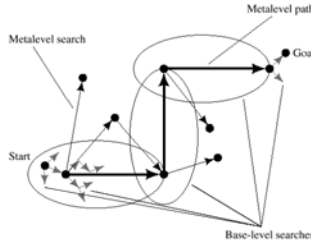
Hierarchical Search

University of Florida
 EEL 5840 - Class #27 - Fall 2009
 © Dr. A. Armitage Armitage

11

EEL5840: Elements of Machine Intelligence
SubjuGator
Planning, Acting & Learning

- **Approximate Search - Hierarchical Search**



1. Search the super-graph with a meta-level search until we produce a path of macro-operators that takes us from a node near the base-level start node s to near the base-level goal node n_g .
2. If the macro-operators are already defined in terms of the base-level operators, expand the macro path into base-level operators and connect by base-level searches as a path $s \rightarrow n_g$
3. If the macro-operators are not already defined in terms of the base-level operators, then do base-level searches along the path of island nodes found by meta-level search

Hierarchical Search

University of Florida
 EEL 5840 - Class #27 - Fall 2009
 © Dr. A. Armitage Armitage

12

EEL5840: Elements of Machine Intelligence
 SubjuGator
Planning, Acting & Learning

- Approximate Search - Hierarchical Planning Example

Hierarchical Planning
 Pushing a Block

1. A pushable block is located in one of the cells.
2. The robot's goal is to push the block to the cell marked G .
3. Assume the robot can sense all eight surrounding cells and can distinguish between the block and any unmovable objects boundaries
4. Assume the robot can move in its column or row into an adjacent free cell or into an adjacent cell containing a movable block (by pushing the block, if such motion is not blocked by an immovable barrier)

University of Florida
 EEL 5840 - Class #27 - Fall 2009
 © Dr. A. Armitage Armitage

13

EEL5840: Elements of Machine Intelligence
 SubjuGator
Planning, Acting & Learning

- Approximate Search - Hierarchical Planning Example
 - > The robot first makes a meta-level plan assuming the block can move in the same way the robot can move (the gray arrow).
 - > Now each step of the block's motion requires expanding into a base-level path.
 - The first one of the block's moves requires the robot to plan a path to a cell adjacent to the block and on the opposite side of the target location for the block. This is the dark arrow in the figure.
 - Subsequent base-level planning is trivial until the block must change direction
 - At each change of direction the robot must plan a path to a cell opposite the block's direction of motion, and so on. These are shown as dark rectangular arrows in the figure.
 - > If there is likelihood of environmental change, we take a few steps of meta-level search & the plan is altered via feedback.

University of Florida
 EEL 5840 - Class #27 - Fall 2009
 © Dr. A. Armitage Armitage

14

EEL5840: Elements of Machine Intelligence
 SubjuGator
Planning, Acting & Learning

- Approximate Search - Limited Horizon Search
 - > When you either do not have enough time to find a path to a goal or when there is a time-limit imposed on the search it may be useful to use the available time to find a path to a node thought to be on a "good" path to the goal even if this is not the goal node itself. This node, call it n^* , is a node having the smallest f value among the nodes on the search frontier when the search must be terminated.
 - > Suppose you are able to search until depth d when search must terminate. The set H of nodes at depth d are called Horizon nodes. Then $n^* = \operatorname{argmin}\{f(n)\} \quad \forall n \in H$
 - > A sense/plan/act agent takes the 1st action on the path to n^* then it senses the resulting state and iterates by searching again.

(We can expect that this 1st action toward a node having optimal heuristic merit has a good chance of being on $s \rightarrow n_1$)

University of Florida
 EEL 5840 - Class #27 - Fall 2009
 © Dr. A. Armitage Armitage

15

EEL5840: Elements of Machine Intelligence
 SubjuGator
Planning, Acting & Learning

- Approximate Search - Limited Horizon Search
 - > Limited horizon search can be efficiently performed using DFS with a depth bound of d . If f is monotonic we can obtain great reductions in search effort. As soon as the first node n_1 , on H is reached, we can terminate search below any node, n , with $f(n) > f(n_1)$. Node n cannot possibly have a descendant with an f value less than $f(n_1)$. (The f values along any path in the search tree are non-decreasing by the following results). $f(n_1)$ is called an *alpha cut-off* value & terminating below node n is an instance of Branch-and-Bound Search. Also if we have a node, $n_2 \in H$, such that $f(n_2) < f(n_1)$, then let $\alpha = f(n_2)$.

