Lecture 24:Resource Bounded Reasoning

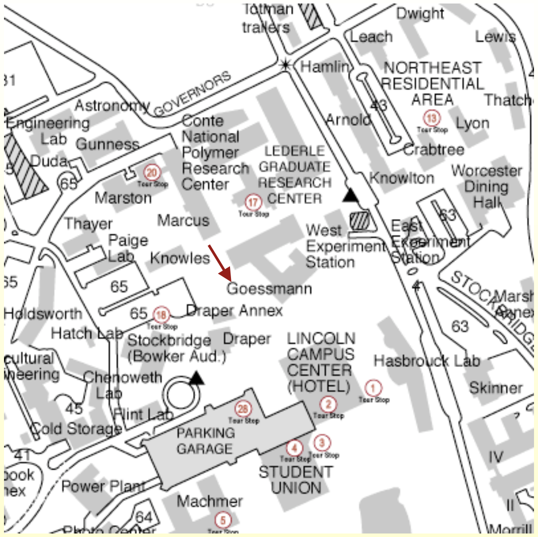
Victor Lesser

CMPSCI 683
Fall 2004

Exam

- **Time**
  - Friday 12/17 8-10am.
- **Location**
  - GSMN 51 Goessmann Lab
- **Open Book**
- **Only on Material not covered on Midterm**

Exam Location



Material for Exam

- **Rational Decision Making under Uncertainty**
  - Utility Theory
  - Value of Information
  - Decision Networks/Influence Diagrams
- **Learning**
  - Decision trees
  - Reinforcement learning
    - Dynamic programming
  - Neural networks
  - Instance-based learning
    - Case-based learning
  - Analytic learning
    - EBL
  - Relational learning (guest lecture)
- **Resource Bounded Reasoning**
- **Multi-Agent Systems**

Today's Lecture

- Resource Bounded Reasoning

Need for Resource-Bounded Reasoning

- Agents have limited computational power.
- They must react within an acceptable time.
- Computation time delays action and reduces the value of the result.
- Must cope with uncertainty and missing information.
- Limited planning horizon.
- The "appropriate" level of deliberation is situation dependent.

Agents cannot be perfectly rational

Building Resource-Bounded Reasoning Systems

A methodology for building satisficing systems by addressing the following four major issues:

1. Elementary algorithm construction
2. Performance measurement and prediction
3. Composability of methods (subsystems)
4. Monitoring and meta-level control

In the context of an Overall System Architecture

Elementary Algorithm Construction

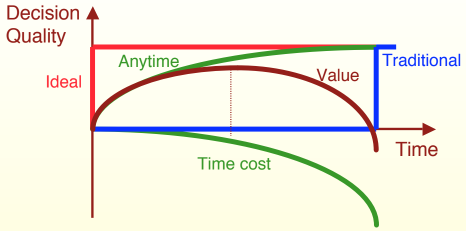
- Two Approaches
  - Anytime Methods
    - Increasing better result with time or other resources
    - Always have an answer available
  - Approximate Methods
    - Approximate solution in shorter time/ less resources than required by optimal solution

- Quality measures replace "correctness"
  - **Certainty** - Likelihood that the answer is correct.
  - **Precision** - Accuracy of the answer.
  - **Specificity** - Level of detail in the answer.
  - **Completeness** - Part of problem solved
  - **Cost** - Overall solution cost.
  - **Multidimensional quality measures.**

Anytime Algorithms

- An anytime algorithm is an algorithm whose quality of results improves gradually as computation time increases
  - computational methods that allow small quantities of resources - such as time, memory, or information - to be traded for gains in the value of computed results.
  - Interruptible algorithms are anytime algorithms whose run time need not be determined in advance
    - They can be interrupted at any time during execution and return a result
- Anytime algorithms have been designed for planning, Bayesian inference, CSPs, combinatorial optimization, diagnosis

Anytime Algorithms



- Ideal (maximal quality in no time)
- Traditional (quality maximizing)
- Anytime (utility maximizing)

