ICAPS 2003 Tutorial

Resource-bounded and Time-critical Reasoning

Lloyd Greenwald, Drexel University

Shlomo Zilberstein, University of Massachusetts

Outline

- Introduction to meta-level control of computation
- Algorithm design, performance modeling and prediction
- Algorithm composition and transformations
- Meta-reasoning and deliberation scheduling
- Sample applications
- Current research directions and wrap-up

Introduction To Meta-level Control Of Computation

The Action Selection Problem

- Agents have limited computational resources.
- They must react to a situation within an acceptable amount of time.
- Key question: How should agents select actions when there is not enough time or memory or information to compute the best one?

Simon's "Bounded Rationality"

"A theory of rationality that does not give an account of problem solving in the face of complexity is sadly incomplete. It is worse than incomplete; it can be seriously misleading by providing "solutions" that are without operational significance"

"The global optimization problem is to find the least-cost or best-return decision, net of computational costs."

-- Herbert Simon, 1958

Good's "Type II Rationality"

"When the expected time and effort taken to think and do calculations is allowed for in the costs, then one is using the principle of rationality of type II."

-- Irving Good, 1971

The Principle Of Satisficing

- Satisficing, a Scottish word that means satisfying, was proposed by Herbert Simon in 1957 to denote decision making that searches until an alternative is found that meets the agent's aspiration level criterion.
- Aspiration level is borrowed from psychology, where it denotes a dynamic, context-dependent criterion typically acquired by experience

Satisficing In Nature

"It appears probable that, however adaptive the behavior of organisms in learning and choice situations, this adaptiveness falls far short of the ideal 'maximizing' postulated in economic theory. Evidently, organisms adapt well enough to 'satisfice'; they do not, in general, 'optimize'."

- Is satisficing more than a vague principle?
- How can it be formalized?

Satisficing In CS And AI

- Problem complexity makes it impossible to reach the provably best decision (e.g. chess, planning under uncertainty, multi-agent coordination, VLSI layout, etc.)
- In some cases, computing the best answer is feasible, but it is not economical (e.g. mobile robot navigation, scheduling, 3D rendering, etc.)

Theories Of Bounded Rationality

"Prospects appear poor for finding a single 'right' theory of bounded rationality due to the many different ways of weakening the ideal requirements, some formal impossibility and tradeoff theorems, and the rich variety of psychological types observable in people, each with different strengths and limitations in reasoning abilities."

-- Jon Doyle, 1997

Achieving Satisficing

Satisficing can be achieved by:

- design - the designer of a system determines the aspiration level.
- run-time deliberation - the agent determines the aspiration level.
- adaptation - the agent learns the aspiration level.

Satisficing vs. Computational Savings

Satisficing implies that there is a static or dynamic solution evaluation criterion.

Unlike: Limiting computational resources:

- Depth-limited search, RTA*

Unlike: Minimizing computational effort while seeking the best answer:

- Admissible heuristic search (A*, AO*, LAO*, IDA*, SMA*)
- Proving dominance of an action (B*, Protos)
- Pruning (alpha-beta search, bidirectional search)

Formalizing The Notion Of Satisficing

Simon's definition is vague and therefore hard to use as a design principle.

- The aspiration level tells us nothing about the problem solving technique.
- What is a "good enough" solution?
- How can a computer measure that?
- Should a satisfactory solution be reached directly or by iterative refinement?
- How to evaluate a satisficing agent?

Approaches To Satisficing

i. **Satisficing** = approximate reasoning

ii. **Satisficing** = approximate modeling

iii. **Satisficing** = optimal meta-reasoning

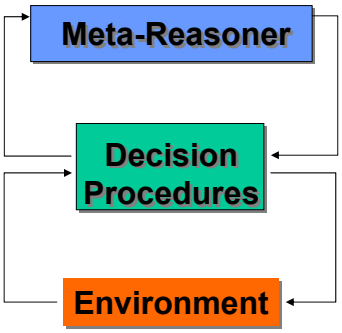
iv. **Satisficing** = bounded optimality

v. **Satisficing** = a combination of the above

Controlling Inference (Meta-reasoning)

- A step toward formalizing the notion of satisficing
- Traditional algorithms not responsive
  - Need to take resource and time pressures into account
- Reactive algorithms too short-sighted
- Need way to control level of computation

Meta-reasoner



Need to introduce a higher level meta-reasoner that controls lower level computation of actions – using decision theory, i.e. higher level is perfectly rational

"do the right thinking"

Applying Decision Theory At Meta-level

- Rational agent will choose action that maximizes expected utility (MEU), given
  - Uncertainty about outcomes
    - Example: Robot navigation
      - Future events
      - Sensor limitations
      - Incomplete knowledge of the effects of actions
      - Incomplete understanding of low-level computations
  - Some measure of preference in outcomes

Decision Theory

- Rational agent will choose action that maximizes expected utility (MEU), given
  - Uncertainty about outcomes
  - Some measure of preference in outcomes
- Utility theory deals with representing and reasoning about *preferences*
- Probability Theory + Utility Theory = Decision theory
- Weigh each outcome (state) by probability that it occurs $\max_a \sum_s P(s \mid a)U(s)$

Utility Function U(s)

- Captures agent's preference between world states
- Captured with a single real number for each state called *Utility*
- Utility function maps states to Utility
- Sometimes called a *Value Function*
- *Expected Utility:* utility weighted by probability $\sum_s P(s \mid a)U(s)$

Applying Decision Theory At Meta-level

- Issue: need to understand interaction of *physical* and *inferential actions*
  - Physical action: directly effects outcome by changing world
  - Inferential action: computation that indirectly affects outcome
- May take sequence of inferential actions before finally taking a physical action

Decision Procedure

- Inferential actions used to select physical action
- May build up internal state in agent
- Inferential actions use resources (e.g. time, space, battery power)
- Resource use may affect utility of outcome

- Meta-reasoning requires an understanding of decision procedures

First Step: Modeling Decision Procedures

- Discrete Chunks
  - Inference steps, methods
  - Combinatorial problems [Etzioni 89] [Russell and Wefald 91]
- Continuous Profiles
  - Flexible computations [Horvitz 87]
  - Anytime algorithms [Dean and Boddy 88]
  - More flexibility in meta-reasoning
- Combination
  - Profile with discrete time steps

Second Step: Solving Meta-level Control Problems

- Composing a sequence of decision procedures – *composition*
- Stopping of single decision procedure
- Resource allocation over a set of alternative decision procedures – *portfolio*
- Resource allocation across a sequence of required decision procedures with deadlines – *schedule*
- Conditional scheduling of a sequence of decision procedures – *policy*

Decision Procedures: Discrete Chunks I

- Model I [Russell and Wefald 91]
  - Default physical action (current best): $\alpha \in \mathcal{A}$
  - Set of potential computations (inferential actions): $\{S_i\}$
  - Utility of world due to action, U([α]), or due to computation, U([S_i]), taking delay and other resource usage into account
  - Change in value due to computation:
    V(S_i) = U([S_i]) - U([α])
- Meta-control choice:
  - take physical action α now, or
  - select next computation from $\{S_i\}$

Meta-level Control Approach

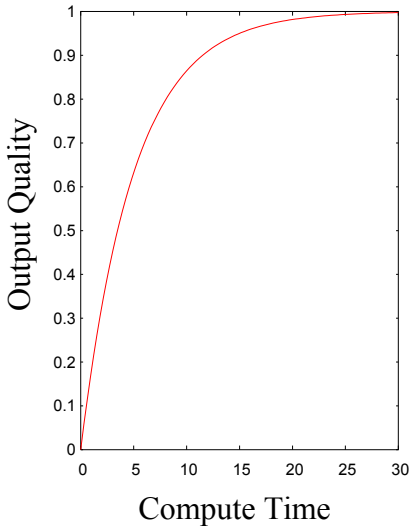
- Choose action (inferential or default) with largest value
- Assumes physical action will occur after *one* inferential step
  - Utility of inference directly tied to next action
  - In reality, a sequence of inferential steps might occur
  - Called *meta-greedy, single-step* assumptions
  - Pearl calls this *one-step horizon*
- Also assumes inferential action can only affect a single specific physical action
  - Called *subtree-independence*
- *Myopic* meta-control

Decision Procedures: Discrete Chunks II

- Model II [Etzioni 89], given:
  - Set of goals
  - Set of methods
  - Deadline
- Meta-control choice:
  - Sequence of methods, where
  - Method $m_{i,j}$ is i^{th} method applied to solve j^{th} goal
  - Once j^{th} goal is solved, agent moves on to next goal
  - $\sigma = m_{1,1},m_{2,1},\ldots,m_{k1,1},m_{1,2},m_{2,2},\ldots,m_{k2,2},\ldots,m_{1,n},m_{2,n},\ldots,m_{kn,n}$
- Proves that this meta-control problem is NP-complete

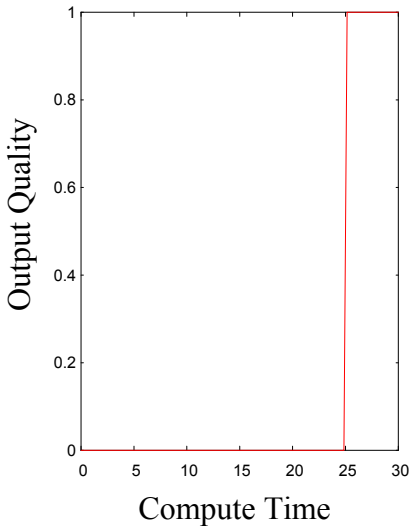
Decision Procedures: Anytime Algorithm

- Iterative improvement algorithm
- Output of computation
  - Quality of computation varies continuously with compute time
  - Quality monotonically non-decreasing
  - Generally displays diminishing returns in limit
- Can be stopped at "any time" (and resumed with minimal cost)
- Term coined by [Dean and Boddy 1988]
- Similar concept termed "Flexible Computations" [Horvitz 1987]



Traditional Algorithm (aka Run-to-Completion Algorithm)

- Algorithm: Well-defined sequence of computational steps that transform one set of values (input) to another set of values (output)
- Output of computation
  - If algorithm runs long enough → optimal result
  - If algorithm doesn't run long enough → no result



Why Anytime Algorithms?

- Useful when computation time reduces overall utility of result
- And level of computation needed changes with situation
- This "any time" property creates on-line meta-level control opportunities
  - Cannot always choose optimal run-time at system design time
  - Can monitor progress and allocate resources dynamically
  - Quality needs to be measurable at run-time

Outline

- Introduction to meta-level control of computation
- Algorithm design, performance modeling and prediction
- Algorithm composition and transformations
- Meta-reasoning and deliberation scheduling
- Sample applications
- Current research directions and wrap-up

Algorithm Design, Performance Modeling And Prediction

Types of Anytime Algorithms

- Numerical approximation procedures
  - e.g. Taylor's series approximations
- Heuristic search
- Dynamic programming
  - Value iteration
  - Policy iteration
- Monte Carlo simulation
  - e.g. for computing posterior probabilities
- Iterative improvement in general
  - e.g. sorting an array, insertion sort, bubble sort

Measures of Quality

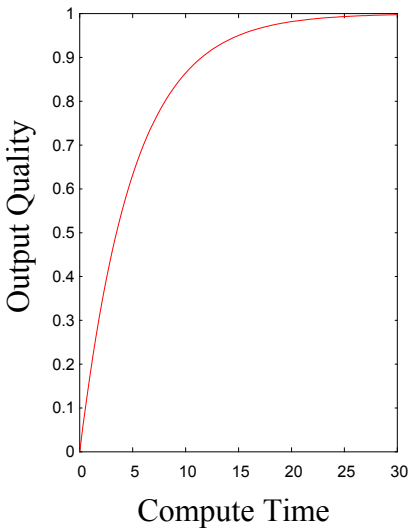
- Normalize to [0,1] metric for convenience
- Measures some aspect of the algorithm's output
  1. Certainty – degree of certainty that the output is correct
     - Example: certainty that diagnosis is correct
  2. Accuracy – approximation, bound on difference from exact solution
     - Example: Taylor series approximation of function
  3. Specificity – level of detail
     - Example: diagnose type of injury (penetrating) but not mechanism (sharp or blunt instrument)
- Measures are somewhat fuzzy and could be multidimensional

(adapted from Zilberstein and Russell 96)

Performance Profile

- Model of expected quality of computation output as function of computing time
- Coined by [Dean and Boddy 88]
- May be captured by
  - Table
  - Continuous function
  - Discontinuous function
  - Parameterized representation

Performance profile is needed for meta-control

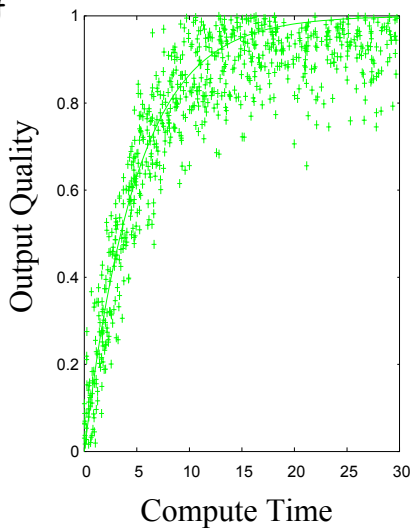


Building A Performance Profile

- Model of **expected** quality of computation output as function of computing time
- Quality not known precisely
  - Varies with
    - Input
    - Internal randomness
- Profile built from
  - Prior knowledge
  - Algorithm analysis
  - Statistics, simulation

$$Q(t) = \sum_{q} \Pr(q \mid t) q$$

  - Model fitting
    - Example: $Q(t) = 1 - e^{-\alpha t}$



A Note On Closed-form Models

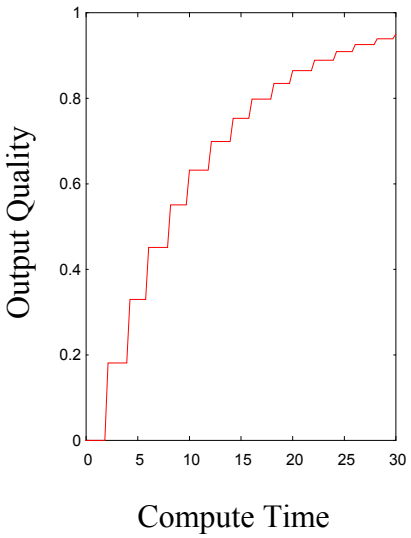
- Generally approximations to true expected value
- Curve fitting
- Benefits:
  - Can perform standard numerical optimizations on closed form parameterized families
  - Generalize to new problems
- Costs:
  - Error in approximations
  - No closure under composition (composed profile is in same curve family as component profiles)

Types Of Performance Profiles

- Expected performance profile $Q_{out}(t)$
- Conditional Performance profile $Q_{out}(Q_{in},t)$
- Bounds on output quality $Q_{min}(t), Q_{max}(t)$
- Probabilistic performance profile $Pr(Q_{out}|Q_{in},t)$
- Dynamic performance profiles $Pr(Q_{t+1}|Q_t,t)$
- Quality trajectories

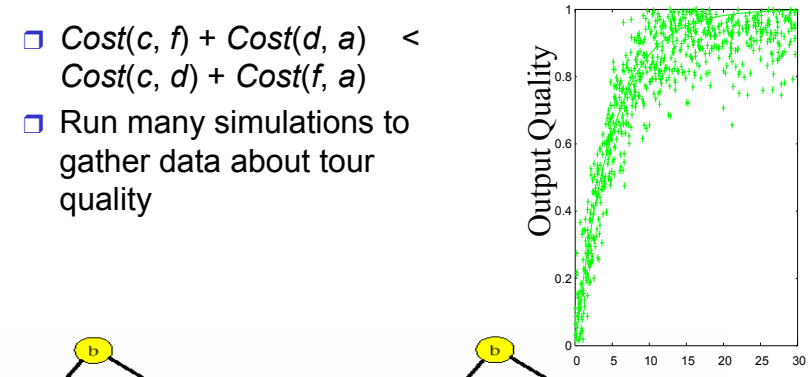
More On Profiles

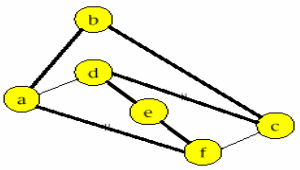
- Algorithms often progress in discrete steps, rather than continuous improvements
- Profiles are sensitive to problem parameters
  - Size of problem
  - Types of input
  - Existence of pre-cached solutions or partial solutions
  - Etc
- Options:
  - Different profiles to capture "reasonably similar" sets of problems – classify problems prior to meta-reasoning
  - Capture additional dimensions in profile, e.g. problem size, input type



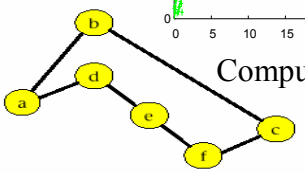
Example Anytime Algorithm: TSP Tour Improvement

- *Cost*(*c*, *f*) + *Cost*(*d*, *a*) < *Cost*(*c*, *d*) + *Cost*(*f*, *a*)
- Run many simulations to gather data about tour quality





(a)



(b)

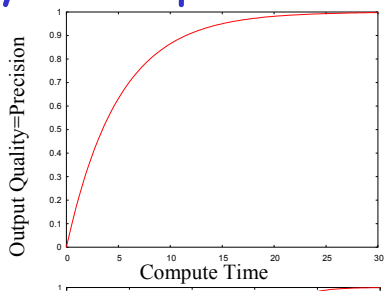
(adapted from Zilberstein 95)

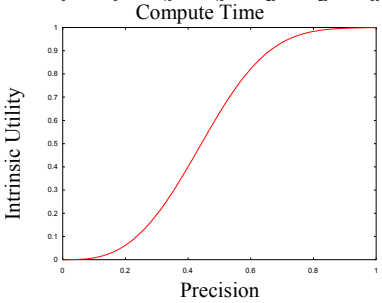
Utility vs. Quality

- Quality of computation must ultimately be tied to physical action selected by computation
- Utility of physical action has two components
  - "goodness" of action itself -- also called *intrinsic utility* or *object-related value*
  - Utility of world after taking physical action, taking into account resources consumed by inferential actions (computation) – also called *comprehensive value*

Utility vs. Quality Example

- Increased quality of computation maps directly to higher precision physical action
- Intrinsic utility of physical action varies non-linearly with precision
  - i.e. need a certain "threshold" of precision before action is any use, after that the utility increases less rapidly
  - Soft step function
- Assume quality = intrinsic utility unless stated otherwise
- Also, need to understand resource consumption costs