RESULT 7 If $h(n)$ is a monotone cost function then A^* does not ever need to redirect pointers (in Step 7) and $g(n) = g^*(n)$

RESULT 8 If $h(n)$ is a monotone cost function then the f values of a sequence of nodes is non-decreasing in A^*

University of Florida
 EEL 5840 - Class #27 - Fall 2009
 © Dr. A. Armitage Armitage

16

EEL5840: Elements of Machine Intelligence
Planning, Acting & Learning

- **Approximate Search - Cycles**
 - > Whenever there are uncertainties and when an agent relies on approximate plans, the use of sense/act/plan agents may result in repetitive cycles of behavior (the agent revisits a state and repeats the action previously taken there). The algorithm *Real-Time A** (RTA*) by Korf (1990) builds an explicit graph of all states actually visited and adjusts the h values of the nodes in this graph in a way that biases against taking actions leading to states previously visited.
- **Approximate Search - Reactive Procedures**
 - > Plans may be pre-computed and stored for retrieval
 - *Offline* search computes a spanning tree rooted at the goal of the state-space graph containing paths from all the nodes in the state space (by searching backward from the goal). Spanning trees can be converted to T-R programs which are completely reactive.

University of Florida
 EEL 5840 - Class #2 - Fall 2009
 © Dr. A. Armitage Armitage

EEL5840: Elements of Machine Intelligence
Planning, Acting & Learning

University of Florida
 EEL 5840 - Class #2 - Fall 2009
 © Dr. A. Armitage Armitage

EEL5840: Elements of Machine Intelligence
Planning, Acting, and Learning

- **Learning Heuristic Functions**

Useful information can also be extracted both from the experience of searching and from the experience of acting in the world.

 - > **Explicit Graphs**
 - For small state spaces it is possible to learn the heuristic function to estimate the cost to the goal.
 - Assumptions: a good model of the effects of the agent's actions and knowledge of the cost of moving from any node to its successor nodes.
 - Initialize $h(n)=0$ and start A* search. After expanding node n_i to produce successors $S(n_i) = M - \Gamma(n_i)$ and with $h(n_g)=0$ then modify as follows:

$$h(n_i) \leftarrow \min[h(n_i) + c(n_i, n_j)] \quad \forall n_j \in S(n_i) \text{ and}$$


$$c(n_i, n_j) = \text{cost of moving from } n_i \text{ to } n_j$$
 - The h values can be stored in a table of nodes because the graph is finite.
 - The first time we search it costs the same as uniform cost, however, subsequent searches to the same goal from different starting positions will be accelerated by using the learned h function. [Korf 1990]

University of Florida
 EEL 5840 - Class #2 - Fall 2009
 © Dr. A. Armitage Armitage

EEL5840: Elements of Machine Intelligence
Planning, Acting, and Learning

- **Learning Heuristic Functions**
 - > If the agent does not have a model of the effects of its actions it may still be able to learn h in the actual world.
 - > Assume: the agent can distinguish the states that it actually visits, it can name them and make an explicit graph (table) to represent states and their estimated value and the transitions caused by actions. Of course, the agent does not know the cost of its actions but rather it learns the cost upon taking an action.
 - > Start by taking an action from the start node (a random action) and transit to another state. As it visits states compute $h(n_i) \leftarrow h(n_i) + c(n_i, n_j)$ where n_i is the node in which action a is taken n_j is the resulting node, $c(n_i, n_j)$ is the revealed cost of the action, $h(n_i)$ is an estimate of the value of n_j which is 0 if n_j has never been visited before or is otherwise stored in the table.