- Performance profiles, $Q(t)$, return quality as a function of time

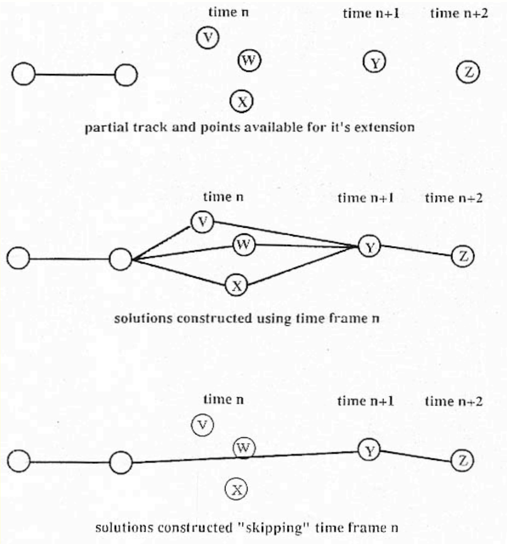
Approximate Methods

- Construct Approximate Methods that have
  - Less variance on their resource usage
  - Lower expected resource usage
- Different Forms of Approximation
  - Process Approximations
  - Knowledge Approximations
  - Data Approximations

Where do Process Approximations Come From?

- Complex problem solving as a multi-step process
  - Sequence of intermediate subgoals
- Sequence partially ordered
  - Not all steps are necessary
- Sequence repeated in multiple contexts/Search
  - Not all contexts need to be looked at
- Problem solving already assumes the solution to a subgoal may not be optimal
  - Adding alternative ways of solving subgoals doesn’t alter things too much

Process Approximation --Time Frame Skipping



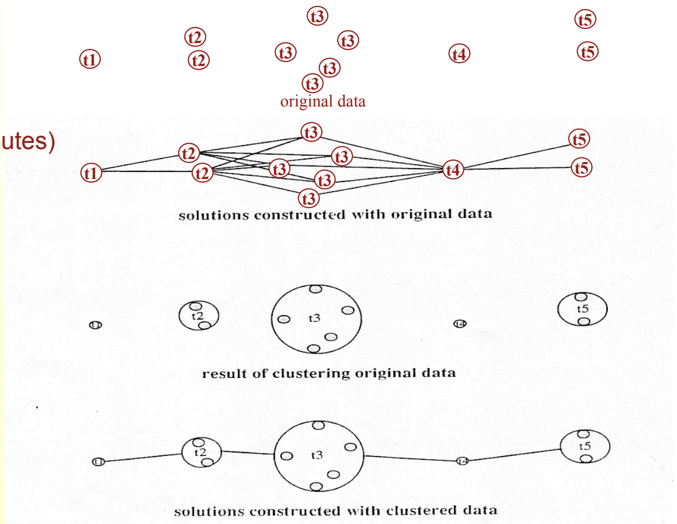
partial track and points available for it's extension

solutions constructed using time frame n

solutions constructed "skipping" time frame n

Data Approximation (Input)

- Incomplete processing (ignoring attributes)
- change in representation
- Clustering information



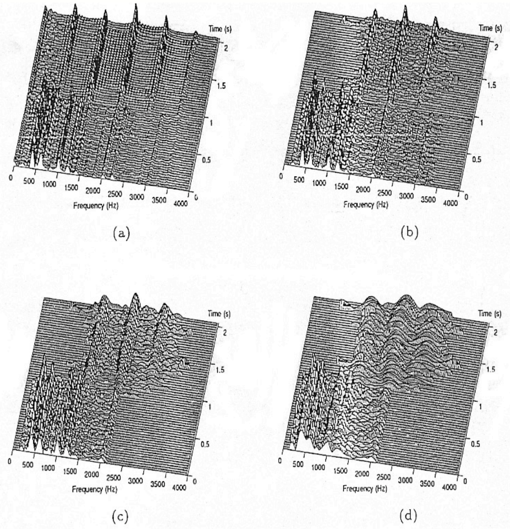
solutions constructed with original data

