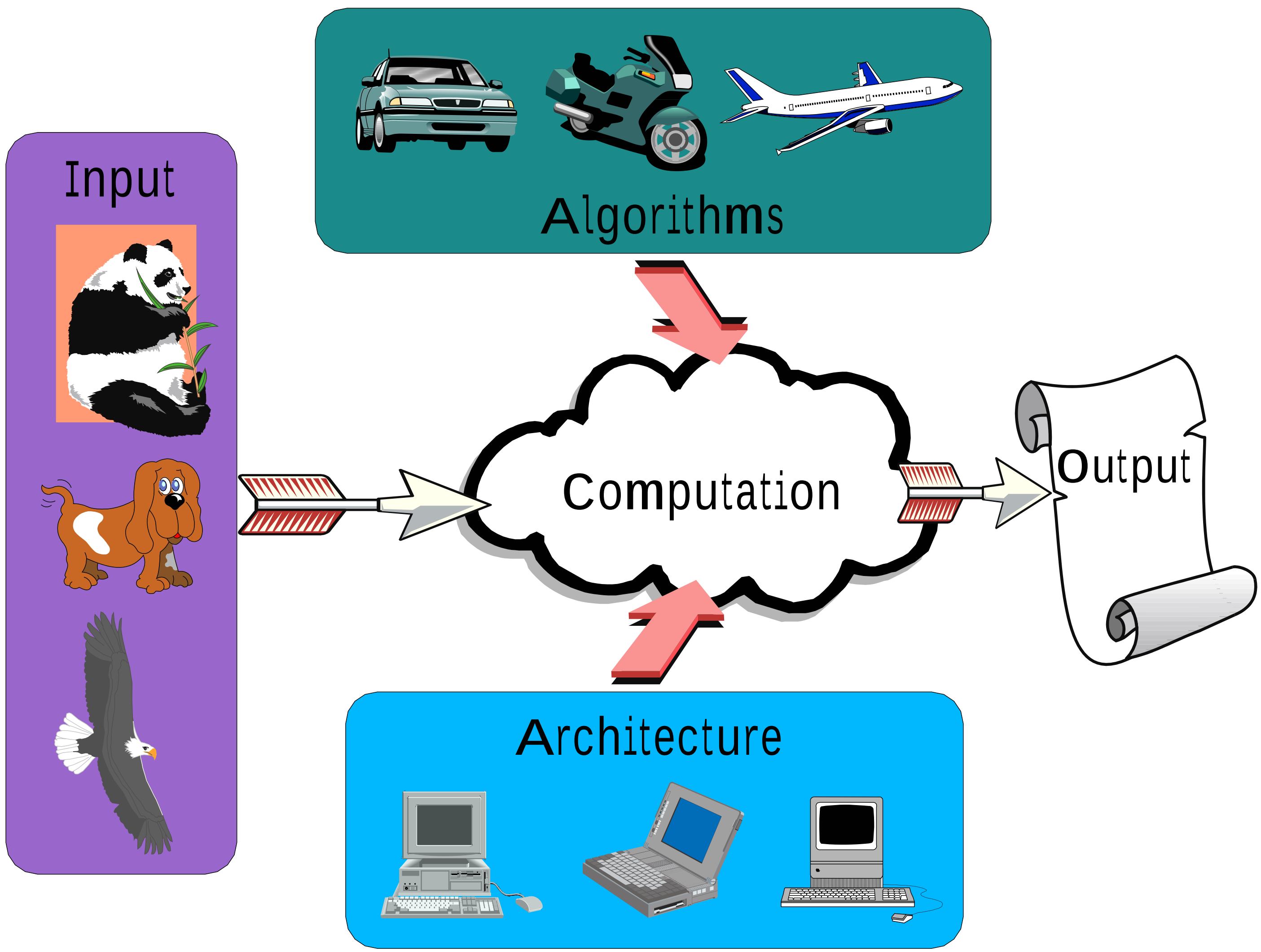
Selecting the Right Algorithm

Michail G. Lagoudakis
Department of Computer Science, Duke University, Durham, NC 27708

Michael L. Littman
Shannon Laboratory, AT&T Labs – Research, Florham Park, NJ 07932

Ronald E. Parr
Department of Computer Science, Duke University, Durham, NC 27708

Uncertainty in Computation



A Little Quiz ...