University of Florida
 EEL 5840 - Class #2 - Fall 2009
 © Dr. A. Armitage Armitage




EEL5840: Elements of Machine Intelligence
Planning, Acting, and Learning

- Learning Heuristic Functions
 - > Whenever the agent is about to take an action at a node, n , having successor nodes stored in the graph, it chooses an action according to the policy $a = \underset{a}{\operatorname{argmin}} [h(\sigma(n,a)) + c(n_i, \sigma(n,a))]$, $\forall a$ where $\sigma(n,a)$ is the description of the state reached from node n after taking action a .
 - > The procedure begins with a random walk, eventually stumbling into a goal and on subsequent trials propagates better h values backward—resulting in better paths.
 - > Because the action selected at node n is the one leading to a node estimated to be on a minimal-cost path from n to a goal it is not necessary to evaluate all the successors of n to update h .

University of Florida
 EEL 5840 - Class #27 - Fall 2009
 © Dr. A. Armitage Armitage

21




EEL5840: Elements of Machine Intelligence
Planning, Acting, and Learning

- Learning Heuristic Functions
 - > The technique can learn non-optimal paths because nodes on the optimal path may never be visited.
 - > To solve this problem allow occasional random actions (instead of those recommended by the policy) to hopefully help the agent learn new (and perhaps better) paths to goals.
 - > This is an instance of the tradeoff involved between exploration (of new paths) and exploitation (of already learned knowledge).
- Implicit Graphs
 - > When we have a model of the effects of actions (when we have operators that transform the description of a state into a description of a successor state) we can perform a search process guided by an evaluation function.

University of Florida
 EEL 5840 - Class #27 - Fall 2009
 © Dr. A. Armitage Armitage

22




EEL5840: Elements of Machine Intelligence
Planning, Acting, and Learning

- Learning Heuristic Functions
 - > Typically the function may produce the same evaluation for several nodes, so applying the function to state descriptions is feasible, but storing all the nodes is not realistic.
 - > Start by guessing a set of sub-functions that we think might be good components of a heuristic function
 - > In the 8-Puzzle, let $h(n) = w_1 W(n) + w_2 P(n) + w_3 S(n) + \dots$
 - > In method 1 we set the weights initially to a reasonable estimate and search using this h function. When we reach a goal, n_g we use the known value $h(n_g) = 0$ to back up h values for all nodes n_i along the path to the goal.
 - > Using these “training samples” we adjust the weights to minimize the sum of the squared errors between the training samples and the h function and iterate.

University of Florida
 EEL 5840 - Class #27 - Fall 2009
 © Dr. A. Armitage Armitage

23



EEL5840: Elements of Machine Intelligence
Planning, Acting, and Learning


- Learning Heuristic Functions
 - > In method 2 we adjust the h function with every node expansion

$$h(n_i) \leftarrow h(n_i) + \beta (\min [h(n_j) + c(n_i, n_j)] - h(n_i)) \quad \forall n_j \in S(n_i)$$

$$h(n_i) \leftarrow (1 - \beta) h(n_i) + \beta \min [h(n_j) + c(n_i, n_j)] \quad 0 < \beta \leq 1$$
 β is a learning rate parameter that controls how closely $h(n_i)$ is made to approach $\min [h(n_j) + c(n_i, n_j)]$. When $\beta = 0$ no change is made at all; when $\beta = 1$ we set $h(n_i)$ equal to $\min [h(n_j) + c(n_i, n_j)]$
 - > Small values of β lead to very slow learning
 - > Large values of β make learning erratic and non-convergent
 - > This method is an instance of temporal difference learning [Sutton 1988], where the weight adjustment depends only on two temporally adjacent values of a function.

University of Florida
 EEL 5840 - Class #27 - Fall 2009
 © Dr. A. Armitage Armitage

24




EEL5840: Elements of Machine Intelligence

Planning, Acting, and Learning

- Rewards Instead of Goals
 - > Suppose the task is an ongoing task where the user expresses his satisfaction with task performance by occasionally giving the agent positive or a negative rewards. The task for the agent becomes one of maximizing the amount of reward it receives.
 - > We seek to describe an action policy that maximizes reward.
 - > Since a future reward might be infinite, we proceed by discounting future rewards by some factor. This means that the agent prefers rewards in the immediate future to those in the distant future.
 - > We will let the agent take an action at every time step. Each action results in a change in the state description — a change actually perceived by the agent or computed by applying an operator to the model.