result of clustering original data

solutions constructed with clustered data

The Effects of Approximate Signal Processing

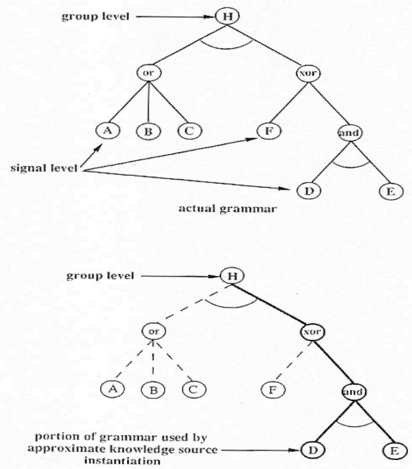
Figure 2.14: A comparison of the exact and the approximate STFTs corresponding to a violin playing a sequence of two notes. Approximate STFTs were calculated using the hybrid narrowing approach with minimum frequency coverage constraint set to 2000 Hz. The Plot in part (a) corresponds to the exact STFT. Plots in (b), (c) and (d) correspond to approximate STFTs with arithmetic complexity relative to that of a pruned FFT restricted to 50%, 25%, and 12.5%, respectively.



Source: Erkan Dorken, Ph.D., Thesis, Boston University

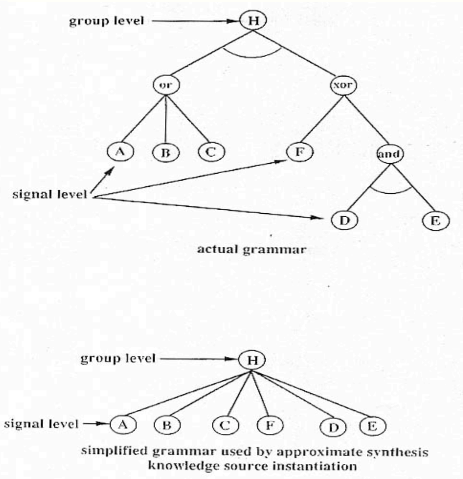
Knowledge Approximation

Partial Support



portion of grammar used by approximate knowledge source instantiation

Eliminate Detail



simplified grammar used by approximate synthesis knowledge source instantiation

Composition

- **Given:**
  - Alternative ways of solving the problem
    - composed of anytime algorithms or approximate methods for solving primitive subgoals
  - (Conditional) Performance Profiles of the primitive methods (components)
    - Quality of input to method leads to different performance profiles
  - A time-dependent/resource dependent utility function
- **Problem:**
  - For given a particular setting of the utility function, calculate the best way to solve the problem

Alternative Compositional Approaches

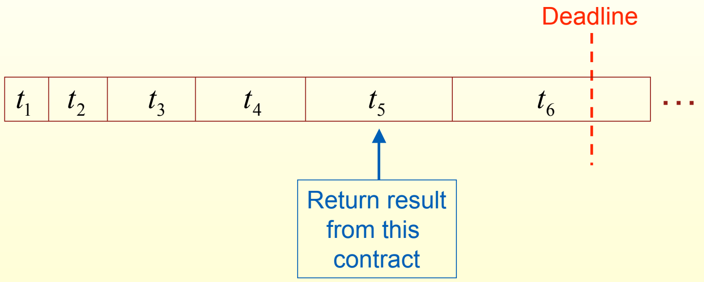
- **Contract Algorithms**
  - Build out of anytime algorithms
  - Allocate a fix amount of time to each anytime algorithm based on deadline
    - Based on performance profile
- **Design-to-Time**
  - Construct a sequence of approximate methods that will likely meet deadline restrictions
    - Involves elements of planning (deciding what to do) and scheduling (deciding when to perform particular actions).
    - Replan/re-adjust if partial sequence not making suitable progress

Contract → Interruptible

- **What if we want to use a contract algorithm in a setting where we don't know the deadline?**
- **We can repeatedly activate the contract algorithm with increasing run times**