Problem

Add some numbers

Question

Which method would you choose

?

- Paper and pencil
- Calculator
- MatLab program
- Pay somebody

The Answer ...

Answer

It depends!

- How many numbers?
- How many digits per number?
- What base?
- What format?

Now, it's clear!

- 142 + 304
- 34A4C324D6 + 7C37B5E349 + 1A8582234C + 326BDFFFF4 + ...

Sorting ...

Problem

Sort some numbers

Question

Which algorithm would you choose

?

- Bubble Sort
- Insertion Sort
- Quick Sort
- Merge Sort
- Shell Sort
- Distribution Sort
- Radix Sort
- Heap Sort

Outline

- *(Recursive) Algorithm Selection*
- *Markov Decision Processes (MDPs)*
- *Algorithm Selection as an MDP*
- *Case Study: Sorting*
- *Case Study: Results*
- *Discussion*
- *Related Work*
- *Conclusion*

Algorithm Selection

Given

- A *set of algorithms* for a given problem.
- A description (*set of features*) of the current instance.

Goal

- Dynamically *select the "**right**" (fastest) algorithm* for any given instance based on its features.

Recursive Algorithm Selection

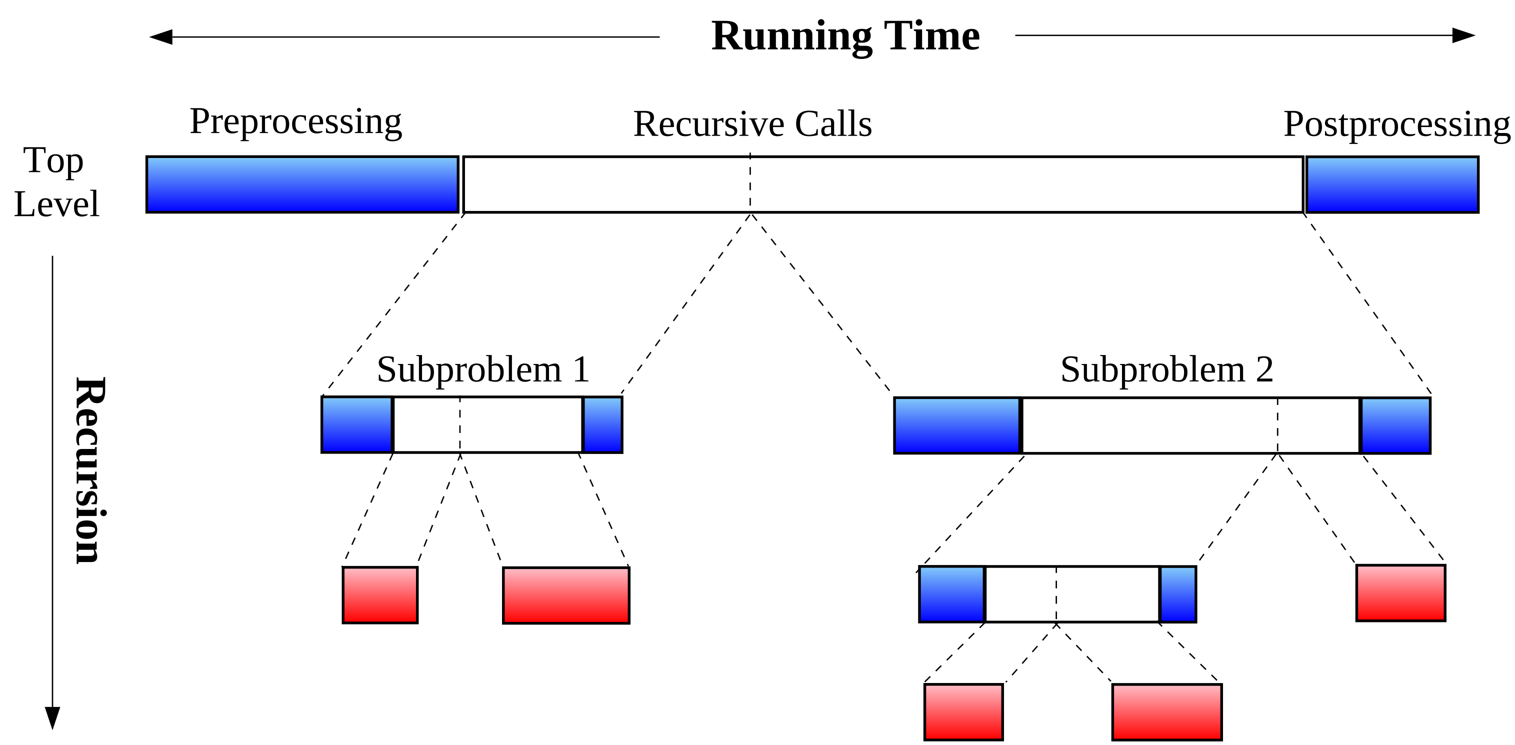