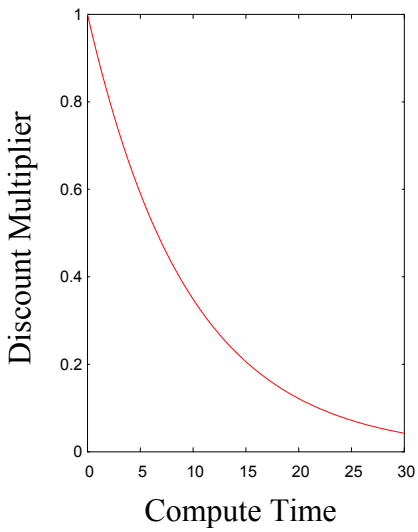




(adapted from Horvitz 87)

Time Cost

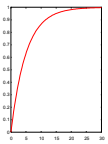
- Value of decision procedure is function of:
  - Physical action utility
  - Resource cost
- Example resource cost: discounting for time cost
  - γ^t
- Deadlines are also important – more later

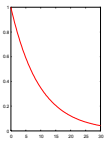


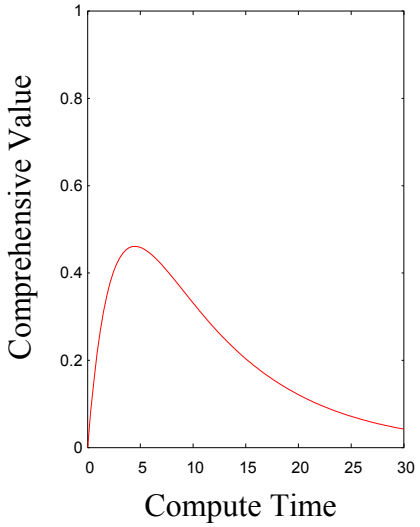
Comprehensive Value (Utility)

- Value of decision procedure is function of:
  - Physical action utility
  - Resource cost
- Generally assumed *separable*
- Example: multiply intrinsic utility by discount factor

  $(1 - e^{(-\alpha t)}) * \gamma^t$



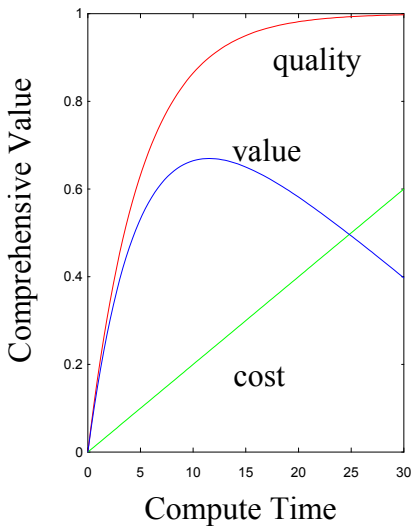




Comprehensive Value Example I

- Additive cost also common

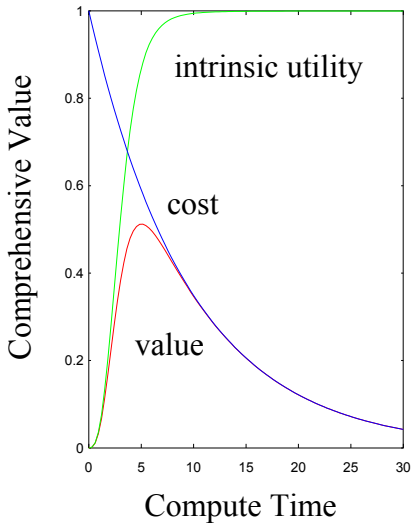
  $(1 - e^{(-\alpha t)}) - \gamma t$
- Also called *time-dependent utility*



$\left(1 - e^{-.2t}\right) - (.02t)$

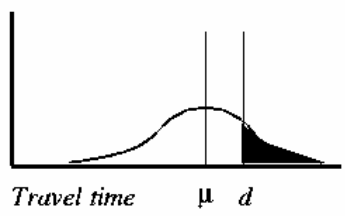
Comprehensive Value Example II

- Take quality
  $Q(t) = 1 - e^{(-\alpha t)}$
- Transform with non-linear utility
  $1 - e^{(-\beta Q(t))^{\chi}}$
- Multiply by discount factor
  γ^{t}



Note On Expected Quality

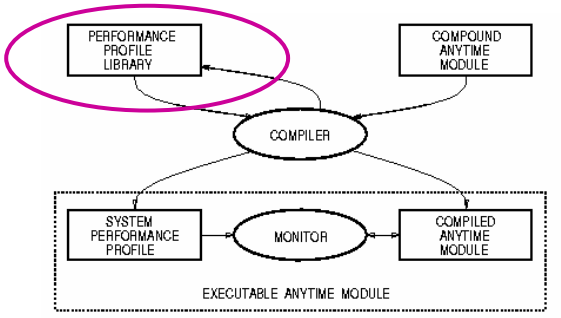
- It is sometimes necessary to look beyond expectations – at the entire distributions
- Variance can make a difference
  - same expected length but different probabilities of completing travel before hard deadline -> different utilities depending on cost of being late
- If variance is not narrow then expected profile is not good enough



(adapted from Boddy and Dean 94)

Anytime Algorithm Development

- Goal: Build large systems made up of anytime algorithms
- First step: libraries of anytime algorithms
  - Use as modules for multiple systems
  - Each algorithm has a performance profile
  - Need common way to represent profiles
  - Need common way to call algorithms



(adapted from Zilberstein and Russell 96)

Anytime Algorithm Development Cycle

- Create iterative improvement algorithm
- Turn into anytime algorithm
  - Permit varying time allocations
  - Make interruptible
- Create performance profile
  - Vary input quality
  - Vary time allocation
  - Record and summarize output quality (expected or full distribution)
- Compose with other anytime algorithms into system

Basic Improvement Step

- Find basic step
  - Must be fairly small – for algorithm to be interruptible
  - Examples: process one element in array, one node in search tree
- Design data structure to represent state after basic step
  - Generate solution from this data structure if interrupted
  - No interruptions during basic step – interruptions return solution from previously completed step

Profile Generation

- Generate sample representative problems
- Run algorithm on sample problems
  - Incrementally giving small amounts of computation and recording quality changes
  - Vary problem inputs if appropriate
- Generate statistics
  - Initial samples used to determine size and granularity of data storage

Measuring Quality

- Need way to measure quality of intermediate solutions
  - Used off-line in building profile
  - Also used on-line if monitoring employed
  - Ideally it is quick and does not require look-ahead to end of run
  - Example: sorting – percent of elements less then left neighbor

Quality Data (Map) For Anytime QuickSort Algorithm

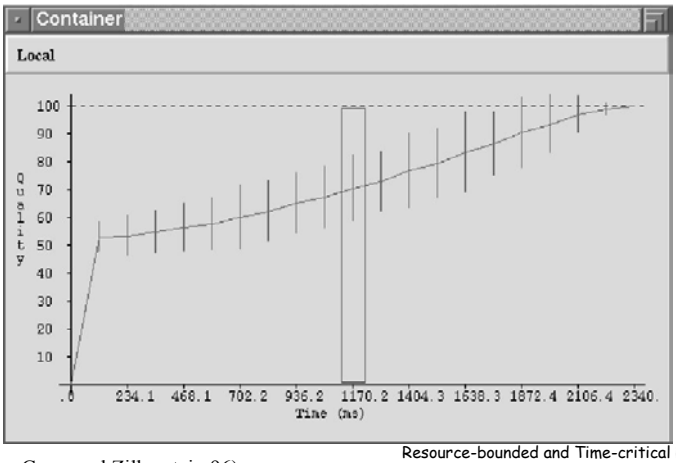
Appl Local File Value Restart History

```
CL-USER 162 > (write-data any-sort 200 50)
Time:       0  117  234  351  468  585  702  819  936 1053 1170 1287 1404 1521 1638 1755 1872 1989 2106 2223 2340
Counts:     0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    2   27   67  139  150   83
            0    0    0    0    0    0    0    0    0    0    0    0    0    0   14   34   80  100   43    2    0
            0    0    0    0    0    0    0    0    0    0    0    0    6   18   51  102   85   21    0    0    0
            0    0    0    0    0    0    0    0    0    0    2    4   33   68   98   54    5    0    0    0    0
            0    0    0    0    0    0    0    0    1    4   17   50   93   92   30    1    0    0    0    0    0
            0    0    0    0    0    0    1    3   21   32   97  109   52   16    0    0    0    0    0    0    0
            0    0    0    0    1    1   18   36   80  108   74   32    1    0    0    0    0    0    0    0    0
            0    0    1   14   21   39   84   98   79   32    8    0    0    0    0    0    0    0    0    0    0
           20   25   51   72   97   98   88   38    9    2    0    0    0    0    0    0    0    0    0    0    0
          151   65   93   98   41   22    9    0    0    0    0    0    0    0    0    0    0    0    0    0    0
           29    7   11    3    1    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
            0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
            0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
            0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
            0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
            0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
            0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
            0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
            0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
          200    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0

Total:    200   97  156  187  161  161  200  175  190  178  198  195  185  194  194  193  197  188  182  152   83
Average:   52   53   53   55   56   58   60   62   65   67   70   73   77   79   83   86   90   93   97   99  100
NIL

CL-USER 163 >
```

(adapted from Grass and Zilberstein 96)

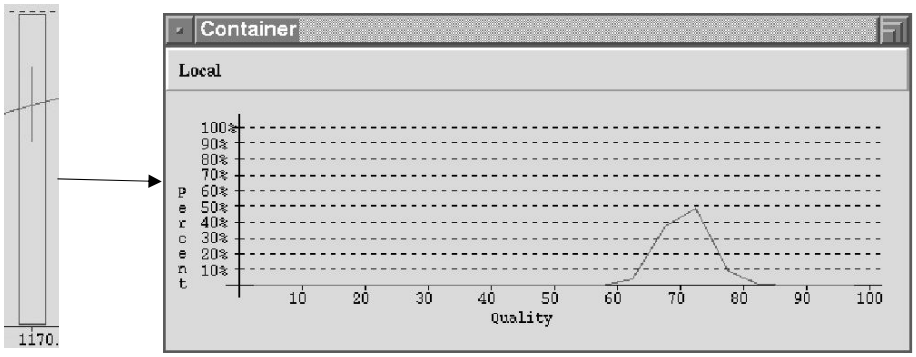
Performance Profile

Expected quality derived from sample data



(adapted from Grass and Zilberstein 96)

Variance In Expected Quality



- Distribution of qualities when run for 1170ms

(adapted from Grass and Zilberstein 96)

Outline

- Introduction to meta-level control of computation
- Algorithm design, performance modeling and prediction
- Algorithm composition and transformations
- Meta-reasoning and deliberation scheduling
- Sample applications
- Current research directions and wrap-up

Algorithm Composition And Transformations

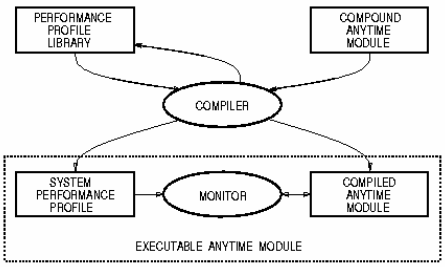
Toward Building Time-Critical Systems

- Build systems: trade quality of results against cost of computation
- Issues: construction, composition, and control of such systems

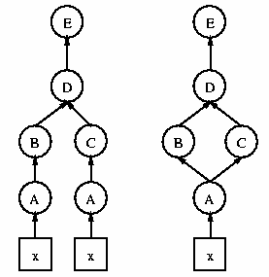
Recall: Anytime Algorithm Development Cycle

1. Create iterative improvement algorithm
2. Turn into anytime algorithm
3. Create performance profile
4. Compose with other anytime algorithms into system

Anytime Algorithm Development



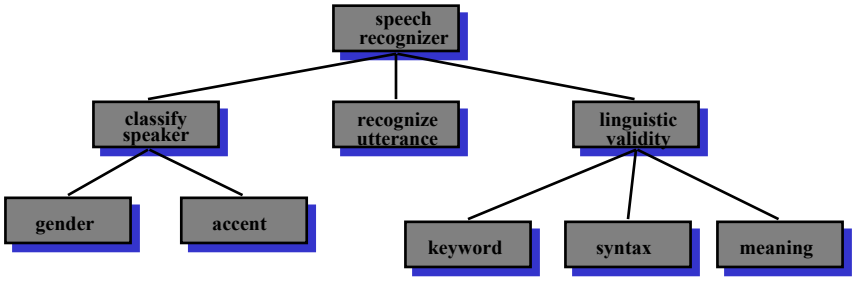
- Building large systems made up of anytime algorithms
- Example compound modules:



- Each algorithm has a performance profile
- A compound module is a graph of algorithms without timing information
- After compilation, the compound has a system profile with timing information, and a monitor
- The monitor observes run-time performance wrt the system profile to determine run-time reallocations and interruptions

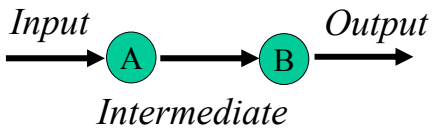
(adapted from Zilberstein and Russell 96)

The Composition Problem



- Why is composition important?
- What problems does it present?
- What solutions are currently available?

Composing Two Algorithms



- Consider example:
  - Patient arrives at emergency room with symptoms → *Input*
  - Some time is spent diagnosing the patient, with time-dependent utility → Diagnosis(*Input)*
  - Then diagnosis is used to determine treatment, with time-dependent AND *diagnosis-dependent* utility → Treatment(Diagnosis(*Input))*
  - Overall quality of solution is composition of result of diagnosis used as input to treatment
  - *Output* ← Treatment(Diagnosis(*Input))*

Compilation Issues

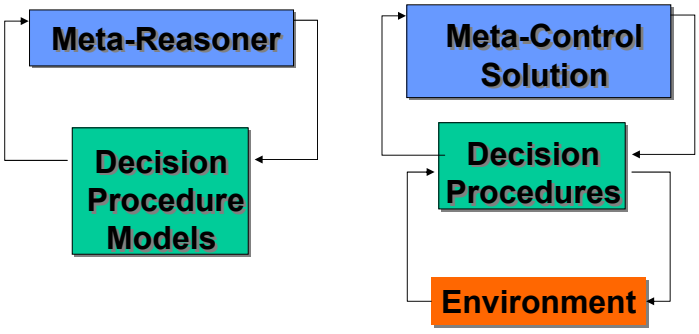
- How can algorithms be combined?
  - Simple chains
  - Graphs
  - Recursion
  - Algorithms with side-effects
- Complexity of compilation is determined by:
  - Composition language
  - Form of profiles
  - Choice of run-time monitoring scheme

Compilation Problem

- Given
  - System of anytime algorithms
  - Total time allocation
- Off-line
  - Allocate resources to algorithms to optimize expected quality of result
  - Build composite profile for system
- On-line
  - *Monitor* progress of algorithms for possible interruption and re-allocation

Off-line vs. On-line Meta-control

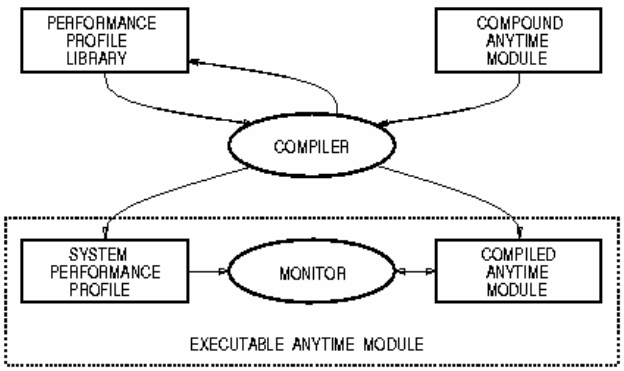
- Model-based meta-reasoning can be completed off-line



Off-line And On-line Components Of Time-critical Systems Building

Off-line

On-line



(adapted from Zilberstein and Russell 96)

Monitoring

- Can we do better than strictly relying on off-line models?
- Use feedback from on-line computation to alter meta-level control
- Particularly important when faced with high variance or unexpected interruptions
- Details later

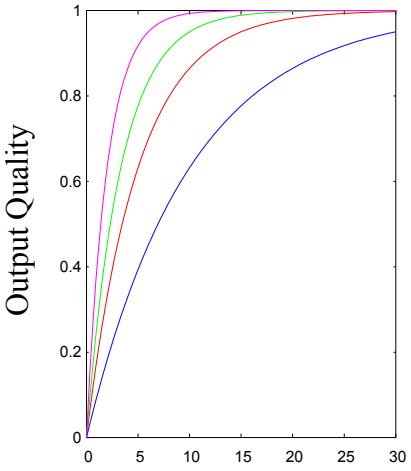
New Modeling Concept Needed: Conditional Performance Profile

- Taking expectation over all inputs may yield very inaccurate profile
- Solution
  - Classify possible inputs
  - Create separate expected profile for each input class
  - Use any available information about inputs to classify
- Expected quality relative to decision procedure and input class

$$Q_{A,I}(t) = \sum_{q'} \Pr_{A,I}(q' \mid t) q'$$

- Can also use three dimensional version

$$Q_A(q,t) = \sum_{q'} \Pr_A(q' \mid q,t) q'$$



Expected Profiles And Composition

- Goal is to compose systems of anytime algorithms
- Profile of system is function of profiles of components
- Unfortunately:
  - Expected value is not closed under composition

$$Q_A(q,t) = \sum_{q'} \Pr_A(q' \mid q,t) q' \quad Q_B(q,t) = \sum_{q'} \Pr_B(q' \mid q,t) q'$$

$$Q_{f(A,B)}(q,t) = \sum_{q'} \Pr_{f(A,B)}(q' \mid q,t) q'$$

$$Q_{f(A,B)}(q,t) \neq f(Q_A(q,t), Q_B(q,t))$$

- Options
  - Study the basic ideas assuming minimal variance → approximately deterministic profiles
  - Use tabular representation in compilation algorithms $\Pr(q \mid t)$