Contract → Interruptible

- **When the deadline occurs, we can return the result produced by the last contract to finish:**

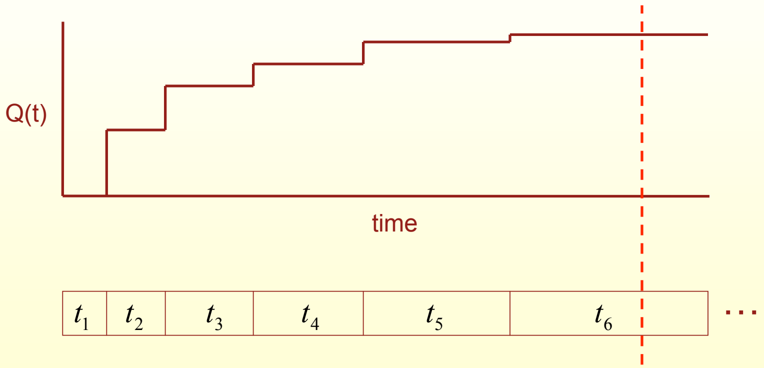


Deadline

t_1 t_2 t_3 t_4 t_5 t_6 . . .

Return result from this contract

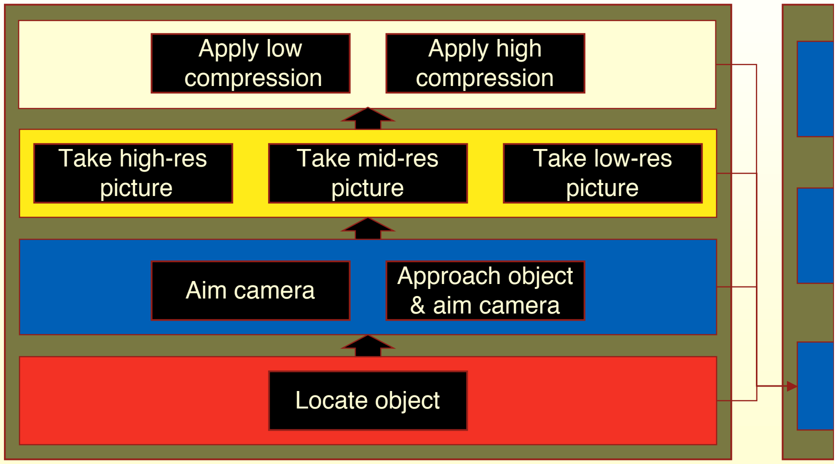
The Resulting Performance Profile



The Progressive Processing Model

- Progressive processing is an approach to performing a set of tasks under tight resource constraints and high-level of uncertainty.
- Each task is composed of a hierarchy of levels each of which offers a tradeoff between resource consumption and quality.
- Problem: (fine-grained scheduling) how to select modules for execution so as to maximize the overall expected utility?

A Sample Robotic Activity Represented as a Progressive Processing Unit



Formal Model

- A progressive processing unit is composed of a sequence of processing levels $(l_1 ... l_L)$
- Each level l_i is composed of a set of p_i alternative modules $\{m_1 \ldots m_{p_i}\}$
- Each module m_i has a module descriptor

$$P_i^j((q', \Delta r) \mid q)$$

delta r is the resource allocation

q is the quality of input to module

- A reward function, $U(q)$, specifies the immediate reward for performing the activity with an overall quality q.

The Reactive Control Problem

Problem: select a set of alternative modules so as to maximize the expected utility over a complete plan.

- Respond quickly to deviations from expected quality or resource consumption of a module.
- Respond quickly to plan modifications.
- Avoid a complex rescheduling process.