Previous Work

- Algorithms are treated as black-boxes
- *Cannot* do better than the best of the individual algorithms

Our work

- **Recursive** algorithms $\Longrightarrow$ **Multiple** algorithm selection problems
- *Any* algorithm can be selected at *each* recursive call
- Algorithm selection is viewed as a ***sequential decision task***
- Yields a hybrid algorithm!
- *Can* potentially do better than the best of the individual algorithms!

Recursive Computation



Markov Decision Processes (MDPs)

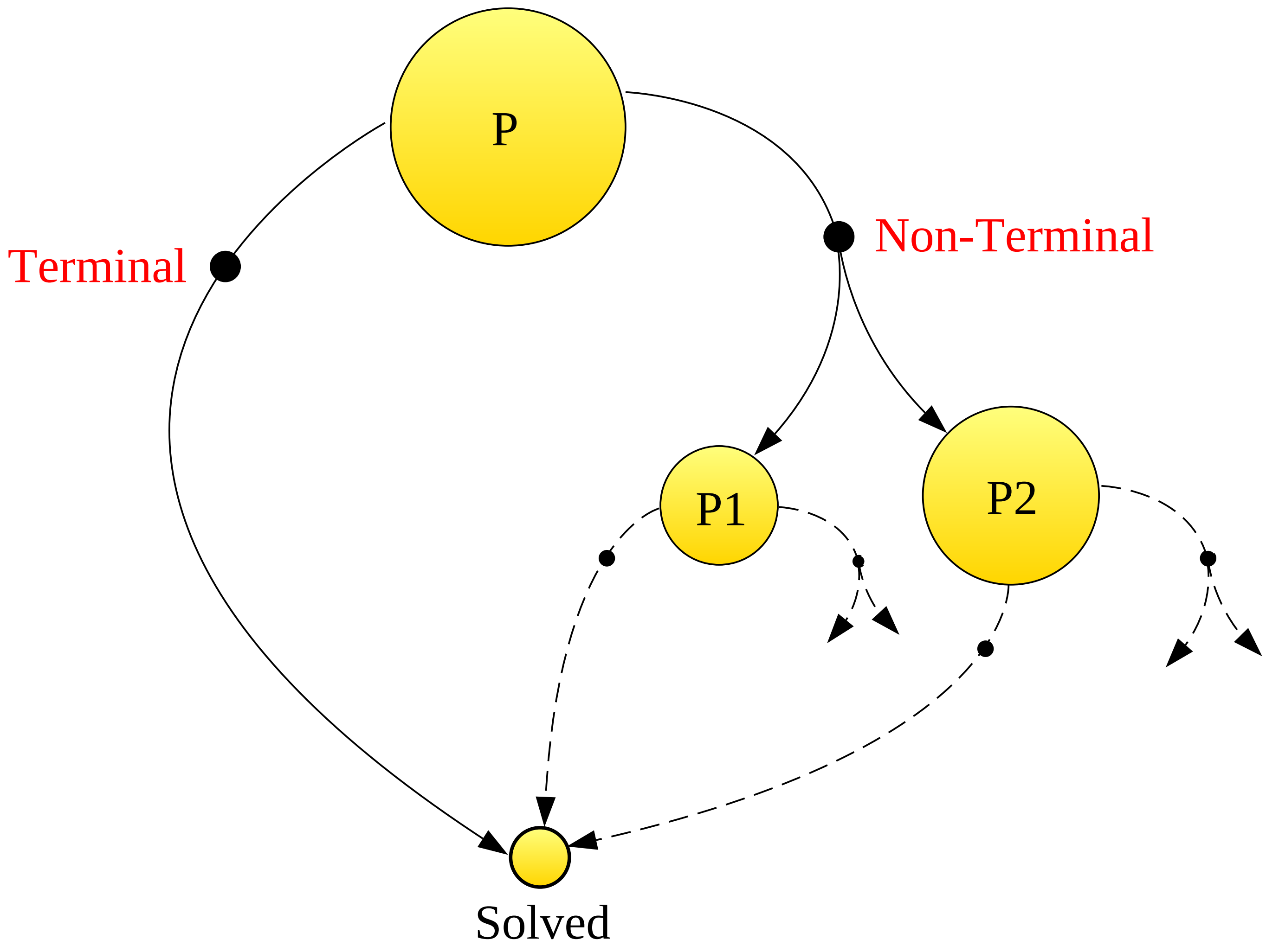
- **States** : $\mathcal{S}, \quad |\mathcal{S}| = n$
- **Actions** : $\mathcal{A}, \quad |\mathcal{A}| = m$
- **Transition Model** : $\mathcal{P}(s, a, s') = P(s'|s, a)$
- **Cost Function** : $\mathcal{R}(s, a)$
- **Episodes** : $s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} s_3 \xrightarrow[r_3]{a_3} s_4 \quad \ldots \xrightarrow[r_{N-1}]{a_{N-1}} s_N$
- **Policy** : $\pi : \mathcal{S} \mapsto \mathcal{A}$
- ***Optimal policy*** : π^* (minimizes the expected total cost):