CPP Of Greedy Selection

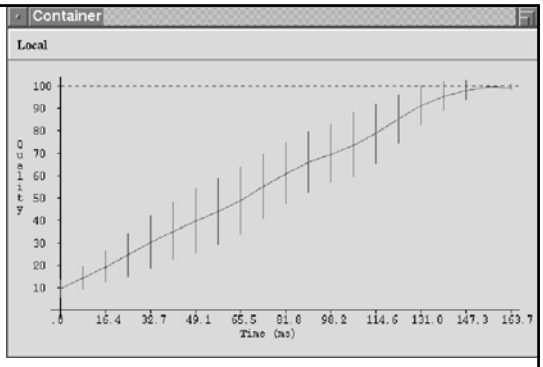
- Sort then Select
- Maximal input quality (output of sorting algoritm)

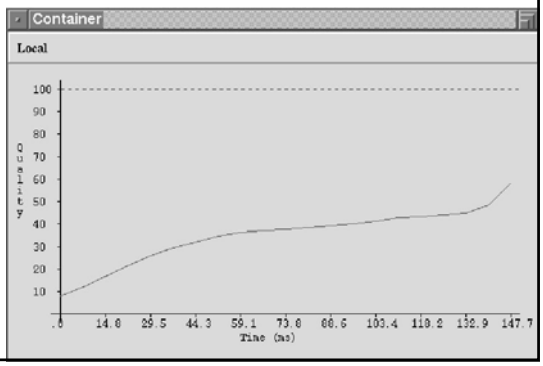
$$Q_{SEEK,SORTMAX}(t) = \sum_{q'} \Pr_{SEEK,SORTMAX}(q' \mid t) q'$$

- Minimal input quality

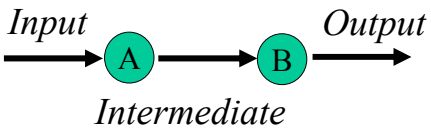
$$Q_{SEEK,SORTMIN}(t) = \sum_{q'} \Pr_{SEEK,SORTMIN}(q' \mid t) q'$$

(adapted from Grass and Zilberstein 96)





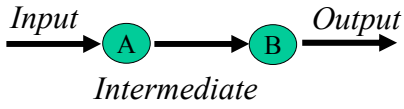
Composing Two Algorithms



Two Examples

1. Closed form profiles
   - Diagnosis and treatment
2. Tabular profile representation
   - Robot path planning (sensing and planning)

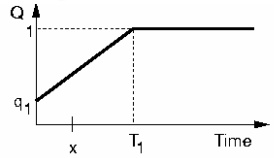
Compilation With Closed-Form Profiles

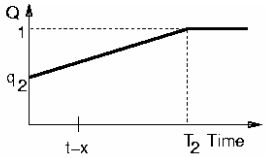


Output ← Treatment(Diagnosis(*Input*))

- Input: patient symptoms
- Intermediate: diagnosis of possibly injuries
- Output: Treatment action/plan

- Profiles: Linear deterministic independent profiles
  - $Q_1(t) = q_1+\alpha_1 t$ $(0 \le t \le T_1)$ – *probability diagnosis is correct*
  - $Q_2(t) = q_2+\alpha_2 t$ $(0 \le t \le T_2)$ –*probability treatment fixes problem, given diagnosis*





(adapted from Zilberstein and Russell 96)

Compilation With Closed-Form Profiles

- What is best allocation? (i.e. contract time to each algorithm)
  - Allocation that maximizes overall quality. Split total time between each algorithm:
    - $\mathcal{T}: \mathcal{R}^+ \rightarrow \mathcal{R}^+ \times \mathcal{R}^+$
  - Overall quality is product of two independent qualities – quality of second algorithm does not depend on first
    - $Q(x) = Q_1(x) * Q_2(t-x)$
    - $= (q_1+\alpha_1 x)\ (q_2+\alpha_2(t-x))$
    - $= q_1 q_2 + q_2\alpha_1 x + q_1\alpha_2 t - q_1\alpha_2 x + \alpha_2 t\alpha_1 x - \alpha_2\alpha_1 x^2$
  - For total time t, find x that maximizes Q(x)
  - Combined profile gives optimal quality for any total time allocation – compiler needs to solve for each t

(adapted from Zilberstein and Russell 96)

Optimal Allocation

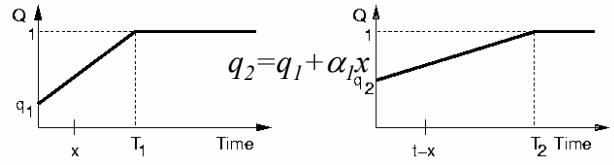
$$\mathcal{T} : t \rightarrow (\frac{1}{2}(t - \frac{q_1}{\alpha_1} + \frac{q_2}{\alpha_2}), \frac{1}{2}(t + \frac{q_1}{\alpha_1} - \frac{q_2}{\alpha_2}))$$

- Proof:
  - $Q(x) = Q_1(x) * Q_2(t-x)$
  - $= (q_1+\alpha_1 x)\ (q_2+\alpha_2(t-x))$
  - $= q_1 q_2 + q_2\alpha_1 x + q_1\alpha_2 t - q_1\alpha_2 x + \alpha_2 t\alpha_1 x - \alpha_2\alpha_1 x^2$
  - For total time t, find x that maximizes Q(x)
  - Maximum achieved when ∂ Q(x) /∂x = 0
    - ∂ Q(x) /∂x = $q_2\alpha_1 - q_1\alpha_2 + \alpha_2 t\alpha_1 - 2\alpha_2\alpha_1 x$
    - Setting equal to 0 and solving for x gives solution above

(adapted from Zilberstein and Russell 96)

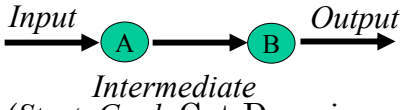
What If The Qualities Are Not Independent?

- If q_2 is defined as quality of first algorithm we have:
  - $Q(x) = Q_1(x) * Q_2(t-x)$
  - $= (q_1+\alpha_1 x)\ ((q_1+\alpha_1 x)+\alpha_2(t-x))$
  - $= q_1^2 + 2q_1\alpha_1 x + \alpha_1^2 x^2 + q_1\alpha_2 t - q_1\alpha_2 x + \alpha_2 t\alpha_1 x - \alpha_2\alpha_1 x^2$
- For total time t, find x that maximizes Q(x)
- Maximum achieved when ∂ Q(x) /∂x = 0
  - ∂ Q(x) /∂x = $2q_1\alpha_1 + 2\alpha_1^2 x - q_1\alpha_2 + \alpha_2 t\alpha_1 - 2\alpha_2\alpha_1 x$
  - Setting equal to 0 and solving for x gives ?



(adapted from Zilberstein and Russell 96)

Compilation With Tabular Profiles

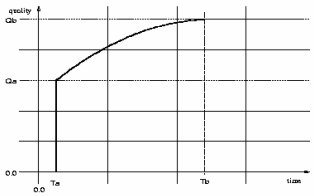


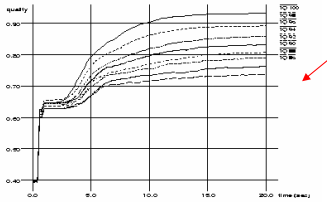
Output ← Path-Plan(*Start*, *Goal*, Get-Domain-Description(*Input*))

- Input: raw data for visual sensing, start and goal locations
- Intermediate: approximate map of environment
- Output: path from start to goal (coarse-to-fine search)

- Tradeoffs
  - Better map → better plan
  - More time spent making map → less time planning

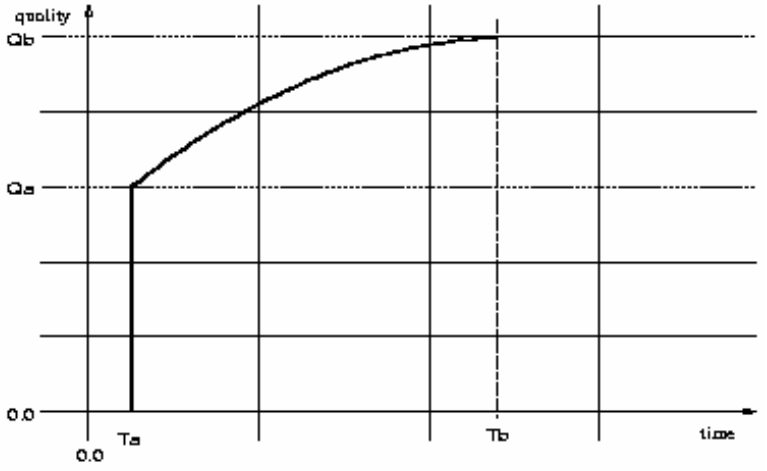
conditional





(adapted from Zilberstein and Russell 96)

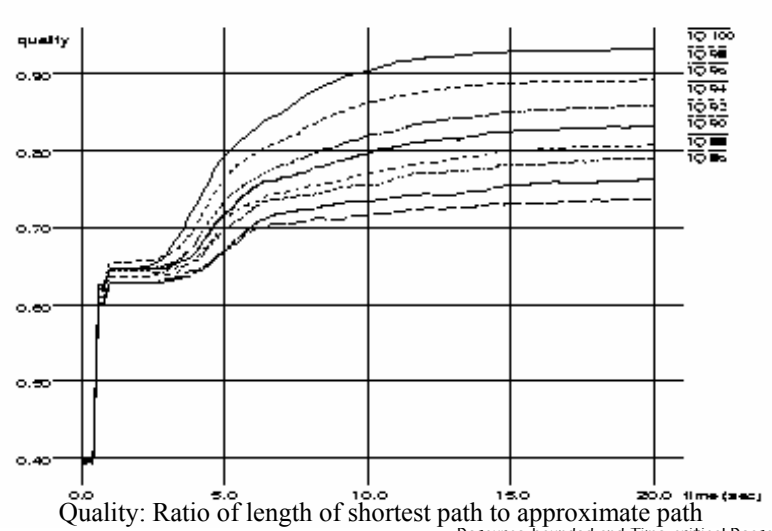
Profiles Expanded: Map Building



Quality: probability that "blocked" space is viewed as free and vice-versa

(adapted from Zilberstein and Russell 96)

Profiles Expanded: Path Planning – Conditional Profile

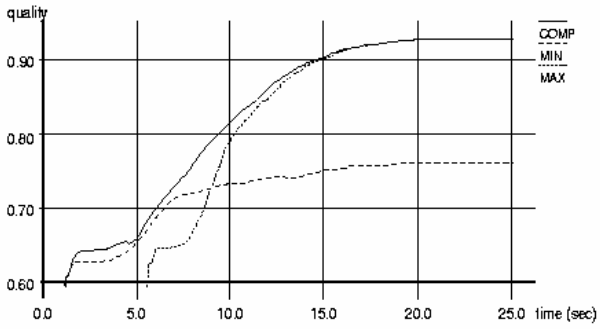


Quality: Ratio of length of shortest path to approximate path

(adapted from Zilberstein and Russell 96)

Compilation With Tabular Profiles

- Resulting profile after optimal compilation – using discrete optimization via search
- Optimal plus default profiles with Min (Ta) and Max (Tb) allocations to vision



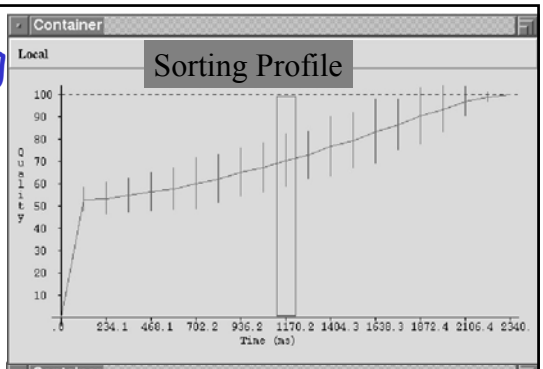
(adapted from Zilberstein and Russell 96)

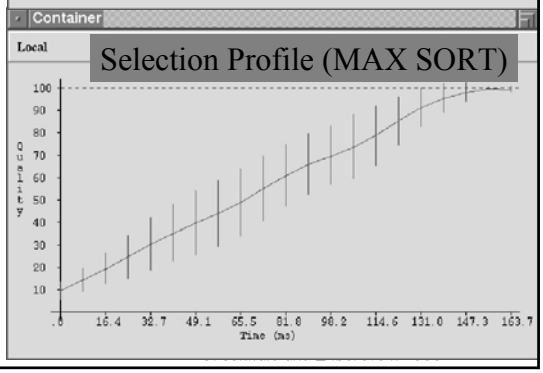
Composing Sorting And Selection

- Sort then Select
- Maximal input quality (output of sorting algoritm)

$$Q_{SEEK,SORTMAX}(t) = \sum_{q'} \Pr_{SEEK,SORTMAX}(q' \mid t) q'$$

- Need profile for composed system

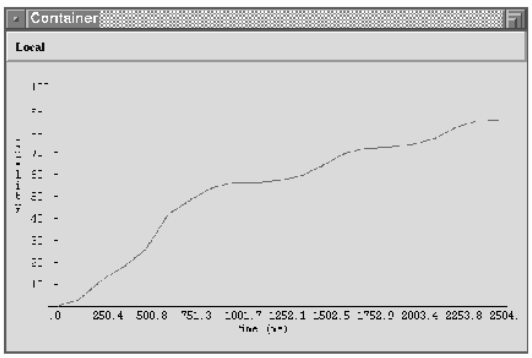


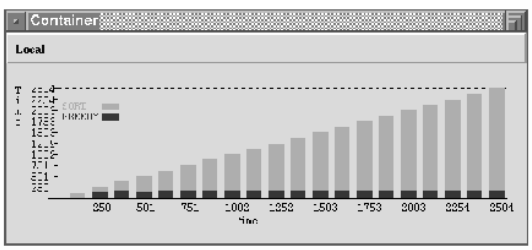


(adapted from Grass and Zilberstein 96)

Compilation

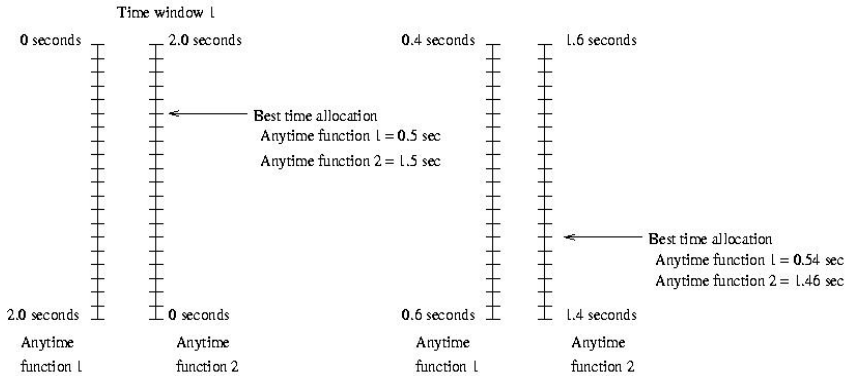
- First determine worst case total time for sorting+seeking
- Apply *multi-resolution search*
  - An incremental algorithm to consider all possible combinations of time allocations
  - By increasing resolution of time, incrementally
- Build a new combined profile





(adapted from Grass and Zilberstein 96)

Multiple Resolution Search Meta-control Algorithm For Compilation



(adapted from Grass and Zilberstein 96)

Compiling The Profiles: Multiplicative Probabilities

$$E\{Q_O\} = \sum_{q_j}(\sum_{q_i} Q_N(t_N)[q_i]Q_M(q_i, t_M)[q_j])q_j$$

Probabilities

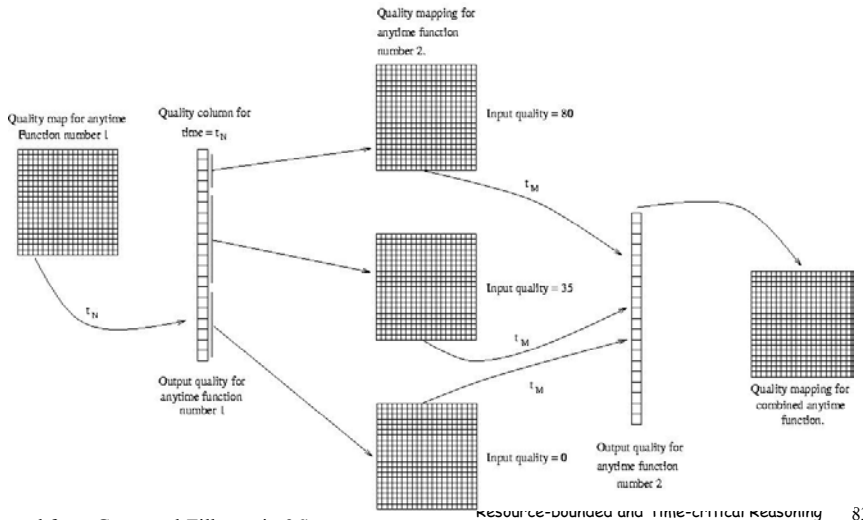
Where:

q_i	is the output quality of the first function
q_j	is the output quality of the second function
Q_O	is the overall output quality
Q_N	is the performance profile of the first function
Q_M	is the performance profile of the second function
t_N	is the time allocation to the first function
t_M	is the time allocation to the second function

$$Q_M(q, t_M)[q'] = \Pr_M(q' \mid q, t_M)$$

(adapted from Grass and Zilberstein 96)

Compilation: Graphical View



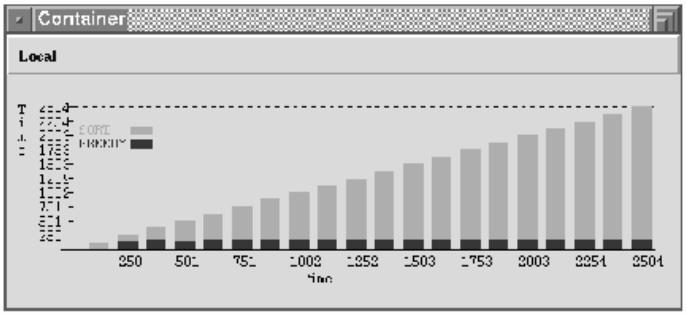
(adapted from Grass and Zilberstein 96)

Composed Algorithm Is A Contract Algorithm

- Anytime algorithm: Can be stopped at "any time"
- What if you can't interrupt an algorithm once it starts?
- Contract algorithm: "Any time" assumption true only at solution design time – not at run time
  - Can not be stopped prior to "contracted" time
  - Term coined by [Zilberstein and Russell]
  - Example: Algorithms that take the deadline as input
    - Depth-bounded or cost-bounded tree search: Given a deadline, set the depth limit so that all the information is processed by the deadline.
    - Discretizing a continuous MDP: Given a deadline, partition the state set so that dynamic programming will finish by the deadline.