Optimal Control of a Single PRU by Mapping to an MDP

- State representation: $S = \{[l_i, q, r] \mid l_i \in u\}$
- Select the best action:

  E_{i+1}^{j} - execute j - th module of the next level
- Rewards and the value function:

$$\Pr([l_{i+1}, q', r - \Delta r] \mid [l_i, q, r], \mathrm{E}_{i+1}^{j}) = P_{i+1}^{j}((q', \Delta r) \mid q)$$

$$V([l_L, q, r]) = U(q)$$

$$V([l_i, q, r]) = \max_{j} \sum_{q', \Delta r} P_{i+1}^{j}((q', \Delta r) \mid q) V([l_{i+1}, q', r - \Delta r])$$

Optimal Control

Theorem: Given a progressive processing unit u, an initial resource allocation r_0 and a reward function $U(q)$, the optimal policy for the corresponding MDP provides an optimal strategy to control u.

Proof: Based on the one-to-one correspondence and the fact that the PRU transition model satisfies the Markov assumption.

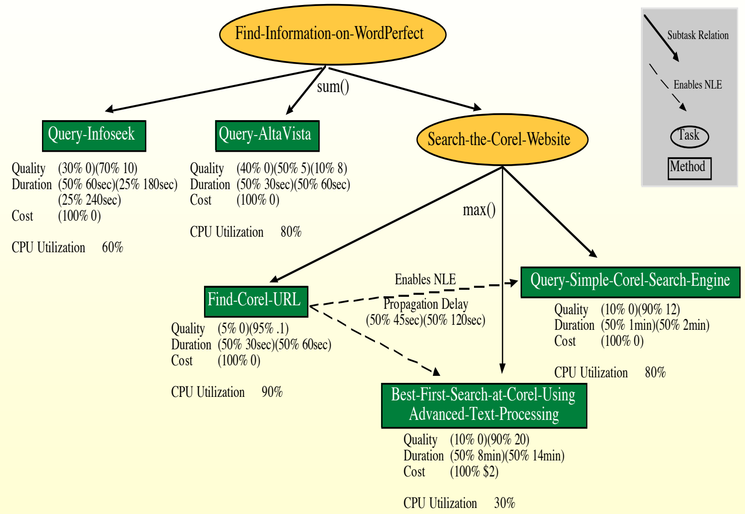
Scheduling Sequence of PRUs

- Can extend the state space to be $[i,l,q,r]$ and apply the same approach to construct a globally optimal policy.
  - *i* is the current PRU in the sequence
- But, hard to reconstruct a global policy on-board or transmit it to the rover.
- How could the remaining plan be factored into the control process? And how to avoid revising the entire policy when the plan is modified?

Example of Design-to-Time Information Gathering Agent

- **Objective:** gather information to support decisions
- **Application:** software evaluation
- **Example:** “Within 20 minutes, help me choose a 3D rendering package that runs under Windows 95 on my current hardware setup, and find a vendor who’ll sell it to me for under $400. Mac compatibility is a bonus.”
- **Results:** recommendation, knowledge gained during search, and source documents or URLs for source documents

Information Gathering Plan Network



Find-Information-on-WordPerfect

sum()

Query-Infoseek
Quality (30% 0)(70% 10)
Duration (50% 60sec)(25% 180sec) (25% 240sec)
Cost (100% 0)
CPU Utilization 60%

Query-AltaVista
Quality (40% 0)(50% 5)(10% 8)
Duration (50% 30sec)(50% 60sec)
Cost (100% 0)
CPU Utilization 80%

Search-the-Corel-Website

max()

Find-Corel-URL
Quality (5% 0)(95% .1)
Duration (50% 30sec)(50% 60sec)
Cost (100% 0)
CPU Utilization 90%