$$\pi^* = \arg\min_{\pi} E_{\pi} \left(\sum_t r_t \right)$$

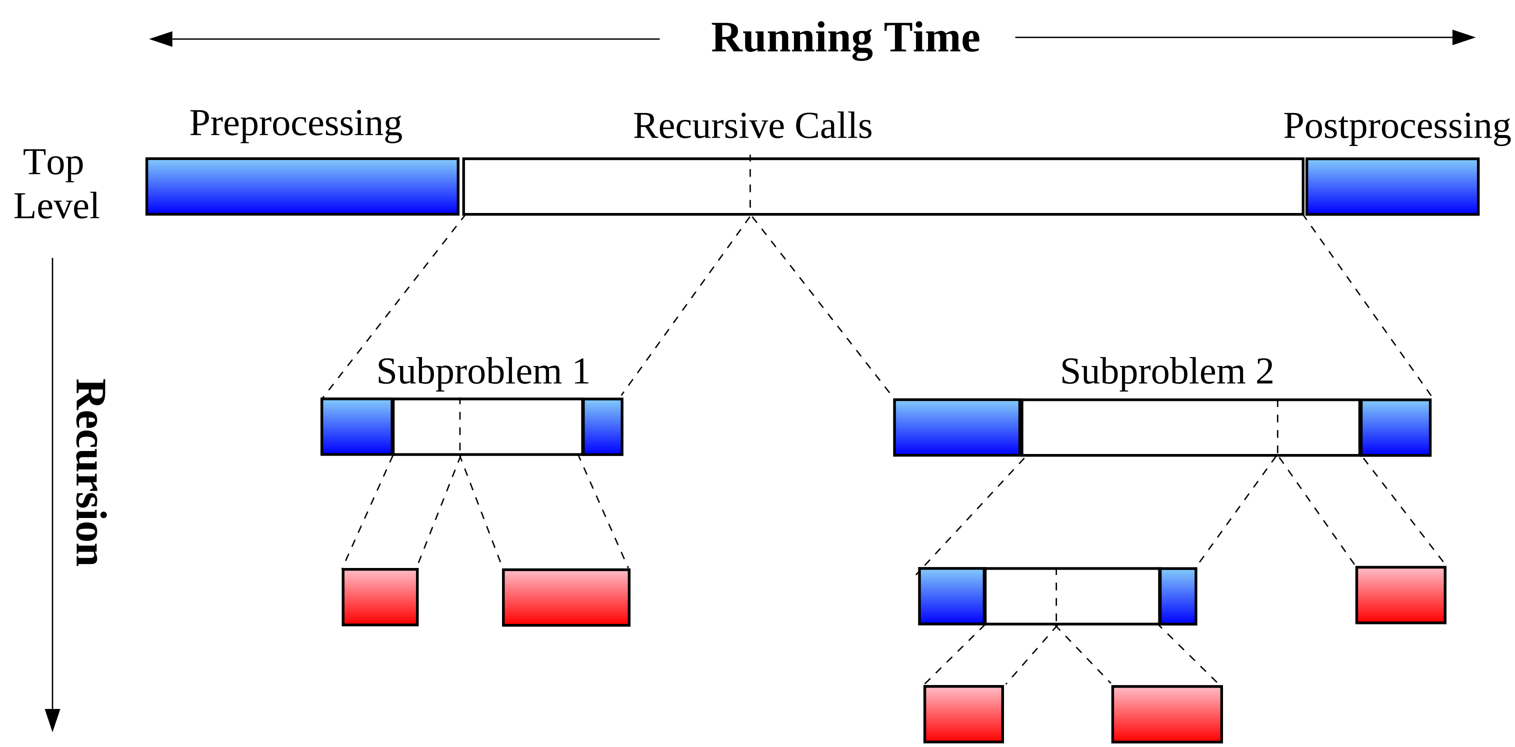
Algorithm Selection as an MDP

- ***Task***: Solve the current instance (episodic task)
- ***State***: Current instantiation of instance features
- ***Actions***: The available algorithms (terminal and non-terminal)
- ***State Transitions***: One-to-Many for non-terminal algorithms!
- ***Transition Cost***: Real time taken (excluding time in recursive calls)
- ***Total Cost***: The total execution time
- ***Objective***: A policy that minimizes the total cost

State Transitions



Time Cost


Running Time
Preprocessing
Recursive Calls
Postprocessing
Top
Level
Recursion
Subproblem 1
Subproblem 2

Case Study: Sorting

Problem

- Rearrange an array of n (unordered) numbers in ascending order.

Algorithms

- **InsertionSort** (terminal, worst-case $O(n^2)$)
- **Randomized QuickSort** (recursive, worst-case $O(n^2)$, average case $O(n \log n)$)
- **MergeSort** (recursive, worst-case $O(n \log n)$)

Sorting as an MDP

- States: Size of the array s
- Actions: The algorithms I, M, and Q.
- Transition Model? Cost Function?

Transition Model