(adapted from Bernstein 2002)

Composed Algorithm Is A Contract Algorithm



- Compilation result is an allocation of time for each component, given a total allocation
- But, you can't go from one allocation to the next, incrementally

(adapted from Grass and Zilberstein 96)

Compilation Of Functional Expressions

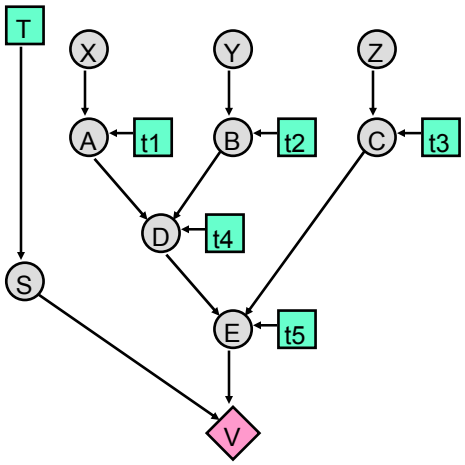
- Functional composition of anytime algorithms
  - Parameters passed by value
  - No side effects
  - Formally: A *functional expression* over the set $\mathcal{F}$ of anytime algorithms with input variables I is
    1. An input variable in I
    2. An expression $f(g_1,\ldots,g_n)$ where $f \in \mathcal{F}$ and each g_i is a functional expression
- Goal: find allocation of t total time to anytime algorithms to optimize composition quality

Influence Diagram Solution

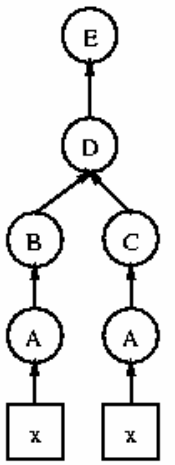
- Given the expression:

  E(D(A(x),B(y)),C(z))

- Construct and solve the following influence diagram:



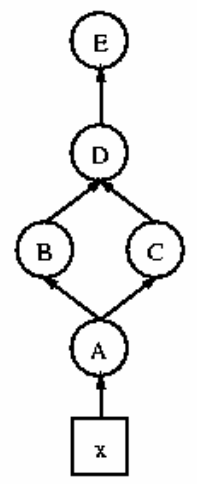
Tree-structured Composition



- $f(g_1,\ldots,g_n)$
- Input variables are at the leaves
- Subexpressions form trees with root f and children subtrees $g_1,\ldots,g_n$
- Multiple copies of nodes for repeated subexpressions
- Example: *F(x)=E(D(B(A(x)),C(A(x))))*

(adapted from Zilberstein and Russell 96)

DAG-structured Composition



- $f(g_1,\ldots,g_n)$
- Input variables are at leaves
- Subexpressions form DAGs with node f and links from children DAGs $g_1,\ldots,g_n$
- Needs only one DAG for each repeated subexpression

(adapted from Zilberstein and Russell 96)

Calculating The Overall CPP

- Global optimization is NP-complete
- Local compilation reduces the complexity
- It is applicable to tree-structured programs
- It is optimal under certain conditions
- Complexity $O(nkT^b)$ for a program composed of n components with k output qualities, maximal allocation of T time units, and a branching factor b.

Compiling Deterministic CPPs

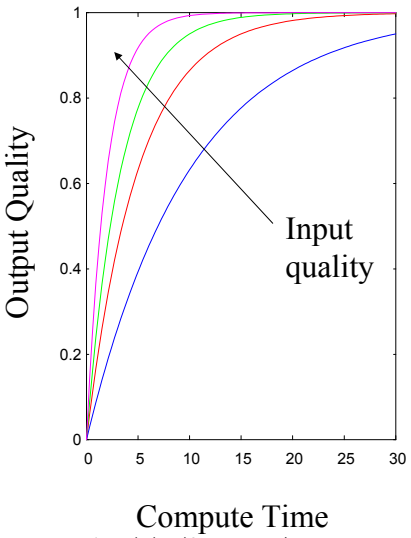
Theorem [Zilberstein IJCAI-95]: Optimality of local compilation of deterministic CPPs: *Let e be a composite expression whose CPPs satisfy the input monotonicity assumption, then for any input and contract time t:*

$$Q_e^L(t) = Q_e^G(t)$$

Proof: By induction on the depth of the tree using the local optimality of local compilation.

Input Monotonicity

- We'd like conditional performance profiles to behave "reasonably" as inputs vary, namely
  - As input quality improves, so should the profile



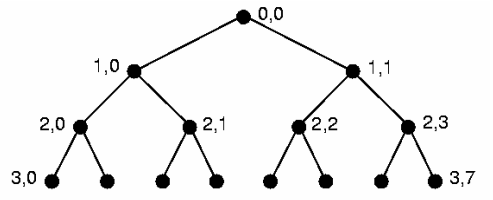
Compiling Probabilistic CPPs

Theorem [Zilberstein IJCAI-95]:
Optimality of local compilation of probabilistic CPPs: *Let e be a composite expression whose CPPs satisfy the input linearity assumption, then for any input and contract time t:*

$$Q_e^L(t) = Q_e^G(t)$$

Proof: An immediate result of the previous theorem and input linearity.

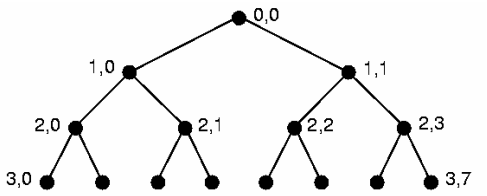
Local Compilation



- Only consider profiles of immediate sub-components to a node
  - Treat sub-components as elementary anytime algorithms
  - Allocating time one piece at a time --- not considering non-local effects
- Each node has conditional performance profile
  - *output quality (input qualities, time allocation)*
  - $Q_{node_{i,j}}(q_1, q_2, t)$
- Allocate time to each node to maximize quality of root

(adapted from Zilberstein and Russell 96)

Local Compilation



- Leaf node profile: $Q^L_{n,j}(t) = Q_{n,j}(q_{n,j,1}, q_{n,j,2}, t)$.
- Internal nodes:

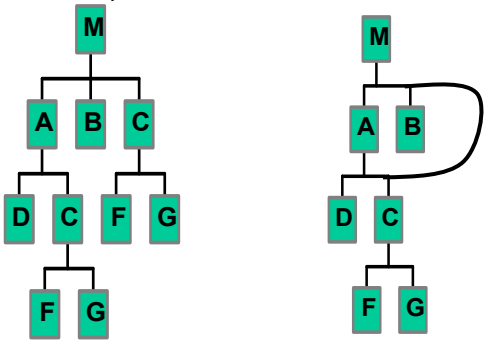
$$Q^L_{i,j}(t) = arg\max_{t_1,t_2}\{Q_{i,j}(Q^L_{i+1,2j}(t_1), Q^L_{i+1,2j+1}(t_2), t - t_1 - t_2)\}$$

- For each internal node, consider all allocations to its sub-components and pick allocation that maximizes node quality, for each total time – locally compiled performance profile
  - For example, multi-resolution search over t_1 and t_2
- Assumes input quality is given prior to compilation

(adapted from Zilberstein and Russell 96)

Repeated Sub-expressions

- One activation of multiple copies is needed
- Graph representation becomes a DAG
- Local compilation does not work



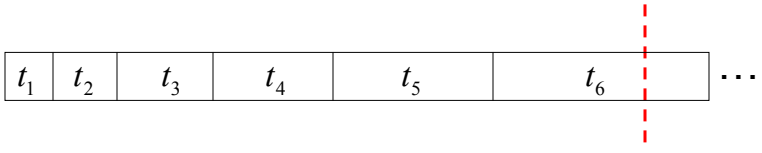
Approximate Time Allocation

Beyond the tree-structure assumption [Zilberstein 1993]

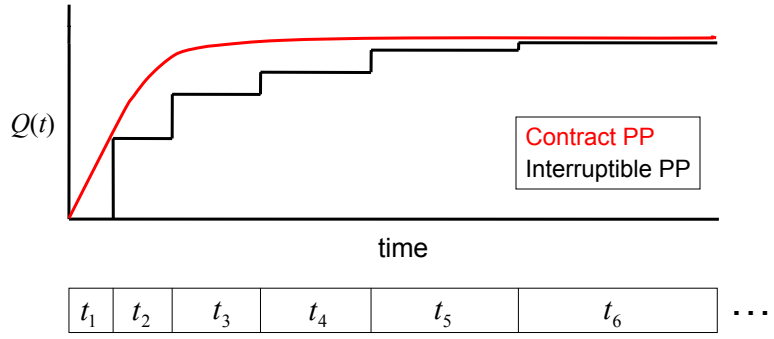
- Hill-Climbing-Allocation -- solves the global compilation problem directly.
- Conditioning-Allocation -- 2-level search using local compilation as a subroutine.
- Trading-Allocation -- uses local compilation to trade time among repeated sub-expressions until only one copy receives non-zero time.

Contract To Interruptible Transformations

- What if we want to use a contract algorithm in a setting where we do not know the deadline?
- We can repeatedly activate the contract algorithm with increasing run times.
- When the deadline occurs, we can return the result produced by the last contract to finish:



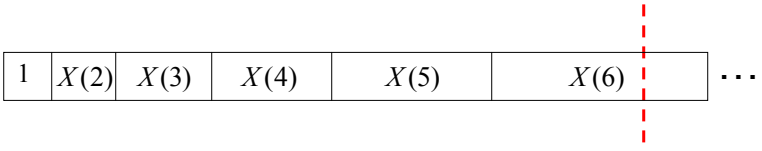
The Resulting Performance Profile



- We want to minimize the inefficiency due to scheduling the contract algorithm.

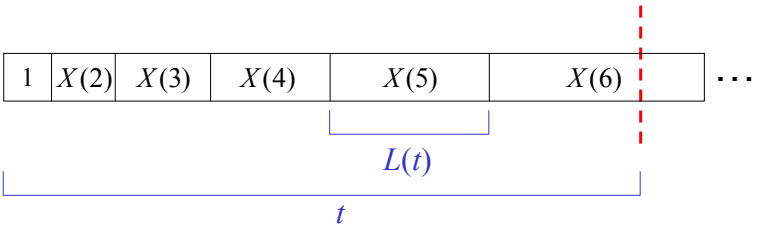
Some Details

- We use X to denote a schedule.
- $X(i)$ is the length of the ith contract algorithm run (we assume w.l.o.g. $X(1) = 1$).
- $G(i) = X(1) + X(2) + \cdots + X(i)$ is the total time taken by the first i contracts.
- We define $L(t)$ to be the length of the last contract algorithm to finish before time t.



Inefficiency Of Scheduling

- If the interruption occurs at time t, the loss from scheduling is $t / L(t)$.



- We are interested in minimizing the time wasted in the worst case.

Acceleration Ratio

- A worst-case measure of the inefficiency of the resulting interruptible algorithm [Russell, Subramanian, and Parr 1993]

$$R(X) = \sup_{t>1} \frac{t}{L(t)}.$$

- Note that the worst time for a contract algorithm to be interrupted is just before it returns a result.
- So, $$R(X) = \sup_{i \neq 1} \frac{G(i)}{X(i-1)}.$$

An Exponential Schedule

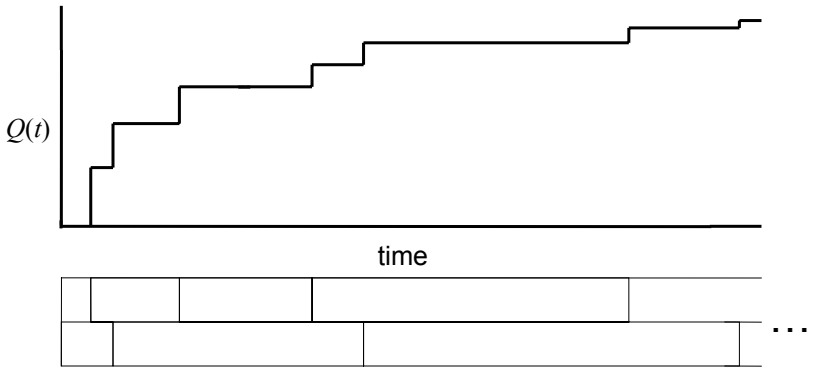
$$X(i) = 2^{i-1}$$

- This schedule achieves an acceleration ratio of 4:

$$\frac{G(2)}{X(1)} = \frac{3}{1}, \frac{G(3)}{X(2)} = \frac{7}{2}, \frac{G(4)}{X(3)} = \frac{15}{4}, \cdots, \frac{G(i)}{X(i-1)} = \frac{2^i - 1}{2^{i-2}}, \cdots$$

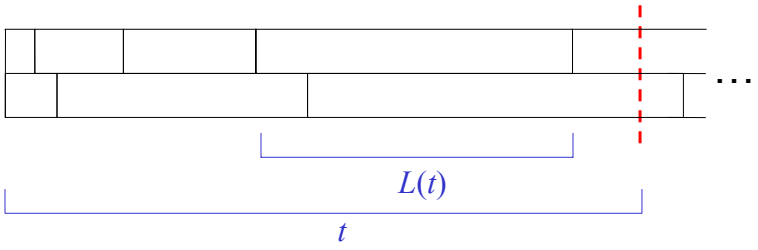
- Is this schedule optimal?
- From [Russell and Zilberstein 1991]. First proof of optimality in [Zilberstein, Charpillet, and Chassaing 1999]

Multiple Processor Case



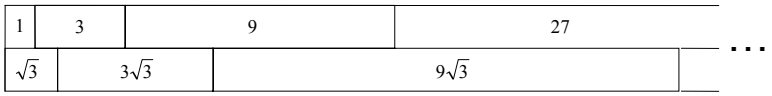
Some Details

- $X(i, j)$ is the length of the ith contract run on the jth processor.
- Upon interruption, the best result from any processor is returned.
- Acceleration ratio:



A Generalized Exponential Schedule

- For m processors, the ith contract algorithm run on any processor has length $[(m+1)^{1/m}]^{i-1}$.
- In the two processor case:



Acceleration Ratio

- It is easy to show that

$$R(X) = \frac{(m+1)^{(m+1)/m}}{m}$$

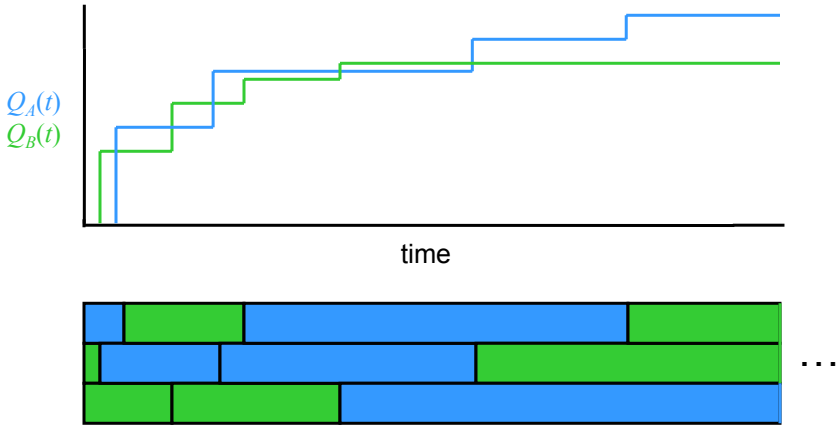
- We also show that no better ratio can be achieved.
- This is a nontrivial extension of the single processor proof [Bernstein, Perkins, Zilberstein, Finkelstein 2002]

Some Optimal Acceleration Ratios

m	Optimal ratio
1	4.00
2	2.59
3	2.11
4	1.86
10	1.39
100	1.05
∞	1.00

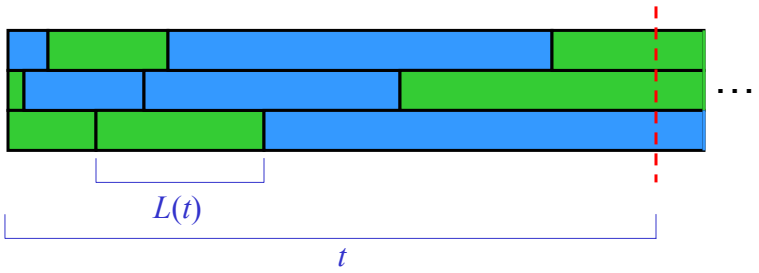
- At $m = \infty$, there is no distinction between contract and interruptible algorithms.

Multiple Problem Instances



Some Details

- $X(i,j) = (r,d)$, where r is the problem and d is the time spent
- Upon interruption, a solution to the least-worked-on problem will be requested
- Acceleration ratio:



A More General Exponential Schedule

- For m processors and n problems, the ith contract algorithm run on any processor has length $[((m+n)/n)^{1/m}]^{i-1}$.
- Problems are handled in a round-robin fashion.

Acceleration Ratio

- It is easy to show that

$$R(X) = \left(\frac{n}{m}\right)\left(\frac{m+n}{n}\right)^{\frac{m+n}{m}}$$

- We can also prove that no better ratio can be achieved by any schedule with the following properties:
  - Problems are addressed in a round-robin manner
  - The lengths of contracts for each problem increase with time
- In [Bernstein, Finkelstein, Zilberstein 2003]

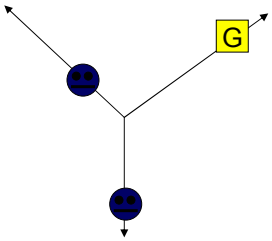
Some Optimal Acceleration Ratios

problems →

↓ processors

m \ n	1	2	10	100	∞
1	4.00	6.75	28.53	273.1	∞
2	2.59	4.00	14.92	137.2	∞
10	1.39	1.71	4.00	28.53	∞
100	1.05	1.10	1.39	4.00	∞
∞	1.00	1.00	1.00	1.00	N/A

A Related Problem: Search On Rays



2 robots,
3 rays

- Imagine m robots searching for a goal on one of p rays (where $m < p$)
- The search starts with all robots at the origin

Competitive Ratio

- $X(i,j) = (r,d)$, where r is the ray and d is the distance travelled out on the ray
- The competitive ratio, $C(X)$:

$$\frac{\text{actual search time}}{\text{search time if goal location was known}}$$

  for the worst possible goal location
- For the one robot case, the best ratio was known previously
- In [Kao, Ma, Sipser, Yin 1998]
- The general case is solved in [Bernstein, Finkelstein, and Zilberstein 2003].