University of Florida
EEL 5840 - Class #27 - Fall 2009
© Dr. A. Armitage Armitage

25



EEL5840: Elements of Machine Intelligence

Planning, Acting, and Learning


- Rewards Instead of Goals
 - > Let n denote a node in the state-space graph for the agent. Let π be a policy function on nodes whose value is the action prescribed by that policy at that node. Let $r(n_i, a)$ be the reward received by the agent when it takes an action a at n_i . If this action results in n_j then ordinarily we would have

$$r(n_i, a) = -c(n_i, n_j) + \rho(n_j),$$
 where $\rho(n_j)$ is the value of any special reward given for reaching node n_j . Some policies lead to larger discounted future rewards than others.
 - > We seek an optimal policy π^* that maximizes future discounted rewards at every node.

This is the basis for methods called
Reinforcement Learning

University of Florida
EEL 5840 - Class #27 - Fall 2009
© Dr. A. Armitage Armitage

26



EEL5840: Elements of Machine Intelligence

Planning, Acting, and Learning


- Rewards Instead of Goals
 - > Given a policy π , we can assign a cost $V^\pi(n)$ for each node in the state space, i.e., $V^\pi(n)$ is the total discounted reward that the agent will receive if it begins at n and follows policy π .
 - > If we are at n_i and we take the action prescribed by $\pi(n_i)$ which results in reaching node n_j then

$$V^\pi(n_i) = r[n_i, \pi(n_i)] + \gamma V^\pi(n_j)$$
 where $0 < \gamma < 1$ is the discount factor that is used in computing the value at time t_i of a reward at time $t = t_{i+1}$ and $\pi(n_i)$ is the action prescribed by policy π at n_i .
 - > For the optimal policy π^* we would then have

$$V^{\pi^*}(n_i) = \max\{r[n_i, a] + \gamma V^{\pi^*}(n_j)\}$$
 The value of n_j under the optimal policy is the amount the agent receives by taking that action at n_i that maximizes the sum of the immediate reward plus the discounted value (by γ) value of n_j under the optimal policy.

University of Florida
EEL 5840 - Class #27 - Fall 2009
© Dr. A. Armitage Armitage

27



EEL5840: Elements of Machine Intelligence

Planning, Acting, and Learning


- Rewards Instead of Goals
 - > Since n_j is a *fcn* of the action a that also makes $V^\pi(n_j)$ a *fcn* of a
 - > If we knew the values of the nodes under the optimal policy we could write the optimal policy in the following way:

$$\pi^*(n_i) = \operatorname{argmax}\{r(n_i, a) + \gamma V^{\pi^*}(n_j)\}$$
 - > Since we do not know these values, the learning procedure called value iteration is used to estimate the values as follows:
 - Assign randomly an estimated value $V'(n)$ to every node n .
 - Suppose we are at node n_i and the estimate value of node n_i is $V'(n_i)$
 - Now we select action a that maximizes the sum of the immediate reward plus the estimated value of the successor node (assuming we have $\Gamma(n)$)
 - When we arrive at successor node n_j by taking action a we update $V'(n_j)$

$$V'(n_j) \leftarrow (1-\beta) V'(n_j) + \beta[r(n_i, a) + \gamma V'(n_j)]$$
 This adjustment moves the value of $V'(n_j)$ closer to $[r(n_i, a) + \gamma V'(n_j)]$
 If $V'(n_i)$ is a good estimate for $V^{\pi^*}(n_i)$ this makes $V'(n_j)$ close to $V^{\pi^*}(n_j)$

University of Florida
EEL 5840 - Class #27 - Fall 2009
© Dr. A. Armitage Armitage

28




EEL5840: Elements of Machine Intelligence

Planning, Acting, and Learning

- Rewards Instead of Goals
 - > For $0 < \beta < 1$ even if the actions have random effects and yield random rewards described by probability functions the procedure will converge if we visit the nodes infinitely often.
 - > The result is surprising because we are exploring the space using only an estimate of the optimal policy at the same time that we are trying to learn values of nodes under an optimal policy. This works without requiring the estimate be good.

University of Florida
EEL 5840 - Class #2 - Fall 2009
© Dr. A. Aravamudan

29



EEL5840: Elements of Machine Intelligence

The End!

University of Florida
EEL 5840 - Class #2 - Fall 2009
© Dr. A. Aravamudan

30