InsertionSort (I)

$$\mathcal{P}(s, I, 1) = 1$$

MergeSort (M)

$$\mathcal{P}(s, M, \{\lfloor s/2 \rfloor, \lceil s/2 \rceil\}) = 1$$

QuickSort (Q)

$$\mathcal{P}(s, Q, \{1, s-1\}) = 2/s \quad \text{and} \quad \mathcal{P}(s, Q, \{p, s-p\}) = 1/s \ , \ p \in [2, s-1]$$

All transitions to states of smaller size!

Cost Function

The only unknown component.

Idea: *Estimate the cost function experimentally!*

- Run each algorithm on many inputs of different sizes.
- Measure the real execution time consumed for each transition.
- Average the measurements and store in a table.
- Execute on the target architecture.

Cost functions

$$I(s) = \mathcal{R}(s, I) \qquad M(s) = \mathcal{R}(s, M) \qquad Q(s) = \mathcal{R}(s, Q)$$

Dynamic Programming

- $Opt(s)$: the minimum expected cost for sorting an input of size s
- $Opt(1) = 0$
- If $Opt(.)$ is known up to size $s - 1$, then

$$Opt(s) = \min\{optQ(s), optI(s), optM(s)\}$$

- $optX(s)$: the total expected cost of choosing action $X \in \{I, Q, M\}$ in state s and following the optimal policy thereafter.
- Optimal choice :

$$\pi^*(s) = \arg\min\{optQ(s), optI(s), optM(s)\}$$

Dynamic Programming (cnt'd)