Back To Contract Algorithms

- There has also been work on incorporating probabilistic information about the performance profile and deadline.
- A stochastic deadline, $P_D(t)$, is a probability distribution of the deadline over time.
- A stochastic performance profile, $P_A(q|t)$, is a distribution of the output quality conditioned on the run time of the contract algorithm.

Using Probabilistic Information

- With probabilistic information the problem is to find a scheduling strategy that maximizes the expected result quality at the deadline.
- Under certain assumptions, the one-processor, one-problem version can be formalized as an MDP:
  - States: (quality so far, time so far)
  - Actions: contract lengths
- Multi-processor, multi-problem case?

Possible Extensions

- Even with probabilistic information, the contract algorithm model still has a worst-case aspect.
- We assume that when the contract algorithm finishes, the next instantiation must "start from scratch."
- It may be useful to explicitly model the information carry-over from one computation to another.

Outline

- Introduction to meta-level control of computation
- Algorithm design, performance modeling and prediction
- Algorithm composition and transformations
- Meta-reasoning and deliberation scheduling
- Sample applications
- Current research directions and wrap-up

Meta-reasoning And Deliberation Scheduling

Recall: Applying Decision Theory At Meta-level

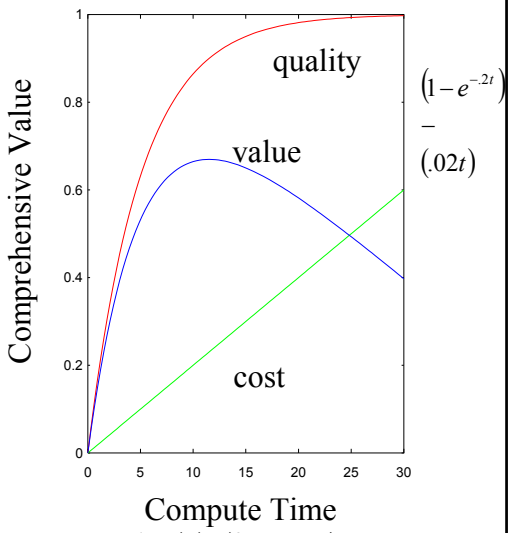
- Recall: *Comprehensive value* captures
  - Physical action effect on world utility
  - Resource consumption costs due to inference
- Meta-level rationality
  - Choose inferential action (or sequence) that maximizes expected value of world outcomes
  - Under uncertainty -- weigh each outcome (state) by probability that it occurs
- World Utility = Expected Comprehensive Value Assumption
  - Assume comprehensive value completely captures expected utility of physical action taken after inferential action
  - Rational meta-level agent chooses inferential action that maximizes comprehensive value

Solving Meta-level Control Problems

- Composing a sequence of decision procedures – *composition*
- Stopping of single decision procedure
- Resource allocation over a set of alternative decision procedures – *portfolio*
- Resource allocation across a sequence of required decision procedures with deadlines – *schedule*
- Conditional scheduling of a sequence of decision procedures – *policy*

Optimal Stopping Problem

- Single decision procedure
  - Anytime algorithm
- Meta-level decision
  - Choose how long to run anytime algorithm before selecting physical action



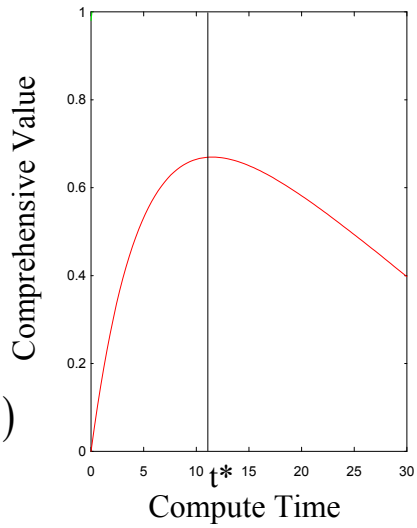
Optimal Stopping Example

- Closed-form model of comprehensive value

$$(1-e^{-.2t})-(.02t)$$

- Find t* such that

$$t^* = \arg\max_t (1-e^{-.2t})-(.02t)$$



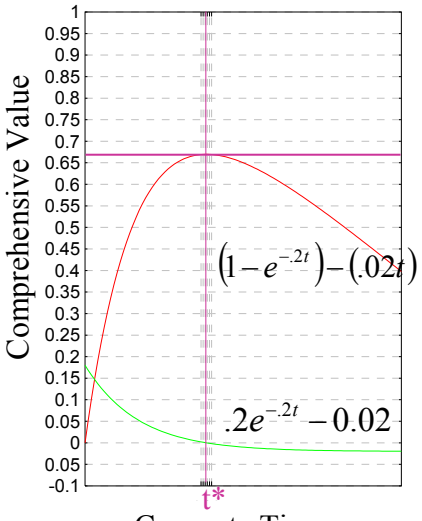
Optimal Stopping: Approach I

- Take derivative of closed-form model of expected comprehensive value function

$$\frac{d}{dt}\left(1-e^{-.2t}\right)-(.02t)$$

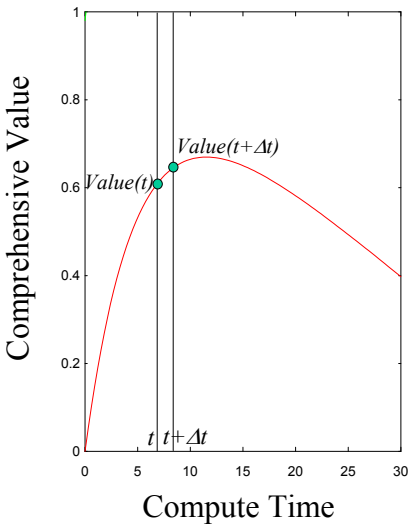
$$=.2e^{-.2t}-0.02$$

- Find t such that derivative is zero
  - t* = -5 ln(0.1) = 11.51
  - Value = (1-e $^{-.2*11.51}$) - .02*11.51 = .67



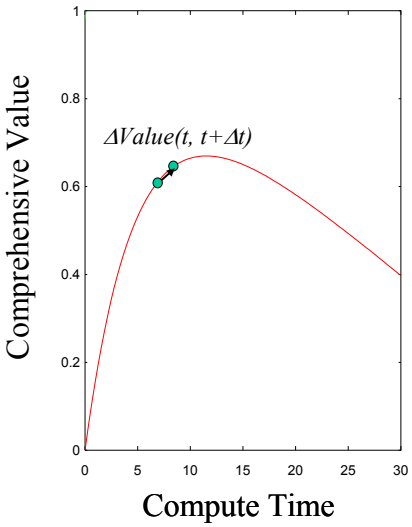
Optimal Stopping: Approach II

- Re-express comprehensive value as an incremental expression
  - Having already computed for *t* time
  - What is expected quality improvement for Δt more time?
  - Δ*Value(t, t+*Δ*t) = Value(t+*Δ*t) - Value(t)*
- Also called marginal *expected value of computation*



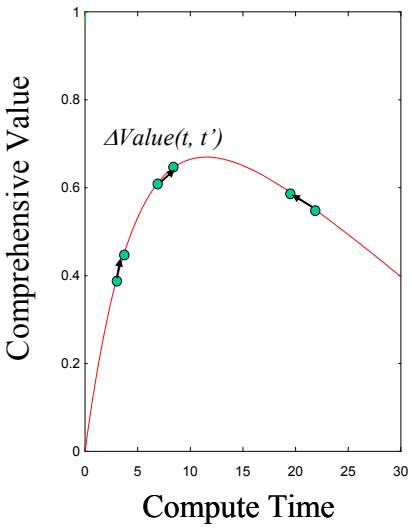
Optimal Stopping: Approach II

- Iterative meta-level control algorithm
  - *t = 0*
  - While (*ΔValue(t, t+Δt) > 0*)
    - *t = t+Δt*
  - Stop. Return *t*.
- *Greedy on-line search*
  - Works forward in time – can be applied on-line
- Treats *ΔValue(t, t+Δt)* as a *black box*
  - Could be closed-form model summarizing non-local data
  - Or statistics-driven with local or non-local data



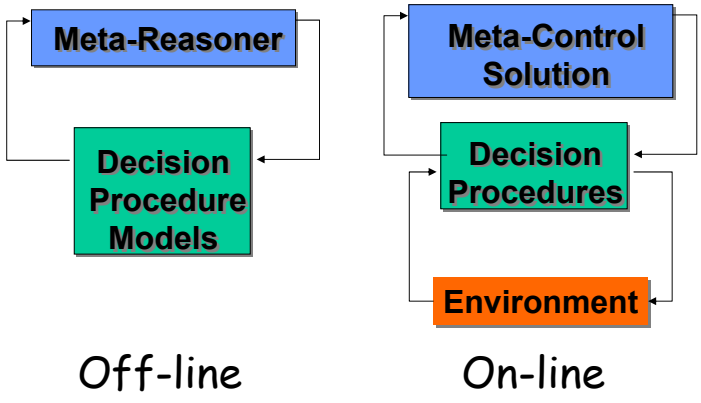
Optimal Stopping: Approach III

- Limitations of greedy approach
  - Finds first local optima
    - global optimum under certain conditions i.e. diminishing returns
  - Complexity and accuracy depend on *Δt*
- Can also apply Hill climbing with gradient *ΔValue(t, t')*
  - If off-line, not limited to finding first local optima
    - Simulated annealing
    - Random re-starts
    - Etc.
  - Can be extended to multiple dimensions



Off-line vs. On-line Meta-control

- Meta-reasoning algorithms based on *models* can be executed completely off-line



Monitoring – Handling Uncertainty

- Algorithm performance is not always known deterministically
- Profile may be:
  - Approximate model based on limited data (uncertainty about performance)
  - True model with considerable variability (randomized algorithm, uncertainty about world/problem)
- Sources of uncertainty
  - Ignorance – not enough data, novel situations
  - Approximation
    - Deterministic model easier to manipulate efficiently
    - True model leads to computationally intractable meta-deliberation problem
  - Stochastic dynamics – world really does behave non-deterministically
  - World may change during deliberation
  - Changing requirements

Monitoring Questions

- How should the variance in solution quality affect the frequency of monitoring?
- How should the cost of monitoring affect the frequency of monitoring?
- When solution quality is hard to determine, what degree of approximation should be used?
- How does approximation of solution quality degrade the effectiveness of monitoring?
- Is it better to monitor periodically or more frequently toward the expected stopping time?

Monitoring – Handling Uncertainty

- On-line feedback
- Monitoring contract algorithms
- On-line stopping
- Off-line conditional plans/policy
- Approximate feedback
- Current Research
  - Model-free approach

Monitoring – On-line Feedback

- Feedback
  - From algorithm itself
  - From observing changes to world
  - Variance from off-line models
    - Expected performance
- Example: measurement of current intrinsic solution quality
  - With respect to initial solution
  - With respect to optimal solution – approximation ratio
  - With respect to bound on optimal solution obtained from relaxed problem
- Difficulties
  - Quality not directly observable
- Cost

Monitoring Contract Algorithms

- $Q_A(q,t)[q_i]$ = the probability of output quality q_i
- $U_A(S,t,q_i)$ = utility of results of quality q_i
- Since the quality of future results is unknown:

$$U_A'(S,t) = \Sigma_i\, Q_A(q,t)[q_i]\, U_A(S,t,q_i)$$

- Since the future state of the domain is unknown:

$$U_A''(t) = \Sigma_s\, Pr(S_t = S|\, S_0)\, U_A'(S,t)$$

- The best contract time:

$$t_c = \text{argmax}_t \{U_A''(t)\}$$

- In [Zilberstein 1993]

Optimality Of The Fixed-contract Approach

Theorem: The fixed-contract monitoring strategy is optimal when the domain has predictable utility, and the system has a deterministic PP.

Examples:

-- Data transfer with limited but known bandwidth

-- Multiple methods for diagnosis and repair

Optimality of the fixed-contract approach in other situations

Re-deriving Comprehensive Value

- Quality of inference action

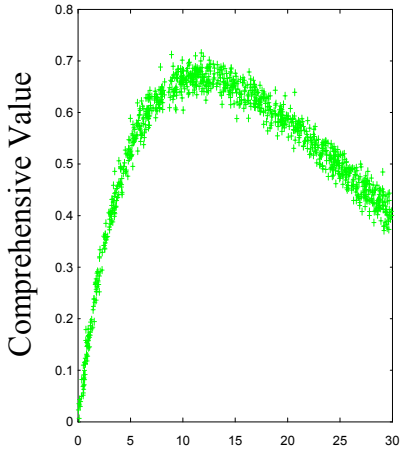
$$Q(t) = \sum_q \Pr(q \mid t) q$$

- Intrinsic utility of inference action

$$IV(t) = \sum_q \Pr(q \mid t) U(q)$$

- Comprehensive Value

$$Value(t) = \sum_q \Pr(q \mid t) U(q,t)$$



Incremental Comprehensive Value

- Now let's create an incremental version

$$\Delta Value(q,t,t+\Delta t) = \sum_{q'} \Pr(q' \mid q,t,t+\Delta t)(U(q',t+\Delta t) - U(q,t))$$

- Compare to previous version
  $\Delta Value(t, t+\Delta t) = Value(t+\Delta t) - Value(t)$
- Doesn't treat Value as a block box
  - Can reason about terms
- New version uses more information
  - current quality level
  - computation expended to get there
- Also called marginal *myopic expected value of computation*

Monitoring I: Stopping With On-line Feedback

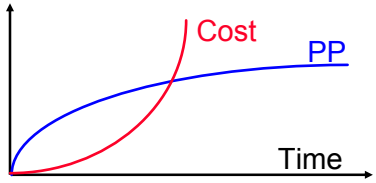
$$\Delta Value(q,t,t+\Delta t) = \sum_{q'} \Pr(q' \mid q,t,t+\Delta t)(U(q',t+\Delta t) - U(q,t))$$

- Iterative on-line meta-level control algorithm with feedback

  $t = 0$
  $q = 0$
  Do
    Compute (deliberate) until $t+\Delta t$
    $t = t+\Delta t$
    Observe $U(q,t)$ /* or observe q and predict U(q,t) */
  Repeat until $(\Delta Value(q,t,t+\Delta t) < 0)$
  Stop

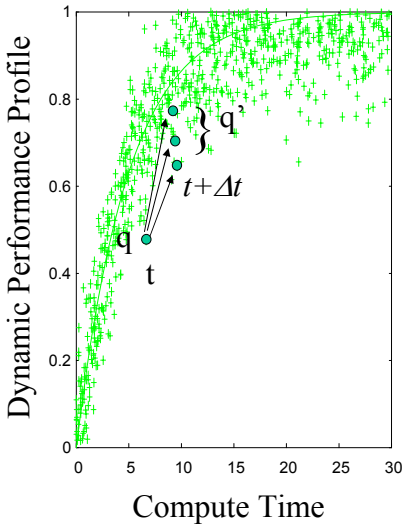
Monitoring I

- Myopic algorithm
  - Only considers expected value at next time step
  - Optimal stopping decision may depend on looking further into future
- Optimal if marginal myopic expected value of computation has diminishing returns
  - Example: the PP is monotonically increasing and concave down, and the cost of time is monotonically increasing and concave up[Zilberstein 1993, Hansen and Zilberstein 2001]
- May account for future by using more data to predict value
  - Model fitting
  - Dynamic programming



A Note On Dynamic Performance Profiles

- Monitoring I algorithm relies on the utility model $U(q,t)$
- As well as the transition model $\Pr(q' \mid q, t, t+\Delta t)$
- The later is a variation of a *dynamic performance profile* (coined by Zilberstein)
  - Makes Markov assumption



Monitoring Policies

Framework:

- Constructing an off-line monitoring policy using dynamic programming.
- Modeling quality improvement with a dynamic performance profile.
- Taking into account the cost of monitoring.
- Monitoring based on exact or estimated solution quality.
- In [Hansen and Zilberstein 1996, 2002]

Monitoring – Off-line Policy Generation

- Sequence of decisions on when to stop deliberation – determined using off-line models of potential on-line information
- Non-myopic: consider both current and future information
- Dynamic programming formulation:

$$V(q,t) = \max(U(q,t), \sum_{q'} \Pr(q' \mid q,t,t+\Delta t) V(q',t+\Delta t))$$

- Monitoring policy: function of current quality and time

$$\pi(q,t) = stop \Leftrightarrow V(q,t) = U(q,t)$$

- Optimally solved with dynamic programming algorithms
  - Polynomial in number of quality levels and time steps
  - Quality must be Markovian
  - Does not take monitoring cost into account
  - Also solved through search (e.g. AO*)

Monitoring Policies Cont.

- Anytime algorithm: traveling salesman problem using randomized tour improvement.