Enables NLE

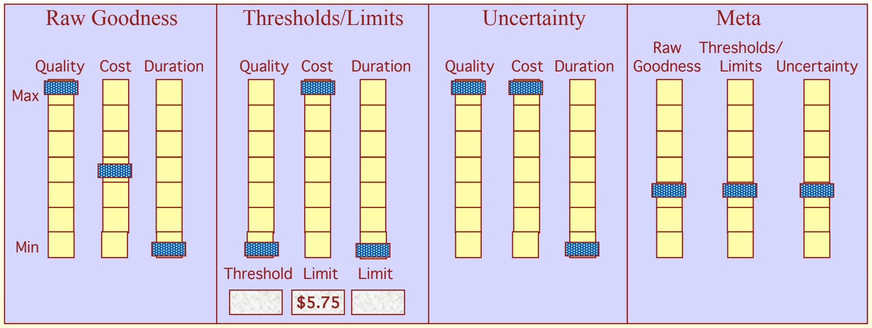
Propagation Delay (50% 45sec)(50% 120sec)

Query-Simple-Corel-Search-Engine
Quality (10% 0)(90% 12)
Duration (50% 1min)(50% 2min)
Cost (100% 0)
CPU Utilization 80%

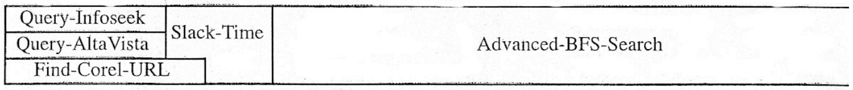
Best-First-Search-at-Corel-Using Advanced-Text-Processing
Quality (10% 0)(90% 20)
Duration (50% 8min)(50% 14min)
Cost (100% $2)
CPU Utilization 30%

Subtask Relation
Enables NLE
Task
Method

Utility Function



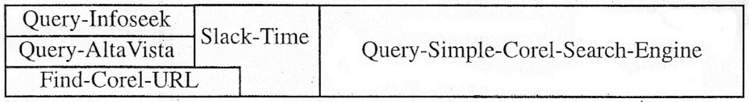
Conceptual Criteria: Quality Importance High, Cost Importance NIL, Duration Importance NIL



Query-Infoseek
Query-AltaVista
Find-Corel-URL
Slack-Time
Advanced-BFS-Search

Schedule Quality: (15% 0)(50% 30)(30% 35)(6% 38), Expected Value = 27
Schedule Cost: (5% 0)(95% 2.0), Expected Value = $1.90
Schedule Duration: (5% 154sec)(47% 844sec)(24% 994sec)(24% 1204sec), Expected Value = 871 seconds

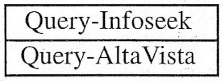
Conceptual Criteria: Quality Importance High, Cost Importance High, Duration Importance Low



Query-Infoseek
Query-AltaVista
Find-Corel-URL
Slack-Time
Query-Simple-Corel-Search-Engine

Schedule Quality: (13% 0)(51% 22)(30% 27)(6% 30), Expected Value = 21
Schedule Cost: (100% 0), Expected Value = $0.00
Schedule Duration: (2% 154sec)(18% 214sec)(44% 274sec)(36% 484sec), Expected Value = 329 seconds

Conceptual Criteria: Quality Importance Low, Cost Importance High, Duration Importance High



Query-Infoseek
Query-AltaVista

Schedule Quality: (20% 0)(39% 10)(35% 15)(7% 18), Expected Value = 10
Schedule Cost: (100% 0), Expected Value = $0.00
Schedule Duration: (50% 60sec)(25% 180sec)(25% 240sec), Expected Value = 135 seconds

Principles of Meta-Level Control

- What’s are the base-level computational methods?
- How does the system represent and project resource/quality tradeoffs?
- What kind of meta-level control is used?
- What are the characteristics of the overall system?

Example of Meta-Level Control Problem

- Problem: How to decide when to stop the execution of an anytime algorithm?

Needed due to the uncertainty regarding:

- The actual quality of the results
- The actual state of the environment

Best monitoring technique depends on:

- Degree of domain uncertainty
- Degree of solution quality uncertainty
- The cost of monitoring
- Interruptible versus contract algorithms

Myopic Control of Interruptible Algorithms

- Approach: Given an interruptible anytime algorithm, its Conditional Performance Profile (CPP), and a time-dependent utility function, run the algorithm as long as the marginal value of computation is positive.
- Theorem [Zilberstein, 1993]: Monitoring using the value of computation is optimal when the CPP is monotonically increasing and concave down, and the cost of time is monotonically increasing and concave up.