InsertionSort

$$optI(s) = I(s)$$

MergeSort

$$optM(s) = M(s) + Opt(\lceil s/2 \rceil) + Opt(\lfloor s/2 \rfloor)$$

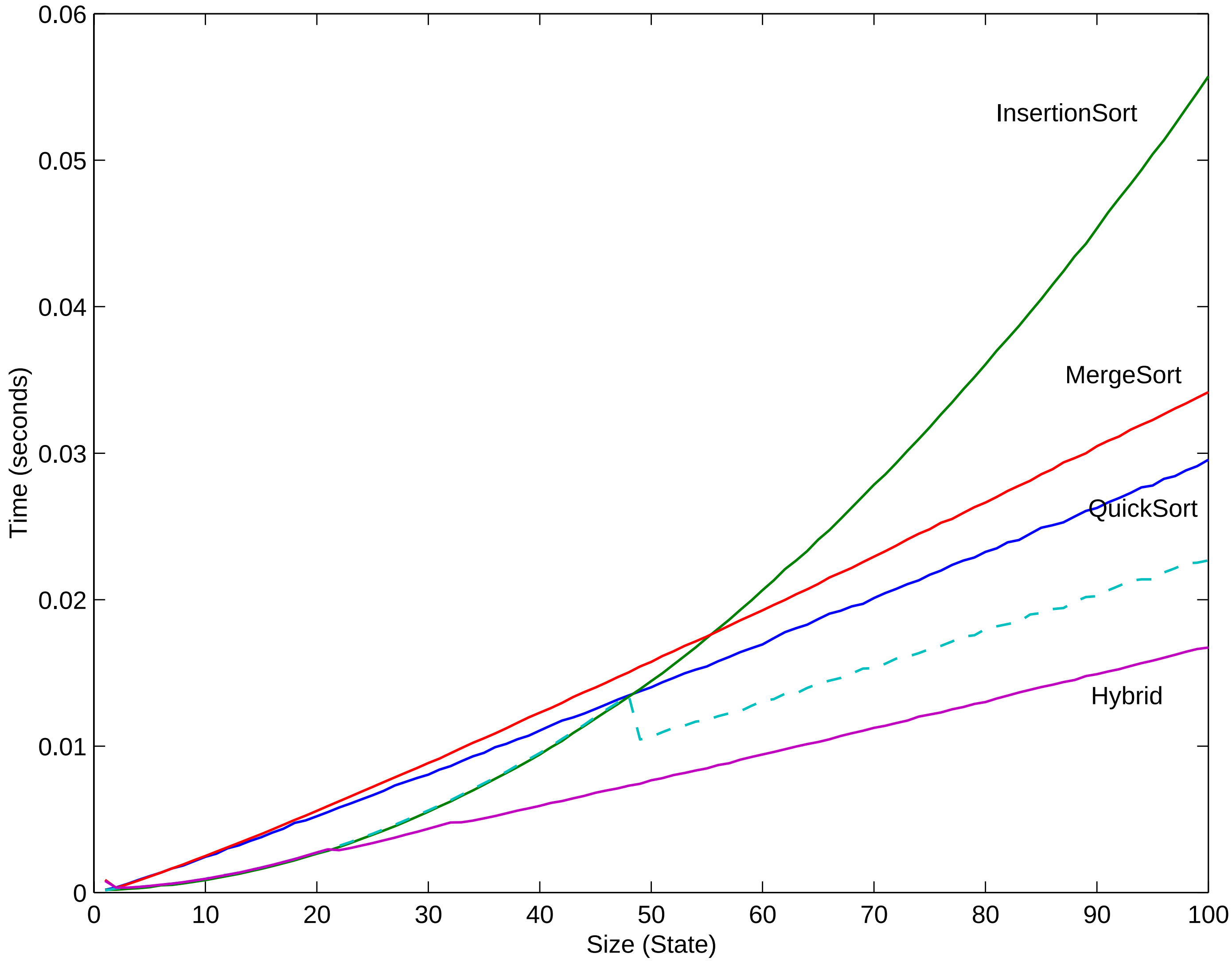
QuickSort

$$\begin{aligned} optQ(s) = Q(s) \quad &+ \quad \frac{1}{s}\sum_{p=2}^{s-1}\Big(Opt(p) + Opt(s-p)\Big) \\ &+ \quad \frac{2}{s}\Big(Opt(1) + Opt(s-1)\Big) \end{aligned}$$