Policy based on actual solution quality

	time-step								
quality	start	...	5	6	7	8	9	10	11
5			0	0	0	0	0	0	0
4			1M	1M	1M	1M	1M	1	0
3			1M	1M	1M	1M	1M	1	0
2			3M	3M	3M	3M	2	1	0
1			4M	5	4	3	2	1	0
0	5M		6	5	4	3	2	1	0

Monitoring With Cost

- Each monitoring action incurs a fixed cost C
- Can reason about frequency of monitoring in addition to stopping time
- Two decisions per time step:
  - How much time to run algorithm (time slices between monitoring not of fixed size)
  - Whether or not to monitor at end
- Also formulated and solved with dynamic programming

Approximate Feedback

- Too costly to determine exact quality at run-time
- Estimate quality
  - Based on easily observable "features"
- Take into account cost of estimation procedure

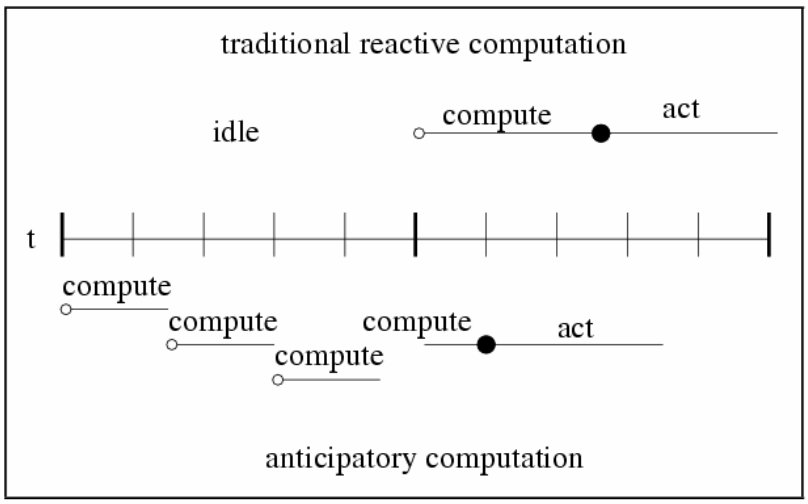
Modeling Issues

- From Hansen and Zilberstein while discussing performance profiles for solving a 12-city TSP instance:
  - "… compiled by generating and solving a thousand random twelve-city traveling salesman problems"
- Too much simulation needed
- No theory on how to generalize over problem instances

Continual Computation

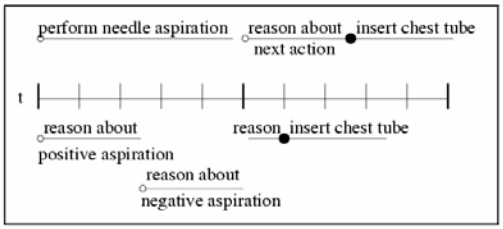
- Uncertain stream of *challenges* encountered over time (also called *problem instances*)
- Some ability to predict *future challenges*
- Allocate computation time to selecting actions for current and future challenges
  - Termed: *pre-computation*
- Motivation: time divided into bursts of challenges and long stretches of *idle time*
  - Example: Web page pre-fetching
- Grand vision: a solution for situated autonomous systems
- Coined in [Horvitz 97], Similar notion called *response planning* [Greenwald, Dean 94]

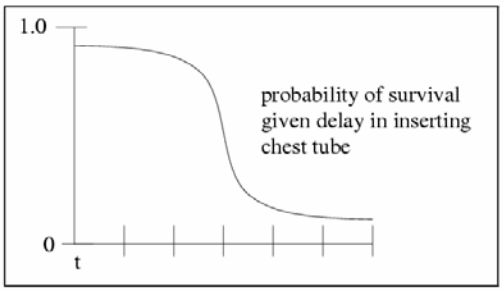
Idle Time Options



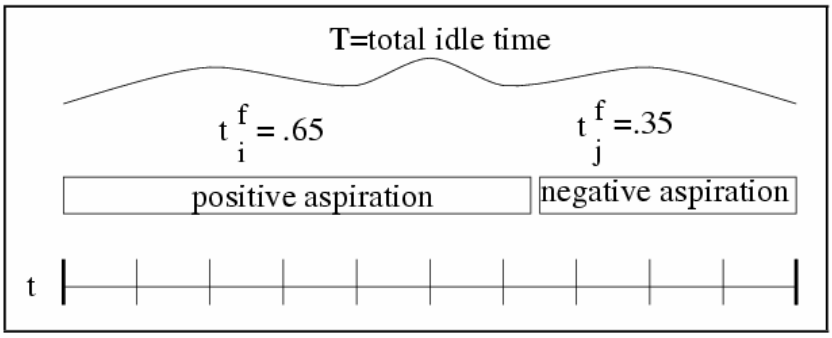
Trauma Care Example

- What to do while waiting for needle aspiration results:
  - Nothing
  - Reason about possible outcomes
  - Reason about other causes of shock
  - Reason about further diagnostic actions
  - Reason about other injuries/challenges





Solution Is A Computational *Portfolio*



Some Simplifying Assumptions

- One decision procedure per challenge – no overlap in reasoning
- Distribution over future challenges known at start of idle time
- No current challenges
- Consider idle time period to be equivalent to:
  - Common release date for all challenges
  - Common, possibly uncertain, deadline for all challenges – when the next real-time challenge occurs
- Only one real-time challenge occurs at deadline and then problem ends

Definitions

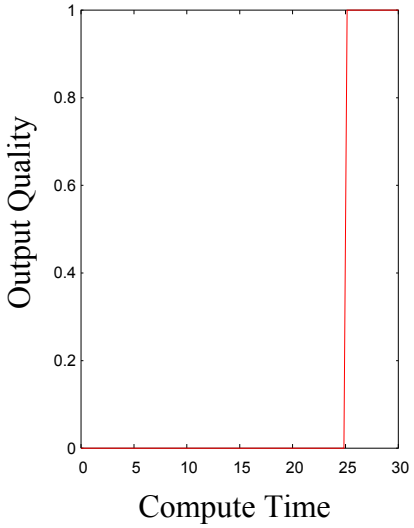
- Set of challenges I
- Idle time T
  - time between prior solved challenge and next challenge
- Pre-computation time $t_{p,i} >= 0$
  - amount of idle time allocated to challenge $i \in I$

$$\sum_{i \in I} t_{p,i} \leq T$$

  - Equivalently, as a *portfolio*, $t^f_i = t_{p,i}/T$
- Distribution over future challenges given evidence at the release date ***P(I/E)***

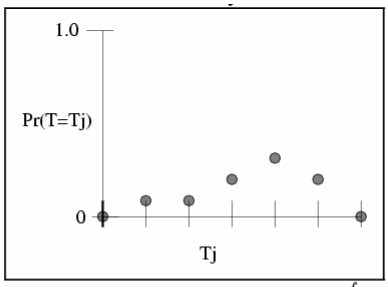
First Problem: All-or-Nothing

- Additional assumption: traditional algorithms (also called *run-to-completion*)
- $t(I_i)$ is fixed time to complete computation for challenge I_i
- Optionally: Intrinsic utility is constant times output quality



Minimizing Expected Delay

- Uncertain idle time



Solution allocates fraction of idle time $0 \leq t_i^f \leq 1$ to each challenge I_i, $\Sigma_i t_i^f = 1$.
Goal: Minimize expected delay in responding to next challenge

$$\sum_j \Pr(T = T_j) \sum_i \Pr(I_i|E)[t(I_i) - T_j t_i^f]$$

Algorithm

1. Schedule in decreasing order of future challenge probability *P(I_i/E)*
2. Compute to completion
3. Repeat while there is still idle time remaining

- In [Horvitz 2001]
- Works on-line, independent of T

- Proof Idea: Taking any processing away from most likely challenge can not decrease expected delay
- In other words: the gradient of expected delay is equal to the challenge probability. The largest diminishment of expected delay is to process challenge with largest gradient.

Example

- Probability of positive aspiration outcome (.6) > negative aspiration (.4)
- Assume $t(I_j) = t(I_i) = 1$; Assume total idle period $T = 1.5$
- Solution: reason about positive outcome completely then use remaining time to reason about negative outcome
- Expected delay without precomputation,

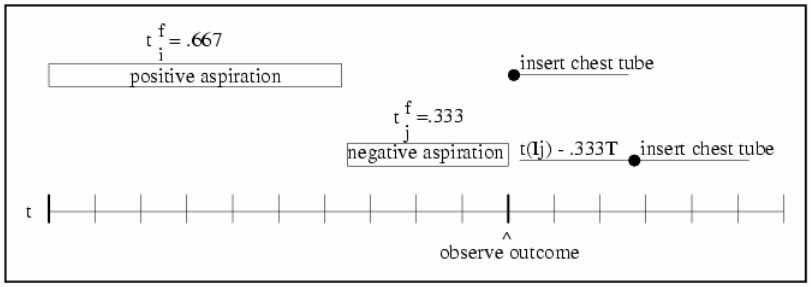
$$\sum_i \Pr(I_i|E)t(I_i) = .6 * 1 + .4 * 1 = 1$$

- Expected delay with precomputation,

$$\sum_i \Pr(I_i|E)[t(I_i) - Tt_i^f]$$

$$= .6*(1-(1.5*.667))+.4*(1-(1.5*.333)) = 0+.4*.5 = .2$$

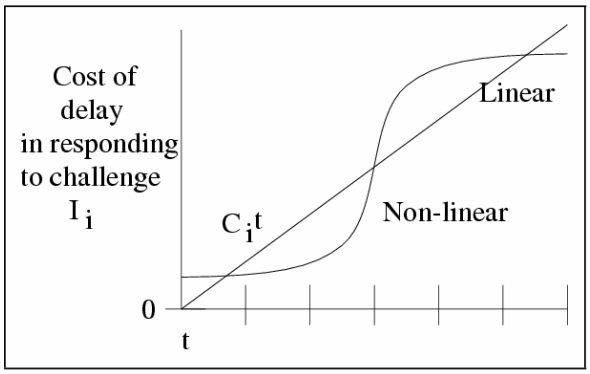
Solution Portfolio



Minimizing Expected Cost Of Delay

- Still All-or-Nothing algorithms
- Now each challenge has a time cost that begins at challenge point (deadline)
  - New model: $\Sigma_i \Pr(I_i|E)\text{Cost}(I_i, t(I_i))$
  - Applying idle time T to challenge i reduces total expected cost by $\Pr(I_i|E)[\text{Cost}(I_i, t(I_i)) - \text{Cost}(I_i, t(I_i) - T)]$

Minimizing Expected Cost of Delay



Linear cost profile result: [Horvitz 2001]

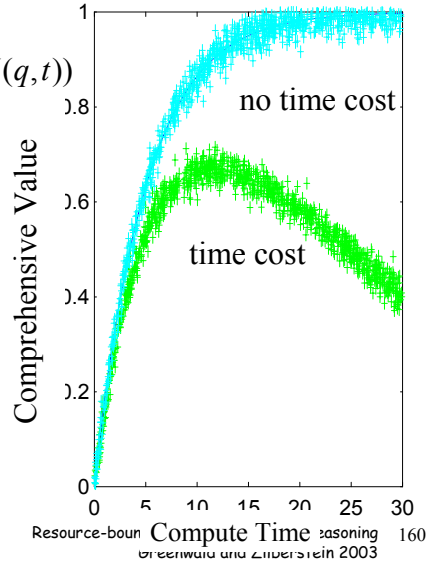
Define *unit cost of time* as C_i, for linear cost profile. Schedule computations in order of decreasing $\Pr(I_i|E) * C_i$; compute to completion.

Gradient of cost of delay is constant

Recall: Myopic Expected Value Of Computation

$$\Delta Value(q,t,t+\Delta t) = \sum_{q'} \Pr(q' \mid q,t,t+\Delta t)(U(q',t+\Delta t) - U(q,t))$$

- Flexible algorithm has already produced quality level q after t units of computation
- Reasoning about spending Δt more time computing
- U includes intrinsic utility of acting based on q and, optionally, time cost
- Does t or Δt computation incur time cost?

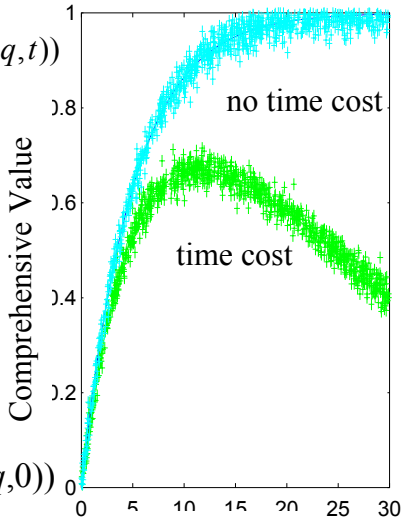


Zero Cost Pre-computation

$$\Delta Value(q,t,t+\Delta t) = \sum_{q'} \Pr(q' \mid q,t,t+\Delta t)(U(q',t+\Delta t) - U(q,t))$$

- If pre-computation incurs no time cost
- Assume cost-free pre-computation for t units
- Flexible algorithm has already produced quality level q after t units of computation
- Marginal myopic expected value of spending Δt more time in pre-computation

$$\Delta Value(q,t,t+\Delta t) = \sum_{q'} \Pr(q' \mid q,t,t+\Delta t)(U(q',0) - U(q,0))$$



Reasoning About Pre-computation

- Three options for evaluating next unit of pre-computation
  1. Pre-computation incurs no cost – [Boddy and Dean 94]

$$\Delta Value(q,t,t+\Delta t) = \sum_{q'} \Pr(q' \mid q,t,t+\Delta t)(U(q',0) - U(q,0))$$

  2. Pre-computation accumulates cost for entire duration – [Horvitz 97]

$$\Delta Value(q,t,t+\Delta t) = \sum_{q'} \Pr(q' \mid q,t,t+\Delta t)(U(q',t+\Delta t) - U(q,t))$$

  3. Pre-computation incurs no cost until challenge is observed – *soft deadline* – [Parkes and Greenwald 2001, Horvitz 2001]

Expected Value Of Pre-computation

- Expected value of challenge *i* given allocation $t_{p,i}$
  - Zero cost $Value_i(t_{p,i}) = \sum_{q'} \Pr_i(q' \mid t_{p,i}) U_i(q', 0)$
  - Non-zero cost $Value_i(t_{p,i}) = \sum_{q'} \Pr_i(q' \mid t_{p,i}) U_i(q', t_{p,i})$
- Probability of challenge I_i

$$\Pr(I_i \mid E)$$

- Expected value of pre-computation

$$EVP_i(t_{p,i}) = \Pr(I_i \mid E) Value_i(t_{p,i})$$

Portfolios

- Consider some current allocation of idle time T
- Pre-computation time $t_{p,i} >= 0$
  - amount of idle time allocated to challenge $i \in I$
- *Portfolio,* t_p

$$\forall_{i \in I} t_{p,i}, s.t. \sum_{i \in I} t_{p,i} \leq T$$

- Equivalently $t^f_i = t_{p,i} / T$

Expected Value Of Portfolio

- Given portfolio $\mathbf{t_p}$
- Probability distribution over challenges I

$$\mathbf{Pr(I \mid E)}$$

- Total expected value of portfolio

$$EVP(\mathbf{t_p}) = \sum_{i \in I} \Pr(I_i \mid E) Value_i\left(t_{p,i}\right)$$

- Re-expressed in terms of marginal values (*flux*)

$$EVP(\mathbf{t_p}) = \sum_{i \in I} \sum_{t=0}^{t_{p,i}-\Delta t} \Pr(I_i \mid E) \Delta Value_i\left(q_i, t, t+\Delta t\right)$$

Greedy Time-slicing For Portfolio Construction

- Find portfolio $\mathbf{t_p}$, that maximizes total expected value

$$EVP(\mathbf{t_p}) = \sum_{i \in I} \sum_{t=0}^{t_{p,i}-\Delta t} \Pr(I_i \mid E) \Delta Value_i\left(q_i, t, t+\Delta t\right)$$

- For any pre-computation time T
- Assume challenges includes *null* challenge with Pr(*null*|E)=0
- Iterative meta-level control algorithm

For all i, $q_i = 0$, $t_{p,i} = 0$
While (challenge not received)
  $dp = argmax_i\ (Pr(I_i|E) * \Delta Value_i(q_i, t_{p,i}, t_{p,i}+\Delta t))$
  Execute decision procedure for challenge dp for Δt time
  Observe q_{dp}
  $t_{p,dp} = t_{p,dp} + \Delta t$

(adapted from Horvitz 2001, Parkes and Greenwald 2001)

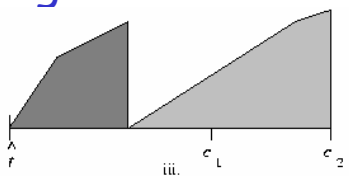
Greedy Time-slicing Analysis

- *Greedy on-line search*
  - Works forward in time – can be applied on-line using observed qualities rather than model
  - Does not use knowledge of T
  - Stops at first local maxima for each challenge
  - Globally optimal if all value functions show diminishing returns
- If T is known we can use off-line hill climbing with models to find optimal portfolio
- Extensions [Parkes and Greenwald 2001]: off-line/on-line approximation method for
  1. non-diminishing returns
  2. approximate models
  3. risk minimization objective (mean/variance tradeoff)
  4. soft deadlines

Sequential Challenges

- Solve *time-dependent planning problem* using *decision-theoretic deliberation scheduling* [Boddy and Dean 94]
  - Meta-reasoning method
  - Schedule decision procedures (anytime algorithms) for a *sequence* of independent challenges – here called *events*
  - Real-time response requirements
- Similar solution approach, exploit properties of:
  - monotonically non-decreasing expected performance profiles
  - independence of events
  - limited notion of time-cost (hard deadlines)

Time-dependent Planning Problem

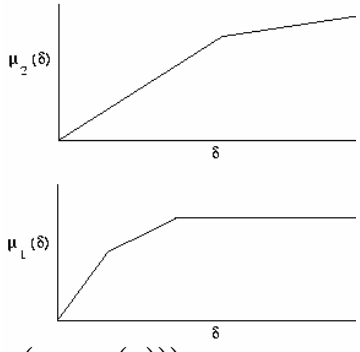


- Given:
  - list of events/conditions (C) and hard deadlines, *time(c)*,
  - exactly one decision procedure for each condition, *dp(c)*, and
  - performance profile for each decision procedure
- Allocate: time between each condition to computing responses to all conditions
- No benefit for responding before deadline
- If responses are independent, maximize:

$$\sum_{c \in C} \mathrm{V}(\mathrm{Response}(\mathrm{c}))$$

(adapted from Boddy and Dean 94)

Time-dependent Planning Problem (continued)



- Set of anytime decision procedures, one per condition
  - Monotonically increasing expected value
  - Interruptible
  - Answer always available
- Performance profiles map to following function for each condition

$$\mu_c(\delta) = E(\mathrm{V}(\mathrm{Response}(\mathrm{c}) \mid c, \mathrm{alloc}(\delta, \mathrm{dp}(c))))$$

  - Expected value of using the action resulting from deliberating for δ time
- Assume $\mu_c(\delta)$ are all
  - piecewise linear
  - with decreasing slope of consecutive segments - *diminishing returns*

(adapted from Boddy and Dean 94)

Compare To Expected Value Of Portfolio

$$EVP(\mathbf{t_p}) = \sum_{i \in I} \Pr(I_i \mid E) Value_i\left(t_{p,i}\right)$$

- Similarities:

$$t_{p,i} \approx \text{alloc}(\delta, \text{dp}(c))$$

$$\sum_{i \in I} \text{Value}_i\left(\text{t}_{\text{p,i}}\right) \approx \sum_{c \in C} E\left(\text{V}\left(\text{Response}\left(\text{c}\right) \mid c, \text{alloc}(\delta, \text{dp}(c))\right)\right)$$

- New wrinkles:
  - In pre-computation the challenges are only expected
  - In deliberation scheduling all challenges (events) will occur
  - Plus, this complication of varying deadlines

Solution: Greedy Time-slicing Algorithm

- Work *backward* from deadline of latest condition
- Choose an interval of time to allocate
- Allocate interval of time among all conditions that lie forward of start of interval
- Choose condition whose performance profile indicates the greatest *expected gain in value* for this interval