Monitoring Policies

- Approach: Given $\Pr(q_j|q_i,\Delta t)$ and $U(q_j,t_k)$ compute $\pi(q_j,t_k)\rightarrow(\Delta t,m)$ by optimizing the following function:

$$V(q_i, t_k) = argmax_{\Delta t,m}\{$$
$$\Sigma_j \Pr(q_j \,|\, q_i, \Delta t)\; U(q_j, t_k+\Delta t) \qquad \text{if } m = \text{stop},$$
$$\Sigma_j \Pr(q_j \,|\, q_i, \Delta t)\; V(q_j, t_k+\Delta t) - C \qquad \text{if } m\text{=monitor}\}$$

- Theorem [Hansen & Zilberstein, 1996]: A monitoring policy that maximizes the above value function is optimal when quality improvement is Markovian.

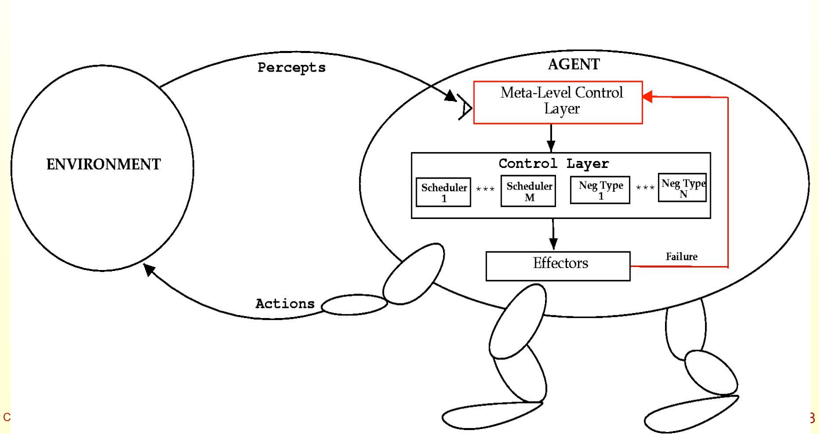
Another Meta-Level Control (MLC) Problem?

- **Dynamically Balance domain and control actions**
  - Domain - primitive actions
  - Control - coordination, scheduling, information gathering
- **Chooses control actions**
  - based on current state and control effort
  - likely to lead to good performance
  - tailored to time pressure and resource bounds
- **Low real-time cost**

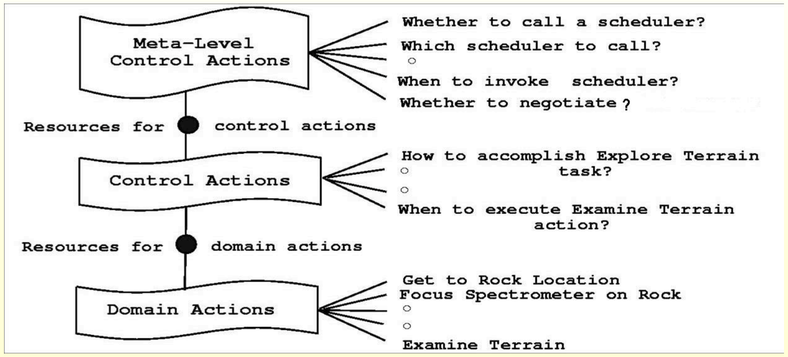
Key for Agent Operating in Open and Evolving Environments

- Uncertainty in Tasks Arrival, Deadlines and Behavior
- Resource Availability -- including other agents

Agent Architecture with Meta-Level Reasoning



Agent's Action Hierarchy



Meta-Level Control Actions

Whether to call a scheduler?
Which scheduler to call?
○
When to invoke scheduler?
Whether to negotiate ?

Resources for ● control actions

Control Actions

How to accomplish Explore Terrain task?
○
○
When to execute Examine Terrain action?