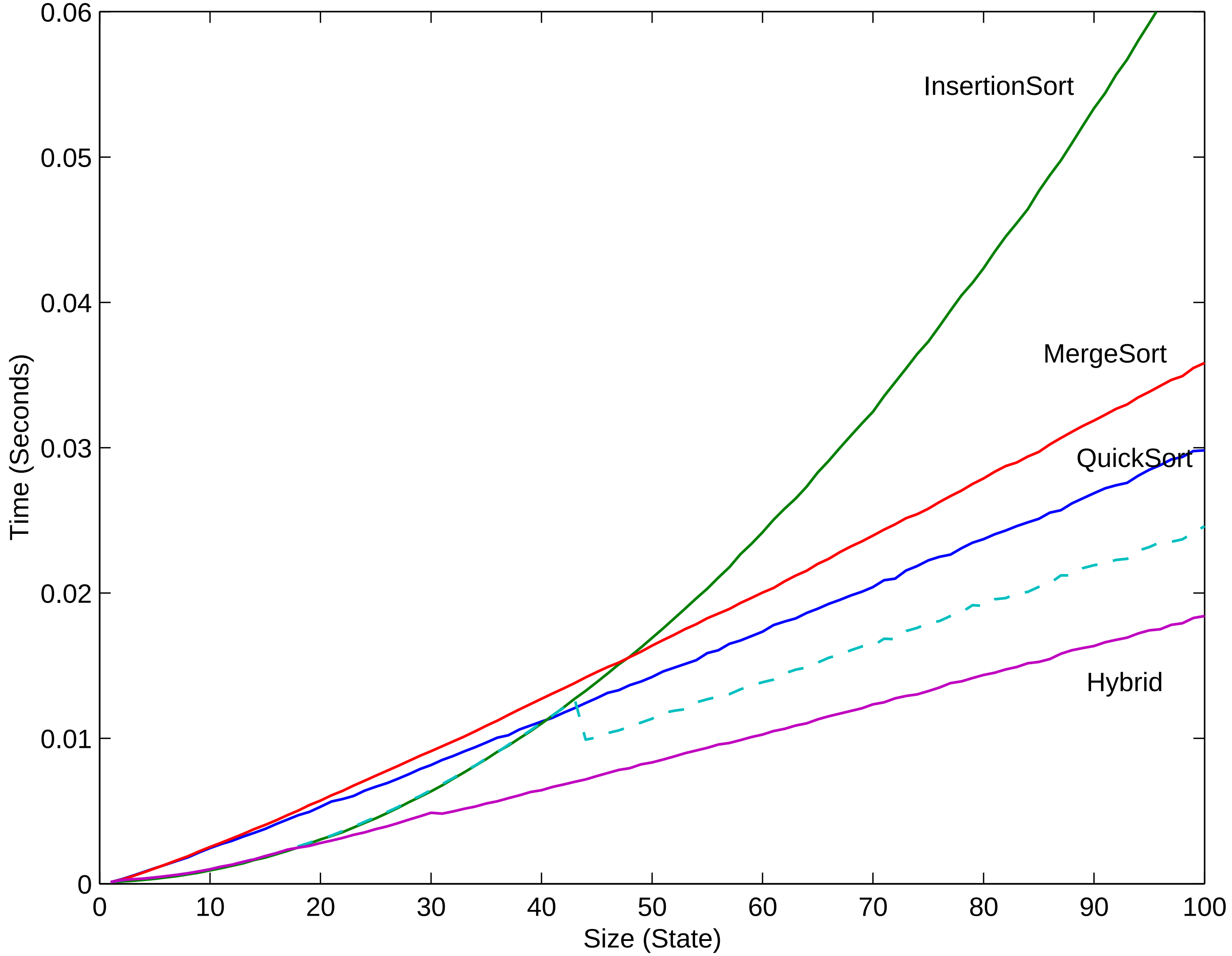
Optimal Hybrid Policies

Sparc	(Solaris)	**Pentium**	(Linux)
Size	Algorithm	Size	Algorithm
2 - 21	InsertionSort	2 - 17	InsertionSort
22 - 32	MergeSort	18 - 30	MergeSort
33 - ...	QuickSort	31 - ...	QuickSort

Performance on a Sparc/Solaris Architecture



Performance on a Pentium/Linux Architecture



Issues

Limitations

- State Description
- Hidden State
- Model Derivation

Adaptation

- Machine Learning
- Function Approximation
- Learning while Computing

Domains

- Hard Combinatorial Problems

Some Related Work

- FFTW (Fast Fourier Transform) [Frigo and Johnson, 1998]
- PYTHIA (Scientific Software) [Houstis et al., 1991-2000]
- LAPACK (Linear Algebra) [Anderson et al., 1987-2001]
- STAGE (Local Search) [Boyan, 1998]
- RLSAT (Satisfiability) [Lagoudakis and Littman, 2001]
- Bayesian Modelling [Horvitz at al., 2001]
- Branch-and-Bound Search [Lobjois and Lemaitre, 1998]
- Problem Solving [Fink, 1998]
- ...