Greedy Time-slicing Algorithm Notation

- time(c) is deadline for condition c
- t is start of current allocation interval
- Set of conditions eligible for allocation

$$\Lambda(t) = \{c \mid (c \in C) \wedge (\text{time}(c) \geq t)\}$$

- Gain γ_i (x) : slope of linear segment of μ_i at x (or positive side of x, if discontinuous)
- Gain is value of more computation time, after already having computed for x time

(adapted from Boddy and Dean 94)

Choosing Next Interval min_alloc

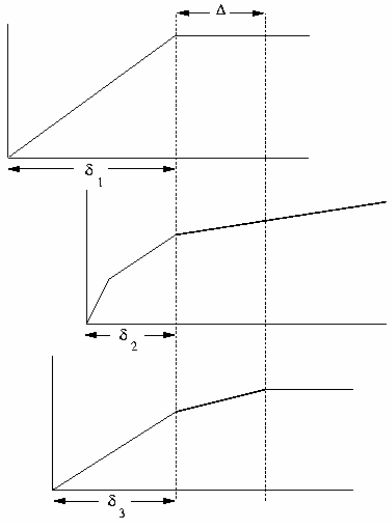


Figure 4: Determining min_alloc($\{\delta_1, \delta_2, \delta_3\}$)
(adapted from Boddy and Dean 94)

- Set of eligible conditions cannot change in the middle of the interval
- last(t) : first time point before t at which the set of eligible conditions (lambda) changes

$$\text{last}(t) = \max\{\text{time}(c) \mid c \in C - \Lambda(t)\}$$

  - i.e. can't go back any further without the set of conditions changing mid-interval
- Gain must be constant over all eligible conditions
- δ_i is time already allocated to condition i

Greedy Time-Slicing Algorithm Sketch

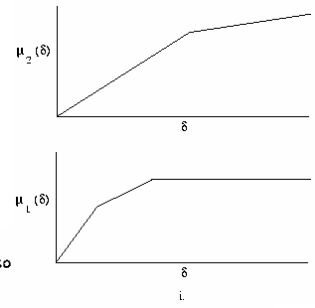
- Loop 1: initialize allocations to all conditions to zero
- Loop 2: time-slice backwards to determine allocations
- Loop 3: re-allocate to determine ordering of execution
- Assume that events/conditions/dp's are labeled by the order of deadlines (time(c))

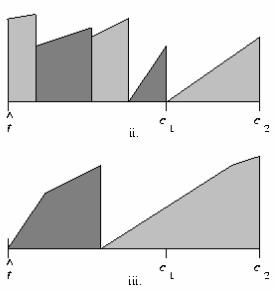
Deliberation Scheduling Algorithm

```
Procedure DS
    ;; Initialize the δ_i's to 0.
    for i = 1 to n,
        δ_i ← 0
    ;; Set t to be the time of the last event in C.
    t ← last(+∞)
    ;; Allocate time working backward from the last event.
    until t = t̂,
        ;; Compute the amount of time to allocate next.
        Δ ← min{t − t̂, t − last(t), min_alloc({δ_i})}
        ;; Find the procedure index with the maximum gain.
        i ← arg_i max{γ_i(δ_i) | c_i ∈ A(t)}
        ;; Allocate the time to the appropriate procedure.
        δ_i ← δ_i + Δ
        ;; Decrement time by the amount of allocated time.
        t ← t − Δ
    ;; Set t to be the current time.
    t ← t̂
    ;; Schedule working forwards in contiguous segments.
    for i = 1 to n,
        ;; Assume that time(c_i) ≤ time(c_j) for all i < j.
        run the ith decision procedure from t til t + δ_i
        ;; Increment time by the amount of allocated time.
        t ← t + δ_i
```

(adapted from Boddy and Dean 94)

Example Of DS






```
Procedure DS
    ;; Initialize the δi's to
    for i = 1 to n,
        δi ← 0
    ;; Set t to be the time of the last event in C.
    t ← last(+∞)
    ;; Allocate time working backward from the last event.
    until t = t̂,
        ;; Compute the amount of time to allocate next.
        Δ ← min{t − t̂, t − last(t), min_alloc({δi})}
        ;; Find the procedure index with the maximum gain.
        i ← arg_i max{γi(δi) | ci ∈ Λ(t)}
        ;; Allocate the time to the appropriate procedure.
        δi ← δi + Δ
        ;; Decrement time by the amount of allocated time.
        t ← t − Δ
    ;; Set t to be the current time.
    t ← t̂
    ;; Schedule working forwards in contiguous segments.
    for i = 1 to n,
        ;; Assume that time(ci) ≤ time(cj) for all i < j.
        run the ith decision procedure from t til t + δi
        ;; Increment time by the amount of allocated time.
        t ← t + δi
```

(adapted from Boddy and Dean 94)

- 2 conditions
- Deadlines c1 and c2
- Performance profiles given

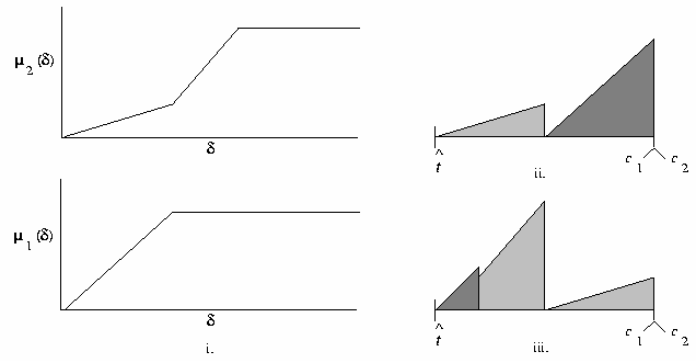
Optimality of DS

- Theorem 1 [Boddy and Dean 94]: DS is optimal in the sense that it generates a set of allocates $\{\delta_c\}$ maximizing

$$\sum_{c \in C} E\left(\mathrm{V}\left(\mathrm{Response}(\mathrm{c}) \mid c, \mathrm{alloc}(\delta, \mathrm{dp}(c))\right)\right)$$

- Proof idea: relies on decreasing slope of consecutive line segments – *diminishing returns*
- Note that DS only produces legal schedules
  - All allocation of time to c occurs before time(c)
  - Total time allocation to conditions in an interval does not exceed length of interval
  - Procedures are scheduled in order of deadline

DS Doesn't Work For Non-concave Profiles



- ii. Allocation produced by DS
- iii. Optimal allocation

Cannot allocate greedily without considering future

(adapted from Boddy and Dean 94)

Deliberation Scheduling Extensions

- Uncertain deadlines [Boddy and Dean 94]
- Continuous profiles
- Mean/variance portfolios, approximations for non-linear profiles, and soft deadlines using construction of [Parkes and Greenwald 2001]

Outline

- Introduction to meta-level control of computation
- Algorithm design, performance modeling and prediction
- Algorithm composition and transformations
- Meta-reasoning and deliberation scheduling
- Sample applications
- Current research directions and wrap-up

Sample Applications

Some Applications

- Mars rover scheduling [Bresina et. al. 99]
- Telescope scheduling [Drummond et. al. 94]
- Information gathering [Grass and Zilberstein 2000] [Lesser et. al. 2000]
- Avionics scheduling [Greenwald and Dean 98]
- Bayesian network inference [Guo and Hsu 2002] [Ramos et. al. 2002]
- Heuristic search [Hansen et. al. 97] [Pemberton and Korf 94]
- Medical decision-making [Horvitz and Rutledge 91]
- Graphics [Horvitz and Lengyel 97]
- Web pre-fetching [Horvitz 2001]
- Robot mapping and navigation [Kwok et. al. 2002]
- Agent bargaining [Larson and Sandholm 2000]
- Network congestion control [Millan-Lopez et. al. 94]
- Satellite scheduling [Pemberton and Greenwald 2002]
- Game search [Russell and Wefald 91]
- Data analysis [Smyth and Wolpert 97]
- And many more …

Example: Anytime A*

- A* is best first search with $f(n) = g(n) + h(n)$
- Three simple changes make it an anytime algorithm:
  (1) Use a non-admissible heuristic so that sub-optimal solutions are found quickly.
  (2) Continue the search after the first solution is found using it to prune the open list.
  (3) When the open list is empty, the last solution generated is optimal.
- How to choose a non-admissible heuristic?

(adapted from Hansen, Zhou, and Zilberstein 2003)

Weighted Evaluation Function

- Use $f(n) = g(n) + w * h(n)$
- Higher weight on h(n) tends to search deeper.
- Admissible if h(n) is admissible and $w \leq 1.0$
- Otherwise, the search is non-admissible, but it normally finds solutions much faster.
- The technique applies to a wide range of heuristic algorithms (e.g. A*, AO*)
- In [Pohl, 1970] [Kool and Kaindl, 1992]

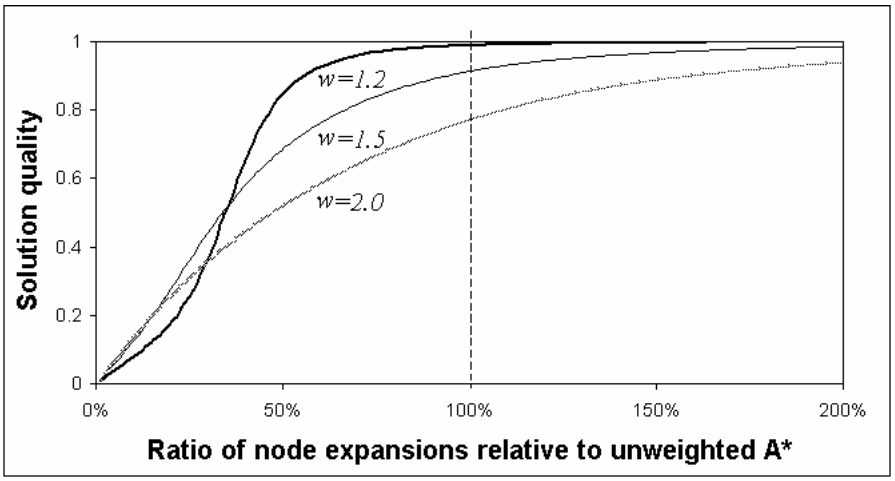
Pseudocode Of Anytime WA*

```
1  g(s) ← 0,  f'(s) ← g(s) + w × h(s)
2  OPEN ← {s}, CLOSED ← ∅, bound ← ∞
3  while OPEN ≠ ∅ do
4        n ← arg min_x{f'(x) | x ∈ OPEN}
5        OPEN ← OPEN \ {n}
6        CLOSED ← CLOSED ∪ {n}
7        if n is a goal node then
8            bound ← g(n) + h(n)
9            output solution and bound
10           for each x ∈ OPEN, g(x) + h(x) ≥ bound do
11               OPEN ← OPEN \ {x}
12       else for each n_i ∈ {x | x ∈ Successors(n),
                        g(n) + c(n, x) + h(x) < bound} do
13           if n_i ∉ OPEN ∪ CLOSED or
                        g(n_i) > g(n) + c(n, n_i) then
14               g(n_i) ← g(n) + c(n, n_i)
15               f'(n_i) ← g(n_i) + w × h(n_i)
16               OPEN ← OPEN ∪ {n_i}
17               if n_i ∈ CLOSED then
18                   CLOSED ← CLOSED \ {n_i}
```

Optimality Of Anytime WA*

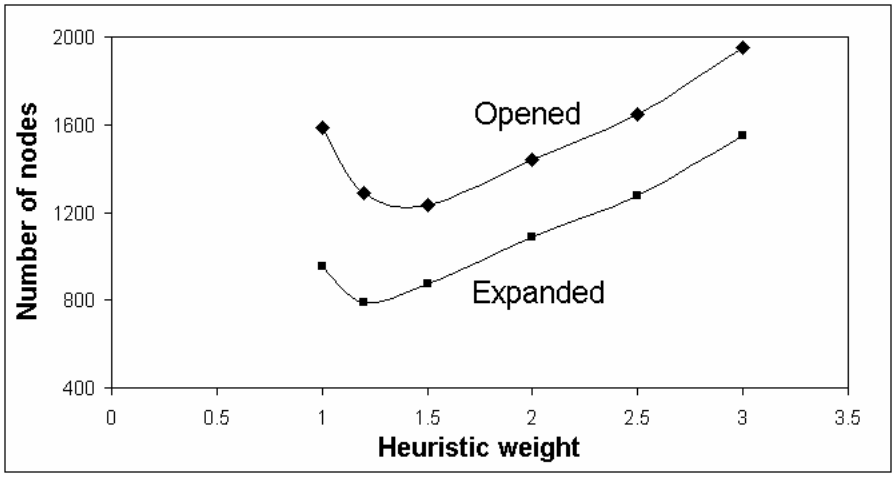
- **Theorem**: Anytime WA* always terminates and the last solution it produces is an optimal solution

Performance Analysis Of AWA*



Performance profiles using three different weights, averaged over all instances of the 8-puzzle.

Performance Analysis Of AWA*



Average number of nodes opened and expanded by AWA* over all instances of the 8-puzzle

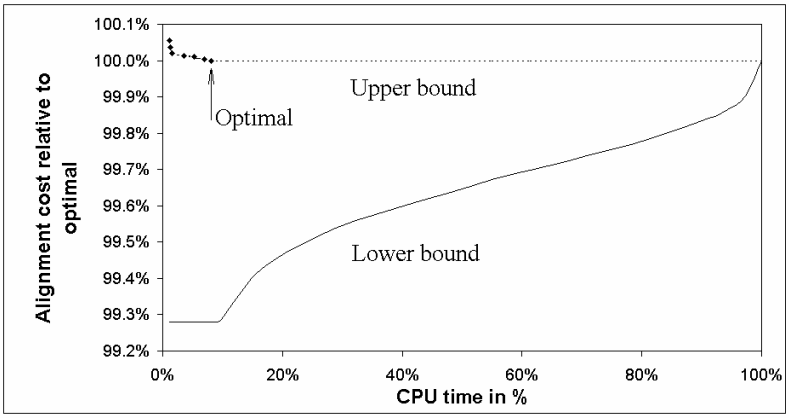
Domain-independent Planning

	A*		AWA*	
Instance	Nodes	CPU secs.	Nodes	CPU secs.
Eight-1	47,275	66.2	32,074	39.6
Blocks-8	510,594	17778.8	44,570	117.0
Hanoi-3	51,032	40.0	42,067	34.1
Gripper-8	233,702	380.7	217,065	348.9
Mystery-30	24,004	29.2	2,994	22.5

Comparison of A* and AWA* on five benchmark problems from the biennial Planning Competitions. The Nodes column gives the total number of nodes stored

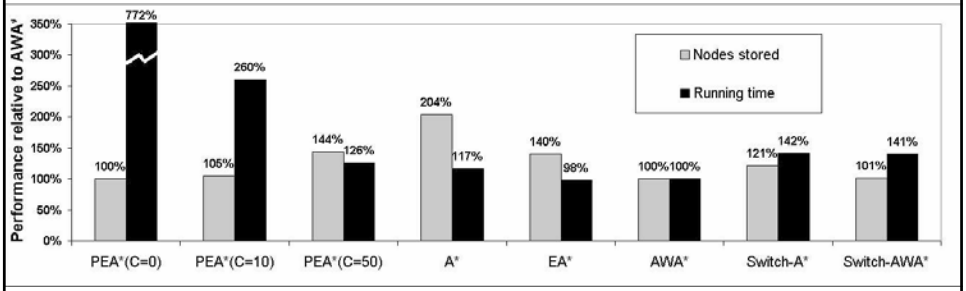
[Experiments were performed on an UltraSparc II with a 300 Mhz CPU and two gigabytes of RAM]

Multiple Sequence Alignment



Convergence of bounds for Anytime WA* in aligning sets of 5 sequences from [Kobayashi and Imai 1998].

Comparison Of Search Algorithms



Average performance of search algorithms in aligning sets of 5 sequences from [Kobayashi and Imai 1998]

Outline

- Introduction to meta-level control of computation
- Algorithm design, performance modeling and prediction
- Algorithm composition and transformations
- Meta-reasoning and deliberation scheduling
- Sample applications
- Current research directions and wrap-up

Current Research Directions And Wrap-up

Progress: Satisficing versus Optimizing

Problem:

"Currently, ad hoc techniques are used for making a system produce a response within a specified time interval."

-- Laffey *et al., AI Magazine,* 1988

Solution:

Many techniques are available now to optimize the quality of the response, net of computational costs.

Progress: Optimal Composition

Problem:

"There is currently no general theory of combining anytime algorithms. For cases in which the decision problems are dependent, there is not a great deal that we can say."

-- Dean and Wellman, *Planning and Control,* 1991

Solution:

Local compilation techniques allow for optimal composition of anytime algorithms for a class of programming constructs.

Progress: Performance Guarantees

Problem:

Anytime algorithms "ensure that some result will be available by a deadline. However, the quality or correctness of that result cannot be guaranteed."

-- Musliner *et al., Artificial Intelligence,* 1995

Solution:

With more informative performance profiles and active monitoring it is possible to guarantee a minimal quality level.

Some Unifying Themes

- Greedy on-line time-slicing algorithms apply to many problem variations, with similar assumptions of diminishing returns
  - Optimal stopping
  - Monitoring
  - Continual computation
  - Deliberation scheduling
- Non-diminishing returns can often be formulated as Markov Decision Problems (MDPs) and solved off-line with dynamic programming

Research Directions

- What properties of resource-bounded reasoning algorithms make them useful and easy to control?
- How important are properties of monotonicity in quality, convergence on optimal results, interruptibility, and diminishing returns?
- How can we best identify and take advantage of dependencies between memory, time, challenges, and informational resources?
- What representations of knowledge allow for efficient implementation of flexible inference strategies?
- What is an ideal representation for performance profiles?
- How can we ideally partition resources between meta-level and object-level reasoning?
- How can we best estimate the value of partial results and predict the outcome of allocating additional amounts of resources?
- How might unsupervised learning and data mining be used to acquire knowledge about problem-solving performance and control?
- How might exploration strategies be applied within on-line model-free approaches?

Acknowledgements

This tutorial consists of original material and material adapted from the following sources:

D.S. Bernstein, T.J. Perkins, S. Zilberstein, and L. Finkelstein. Scheduling contract algorithms on multiple processors. *Proceedings of the 18th National Conference on Artificial Intelligence*, 702-706, Edmonton, Alberta, 2002.