Resources for ● domain actions

Domain Actions

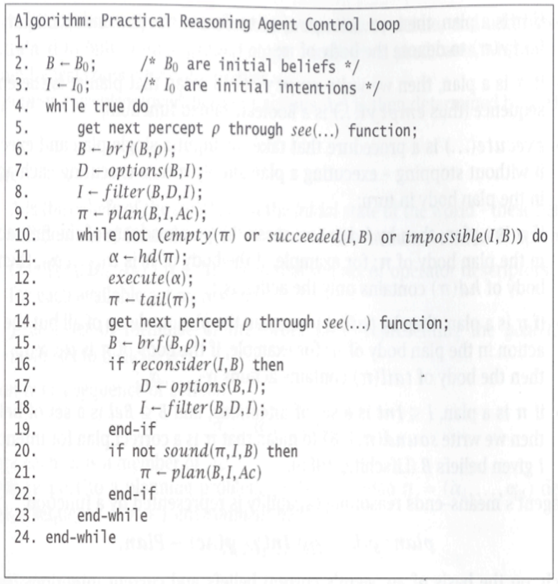
Get to Rock Location
Focus Spectrometer on Rock
○
○
Examine Terrain

Building a Resource Bounded Agent Architecture

BDI Architecture

Belief Desire Intention

from
M. Wooldridge's *An Introduction to MultiAgent Systems*. Copyright 2002. John Wiley & Sons, Ltd.



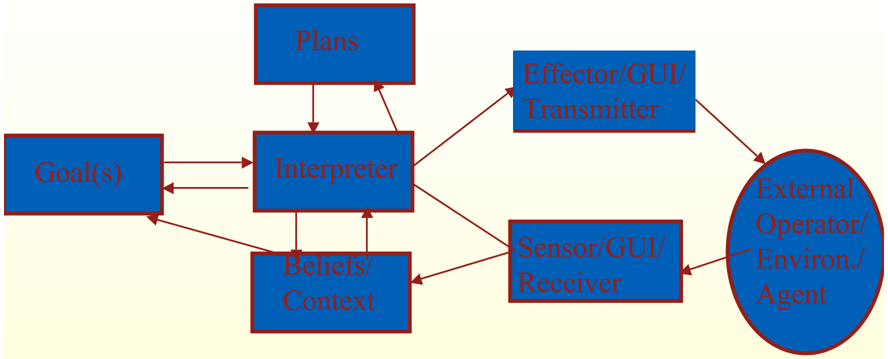
```
Algorithm: Practical Reasoning Agent Control Loop

$B \leftarrow B_0$;        /* $B_0$ are initial beliefs */
$I \leftarrow I_0$;        /* $I_0$ are initial intentions */
while true do
    get next percept $\rho$ through $see(\ldots)$ function;
    $B \leftarrow brf(B,\rho)$;
    $D \leftarrow options(B,I)$;
    $I \leftarrow filter(B,D,I)$;
    $\pi \leftarrow plan(B,I,Ac)$;
    while not ($empty(\pi)$ or $succeeded(I,B)$ or $impossible(I,B)$) do
        $\alpha \leftarrow hd(\pi)$;
        $execute(\alpha)$;
        $\pi \leftarrow tail(\pi)$;
        get next percept $\rho$ through $see(\ldots)$ function;
        $B \leftarrow brf(B,\rho)$;
        if $reconsider(I,B)$ then
            $D \leftarrow options(B,I)$;
            $I \leftarrow filter(B,D,I)$;
        end-if
        if not $sound(\pi,I,B)$ then
            $\pi \leftarrow plan(B,I,Ac)$
        end-if
    end-while
end-while
```

Figure 4.3 A practical reasoning agent.

Why not appropriate for Real-time?

PRS Agent Architecture



Interpreter pursues goals by retrieving and executing plans that satisfy the context, leading to actions and the acquisition of new context and beliefs, in turn creating new goals.

Next Lecture

- **Introduction to Multi-Agent Systems**
- **Short Summary of Course**