Conclusion

The Past

- There is no uncertainty in computation.
- The best algorithm is the one with the best worst-case guarantees.
- Static Software (fixed algorithms)

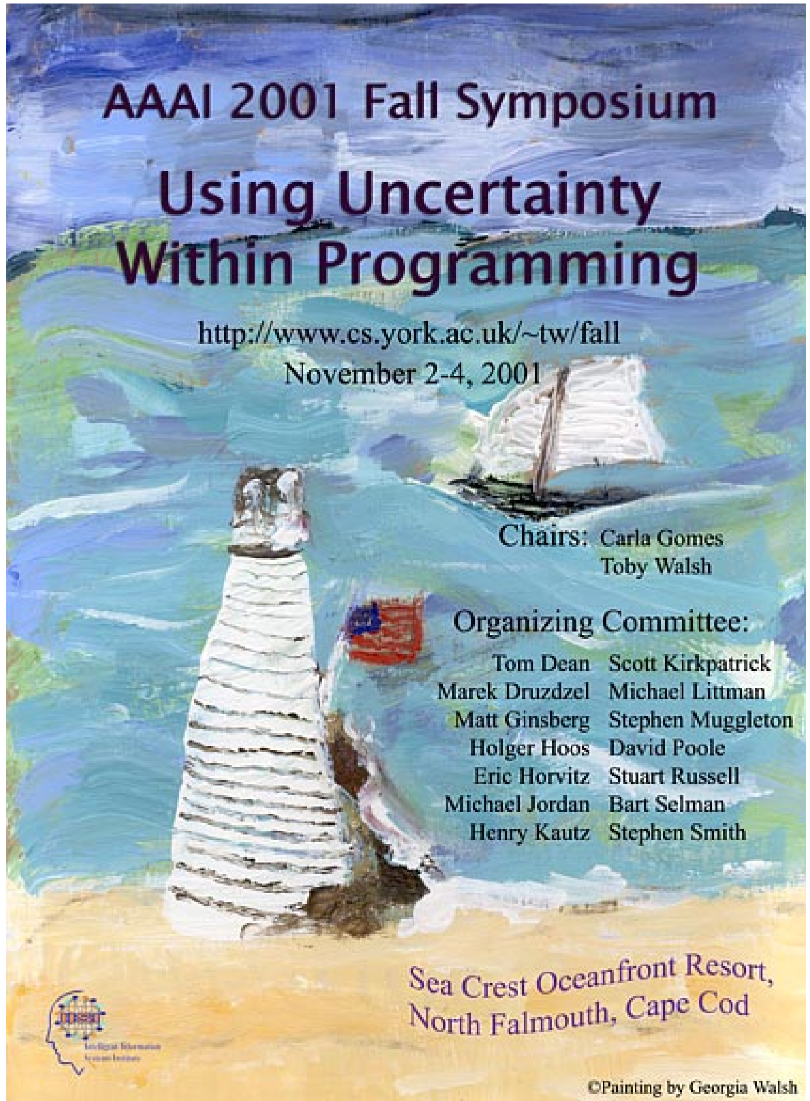
The Present

- There *is* uncertainty in computation.
- Optimization, reasoning, and learning methods can cope with uncertainty.
- Optimized Software (hybrid algorithms)

The Future

- Adaptive systems that encapsulate many algorithms.
- Systems that "learn" from past experience and improve their performance.
- Intelligent Software (AI?)

Thank You!


AAAI 2001 Fall Symposium
Using Uncertainty Within Programming
http://www.cs.york.ac.uk/~tw/fall
November 2-4, 2001
Chairs: Carla Gomes
Toby Walsh
Organizing Committee:
Tom Dean Scott Kirkpatrick
Marek Druzdzel Michael Littman
Matt Ginsberg Stephen Muggleton
Holger Hoos David Poole
Eric Horvitz Stuart Russell
Michael Jordan Bart Selman
Henry Kautz Stephen Smith
Sea Crest Oceanfront Resort,
North Falmouth, Cape Cod
©Painting by Georgia Walsh