M. Boddy and T. Dean. Decision-theoretic deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245-286, 1994.

J. Doyle. Bounded rationality. In MIT Encyclopedia of the Cognitive Sciences, Cambridge: MIT Press, 1999.

J. Grass and S. Zilberstein. Anytime algorithm development tools. In M. Pittarelli (Ed.), *SIGART Bulletin Special Issue on Anytime Algorithms and Deliberation Scheduling*, 7(2):20-27, 1996.

E. Hansen and S. Zilberstein. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence*, 126(1-2):139-158, 2001.

E. Horvitz. Reasoning about Beliefs and Actions under Computational Resource Constraints. KSL Tech Report. 1987.

E. Horvitz. Principles and applications of continual computation. *Artificial Intelligence*, 126(1-2):159-196, 2001.

D. Parkes and L. Greenwald. Approximate and compensate: A method for risk-sensitive meta-deliberation and continual computation. *AAAI Fall Symposium on Using Uncertainty in Computation*, Cape Cod, Massachusetts, 2001.

S. Russell. Rationality and intelligence. *Artificial Intelligence*, 94(1):57-77, 1997.

S. Zilberstein. Using anytime algorithms in intelligent systems. *Artificial Intelligence Magazine*, 17(3):73-83, 1996.

S. Zilberstein and S. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1-2):181-213, 1996.

Bibliography

- D.S. Bernstein, T.J. Perkins, S. Zilberstein, and L. Finkelstein. Scheduling contract algorithms on multiple processors. *Proceedings of the 18th National Conference on Artificial Intelligence*, 702-706, Edmonton, Alberta, 2002.
- J. Bresina, K. Golden, D.E. Smith, and R. Washington. Increased flexibility and robustness for Mars rovers. *Fifth International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, ESTEC, Noordwijk, Netherlands, 1999.
- M. Boddy and T. Dean. Decision-theoretic deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245-286, 1994.
- V. Bultiko, I. levner, and R. Greiner. Real-time lookahead control policies. *AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems*, 28-36, Edmonton, Alberta, 2002.
- P. Chakrabarti, S. Ghosh, and A. Acharya. Heuristic search in restricted memory. *Artificial Intelligence*, 47:197-221, 1989.
- A. Darwiche. Any-space probabilistic inference. *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, 133-142, Stanford, California, 2000.
- T. Dean and M. Boddy. An analysis of time-dependent planning. *Proceedings of the 7th National Conference on Artificial Intelligence*, 49-54, Minneapolis, Minnesota, 1988.
- T. Dean, Context-dependent computational components, In M. Pittarelli (Ed.), *SIGART Bulletin Special Issue on Anytime Algorithms and Deliberation Scheduling*, 7(2):3-6, 1996.
- T. Dean, L. Kaelbling, J. Kirman, and A. Nicholson. Planning with deadlines in stochastic domains. *Proceedings of the 11th National Conference on Artificial Intelligence*, 574-579, Washington, D.C., 1993.
- S. de Givry and G. Verfaillie. Optimum anytime bounding for constraint optimization problems. *AAAI Workshop on Building Resource-Bounded Reasoning Systems*, 37-42, Providence, Rhode Island, 1997.
- A. Delhay, M. Dauchet, P. Taillibert, and P. Vanheeghe. Maximization of the average quality of anytime contract algorithms over a time interval. *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 212-217, Stockholm, Sweden, 1999.
- J. Doyle. Rationality and its roles in reasoning. *Proceedings of the 8th National Conference on Artificial Intelligence*, 1093-1100, Boston, Massachusetts, 1990.
- J. Doyle. Bounded rationality. In MIT Encyclopedia of the Cognitive Sciences, Cambridge: MIT Press, 1999.
- D.H. Dash and M.J. Druzdzel. A hybrid anytime algorithm for the construction of causal models from sparse data. *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence*, 142-149, San Francisco, CA, 1999.
- M. Drummond and J. Bresina. Anytime synthetic projection: Maximizing the probability of goal satisfaction. *Proceedings of the National Conference on Artificial Intelligence*, 138-144, Boston, MA, 1990.
- M. Drummond, J. Bresina, and K. Swanson. Just-in-case scheduling. *Proceedings of the Twelth National Conference on Artificial Intelligence*, 1098-1104, Seattle, WA, 1994.
- Oren Etzioni. *Tractable decision-analytic control.* In First International Conference on Principles of Knowledge Representation and Reasoning, pages 114- 125. Morgan-Kaufmann, 1989.

Bibliography

- L. Finkelstein and S. Markovitch. Optimal schedules for monitoring anytime algorithms. Artificial Intelligence, 126(1-2):63-108, 2001.
- L. Finkelstein, S. Markovitch, and E. Rivlin. Optimal schedules for parallelizing anytime algorithms: The case of independent processes. Proceedings of the 18th National Confernce on Artificial Intelligence, 719-724, Edmonton, Alberta, 2002.
- A. Garvey and V. Lesser. Design-to-time real-time scheduling. IEEE Transactions on Systems, Man and Cybernetics, 23(6):1491-1502, 1993.
- A. Garvey and V. Lesser. A survey of research in deliberative real-time artificial intelligence. Journal of Real-Time Systems, 6(3):317-347, 1994.
- S. Ghosh, A. Mahanti, and D. Nau. BDFS: A resource bounded search procedure for combinatorial optimization problems. AAAI Fall Symposium on Flexible Computation in Intelligent Systems, 44-48, Boston, Massachusetts, 1996.
- C.P. Gomes and B. Selman. Algorithm portfolio design: Theory vs. practice. Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence, San Francisco, California, 1997.
- I. Good. Twenty-seven principles of rationality. In Godambe, V. P., and Sprott, D. A. (Eds.) Foundations of Statistical Inference, Toronto: Holt, Rinehart, Winston, 1971.
- J. Grass and S. Zilberstein. A value-driven system for autonomous information gathering. Journal of Intelligent Information Systems, 14:5-27, 2000.
- J. Grass and S. Zilberstein. Anytime algorithm development tools. In M. Pittarelli (Ed.), SIGART Bulletin Special Issue on Anytime Algorithms and Deliberation Scheduling, 7(2):20-27, 1996.
- L. Greenwald and T. Dean. Solving time-critical decision-making problems with predictable computational demands. Proceedings of the Second International Conference on AI Planning Systems, 1994.
- L. Greenwald and T. Dean. A conditional scheduling approach to designing real-time systems. Proceedings of the Fourth International Conference on AI Planning Systems, 1998.
- H. Guo and W.H. Hsu. A survey of algorithms for real-time bayesian network inference. AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems, 1-12, Edmonton, Alberta, 2002.
- P. Haddawy. Focusing attention in anytime decision-theoretic planning. In M. Pittarelli (Ed.), SIGART Bulletin Special Issue on Anytime Algorithms and Deliberation Scheduling, 7(2):34-40, 1996.
- E. Hansen, A. Barto, and S. Zilberstein. Reinforcement learning for mixed open-loop and closed-loop control. Proceedings of Neural Information Processing Systems Conference, 1026-1032, Denver, Colorado, 1996.
- E. Hansen and S. Zilberstein. Monitoring the progress of anytime problem solving. Proceedings of the 13th National Conference on Artificial Intelligence, 1229-1234, Portland, Oregon, 1996.
- E. Hansen, S. Zilberstein, and V. Danilchenko. Anytime heuristic search: First results. University of Massachusetts, Computer Science Department, Technical Report #97-50, 1997.
- E. Hansen and S. Zilberstein. Monitoring and control of anytime algorithms: A dynamic programming approach. Artificial Intelligence, 126(1-2):139-158, 2001.

Bibliography

- O. Hansson and A. Mayer. Heuristic search as evidential reasoning. Proceedings of the 5th Workshop on Uncertainty in Artificial Intelligence, Windsor, Ontario, 1989.
- O. Hansson and A. Mayer. DTS: A decision-theoretic scheduler for space telescope applications. In M. Fox and M. Zweben (eds.), Intelligent Scheduling, Chapter 13, 371-388, Morgan Kaufmann, 1994.
- R. Holte, M. Perez, R. Zimmer, and A. MacDonald. The tradeoff between speed and optimality in hierarchical search. AAAI Fall Symposium on Flexible Computation in Intelligent Systems, 60-67, Boston, Massachusetts, 1996.
- M.C. Horsch and D. Poole. An anytime algorithm for decision making under uncertainty. Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, 246-255, Madison, Wisconsin, 1998.
- M.C. Horsch and D. Poole. Estimating the value of computation in flexible information refinement. Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence, San Francisco, California, 1999.
- E. Horvitz. Reasoning about Beliefs and Actions under Computational Resource Constraints. KSL Tech Report. 1987.
- E.J. Horvitz. Reasoning under varying and uncertain resource constraints. Proceedings of the 7th National Conference on Artificial Intelligence, 111-116, 1988.
- E.J. Horvitz, G.F. Cooper, and D.E. Heckerman. Reflection and action under scarce resources: Theoretical principles and empirical study. Proceedings of the 11th International Joint Conference on Artificial Intelligence, 1121-1127, Detroit, Michigan, 1989.
- E.J. Horvitz, H.J. Suermondt, and G.F. Cooper. Bounded conditioning: Flexible inference for decision under scarce resources. Proceedings of the 5th Workshop on Uncertainty in Artificial Intelligence, 182-193, Windsor, Ontario. 1989
- E. Horvitz and J. Breese. Ideal partition of resources for metareasoning. Technical Report KSL-90-26, Stanford Knowledge Systems Laboratory, 1990.
- E.J. Horvitz and G. Rutledge. Time-dependent utility and action under uncertainty. Proceedings of 7th Conference on Uncertainty in Artificial Intelligence, 151-158, 1991.
- E. Horvitz and J. Lengyel. Perception, attention, and resources: A decision-theoretic approach to graphics rendering. Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence, 238-249, 1997.
- E. Horvitz. Principles and applications of continual computation. Artificial Intelligence, 126(1-2):159-196, 2001.
- E. Horvitz, Y. Ruan, C. Gomes, H. Kautz, B. Selman, D. M. Chickering. A Bayesian approach to tackling hard computational problems. Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence, August 2001.
- R. Howard. Information Value Theory. IEEE Transactions on System Science and Cybernetics, SSC-2(1):22-26, 1966.
- D. Hull and W. Feng and J. W. S. Liu. Operating System Support for Imprecise Computation. AAAI Fall Symposium on Flexible Computation, 1996.
- R. Jensen and J. Schlimmer. An anytime solution to inductively learn decision trees. AAAI Fall Symposium on Flexible Computation in Intelligent Systems, 106-110, Boston, Massachusetts, 1996.

Bibliography

- N. Jitnah and A.E. Nicholson. treeNets: A framework for anytime evaluation of belief networks, ECSQARU-FAPR, 350-364, 1997.
- N. Jitnah and A.E. Nicholson. A best-first search method for anytime evaluation of belief networks. AAAI Workshop on Building Resource-Bounded Reasoning Systems, 69-73, Providence, Rhode Island, 1997.
- Ming-Yang Kao, Yuan Ma, Michael Sipser, and Yiqun Yin. *Optimal constructions of hybrid algorithms*. Journal of Algorithms, 29:142--164,1998.
- H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, B. Selman. Dynamic restart policies. Proceedings of the 18th National Conference on Artificial Intelligence, 674-681, Edmonton, Alberta, 2002.
- S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. Proceedings of the International Conference on Robotics and Automation, 2002.
- R. Korf. Real-time heuristic search. Artificial Intelligence, 42(3):189-212, 1990.
- C.T. Kwok, D. Fox, and M. Meila. Real-time particle filters using mixtures of samples sets. AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems, 2002.
- M. Lagoudakis, M. Littman, and R. Parr. Selecting the right algorithm. AAAI Fall Symposium Series: Using Uncertainty within Computation, Cape Cod, MA, 2001.
- K. Larson and T. Sandholm. Bargaining with limited computation: deliberation equilibrium. Artificial Intelligence, 132(2):183-217, 2000.
- K. Larson and T. Sandholm. An alternating offers bargaining model for computationally limited agents. Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, Bologna, Italy, 2002.
- V. Lesser, J. Pavlin and E. Durfee. Approximate processing in real-time problem-solving. Artificial Intelligence Magazine, 9(1):49-61, 1988.
- V.R. Lesser, B. Horling, F. Klassner, A. Raja, T. Wagner, and S.X.Q. Zhang. BIG: An Agent for Resource-Bounded Information Gathering and Decision Making. In Artificial Intelligence Journal, Special Issue on Internet Information Agents, 118(1-2):197-244, May 2000.
- C.-L. Liu and M. Wellman. On state-space abstraction for anytime evaluation of Bayesian networks. In M. Pittarelli (Ed.), SIGART Bulletin Special Issue on Anytime Algorithms and Deliberation Scheduling, 7(2):50-57, 1996.
- J.W.S. Liu, K.J. Lin, W.K. Shih, A.C. Yu, J.Y. Chung, and W. Zhao. Algorithms for scheduling imprecise computations, IEEE Computer, 24:58-68, 1991.
- M. Marengoni, A. Hanson, S. Zilberstein, and E. Riseman. Control in a 3D reconstruction system using selective perception. Proceedings of the 7th IEEE International Conference on Computer Vision, 1229-1236, Kerkyra, Greece, 1999.
- V. Millan-Lopez, W. Feng, and J.W.S. Liu. Using the imprecise computation technique for congestion control on a real-time traffic switching element. International Conference on Parallel and Distributed Systems, 1994.
- A.-I. Mouaddib and S. Zilberstein. Optimal scheduling of dynamic progressive processing. Proceedings of the 13th Biennial European Conference on Artificial Intelligence, 449-503, Brighton, UK, 1998.

Bibliography

- S.H. Nawab, A.V. Oppenheim, A.P. Chandrakasan, J.M. Winograd, and J.T. Ludwig. Approximate signal processing. Journal of VLSI Signal Processing, 15:177-200, 1997.
- D. Parkes and L. Greenwald. Approximate and compensate: A method for risk-sensitive meta-deliberation and continual computation. AAAI Fall Symposium on Using Uncertainty in Computation, Cape Cod, Massachusetts, 2001.
- J. Pemberton and L. Greenwald. On the need for dynamic scheduling of imaging satellites. Future Intelligent Earth Observing Satellites Symposium, 2002.
- J. Pemberton and R. Korf. Incremental search algorithms for real-time decision making. Proc. Second International Conference on AI Planning Systems,1994.
- J. Pemberton. k-best: A new method for real-time decision making. Proceedings of the 14th International Joint Conference on Artificial Intelligence, 227-233, Montreal, Canada, 1995.
- A. Pos. Time-constrained model-based diagnosis. Master Thesis, Department of Computer Science, University of Twente, The Netherlands, 1993.
- F.T. Ramos, F.G. Cozman, and J.S. Ide. Embedded Bayesian networks: Anyspace, anytime probabilistic inference. AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems, 13-19, Edmonton, Alberta, 2002.
- S. Russell. Efficient memory-bounded search methods. Proceedings of the 10th European Conference on Artificial Intelligence, 1-5, Vienna, Austria, 1992.
- S. Russell. Rationality and intelligence. Artificial Intelligence, 94(1):57-77, 1997.
- S. J. Russell, D. Subramanian, and R. Parr. *Provably bounded optimal agents.* In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93), pages 338--344, Chambery, France, 1993.
- S. Russell and D. Subramanian. Provably bounded-optimal agents. Journal of Artificial Intelligence Research, 2, 1995.
- S. Russell and E. Wefald. Do the Right Thing: Studies in limited rationality. Cambridge, Massachusetts: MIT Press, 1991.
- S. Russell and E. Wefald. Principles of metareasoning. Artificial Intelligence, 49:361-395, 1991.
- T.W. Sandholm and V.R. Lesser. Coalitions among computationally bounded agents. Artificial Intelligence, 94(1):99-137, 1997.
- H. Simon. From substantive to procedural rationality. In H. Simon, Models of Bounded Rationality, Volume 2, Cambridge, Massachusetts: MIT Press, 1982.
- P. Smyth and D. Wolpert. Anytime exploratory data analysis for massive data sets. Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, 54-60, 1997.
- L. Steinberg. Searching stochastically generated multi-abstraction-level design spaces. Artificial Intelligence, 129(1-2):63-90, 2001.

Bibliography

- K. Toyama. Handling tradeoffs between precision and robustness with incremental focus of attention for visual tracking. AAAI Fall Symposium on Flexible Computation in Intelligent Systems, 142-147, Boston, Massachusetts, 1996.
- R. Wallace and E. Freuder. Anytime algorithms for constraint satisfaction and SAT problems. In M. Pittarelli (Ed.), SIGART Bulletin Special Issue on Anytime Algorithms and Deliberation Scheduling, 7(2):7-10, 1996.
- W. Ruml. Real-time heuristic search for combinatorial optimization and constraint satisfaction. AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems, 85-90, Edmonton, Alberta, 2002.
- W. Zhang. Complete anytime beam search. Proceedings of the 15th National Conference on Artificial Intelligence, 425-430, Madison, Wisconsin, 1998.
- W. Zhang. Iterative state-space reduction for flexible computation. Artificial Intelligence, 126(1-2):109-138, 2001.
- R. Zhou and E.A. Hansen. Memory-bounded A* graph search. Fifteenth International FLAIRS Conference, Pensacola, Florida, 2002.
- S. Zilberstein. Resource-bounded sensing and planning in autonomous systems. Autonomous Robots, 3:31-48, 1996.
- S. Zilberstein. Using anytime algorithms in intelligent systems. Artificial Intelligence Magazine, 17(3):73-83, 1996.
- S. Zilberstein and S. Russell. Anytime Sensing, Planning and Action: A Practical Model for Robot Control. Proc. of the Int. Joint Conf. on Artificial Intelligence, 1993.
- S. Zilberstein and S. Russell. Optimal composition of real-time systems. Artificial Intelligence, 82(1-2):181-213, 1996.
- S. Zilberstein and A.-I. Mouaddib. Optimizing resource utilization in planetary rovers. Proceedings of the 2nd NASA International Workshop on Planning and Scheduling for Space, 163-168, San Francisco, California, 2000.
- S. Zilberstein, R. Washington, D.S. Bernstein, and A.I. Mouaddib. Decision-theoretic control of planetary rovers. To appear in Springer LNAI, 2002.
- M. Zweben, E. Davis and R. Guargan. Anytime rescheduling. Proceedings of DARPA Workshop on Innovative Approach for Planning and Scheduling, 251-259, San Francisco, 1990.