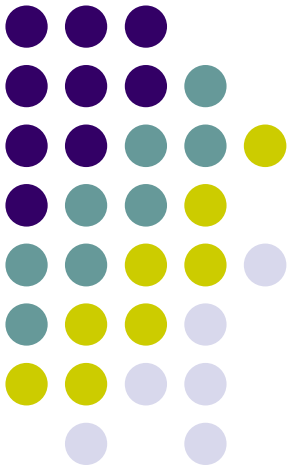
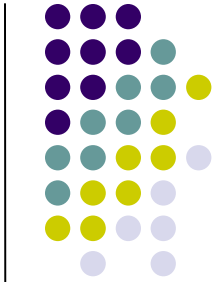
Adaptive Compilation

Yi-Fan Tsai

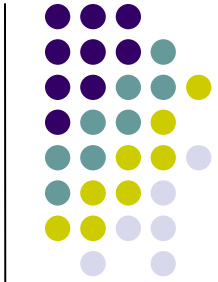


Outline



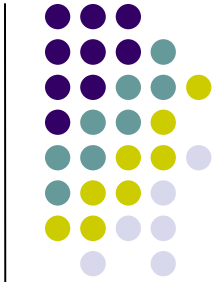
- Overview
- Phase Ordering Problem
- Determining Good Optimization Settings

Overview



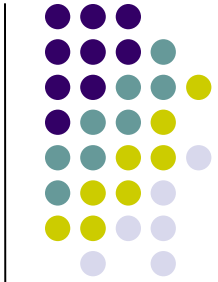
- The goal of adaptive compilation is to find the best combination of optimizations and parameters

Obstacles



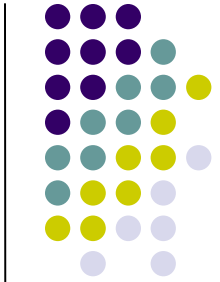
- The large amount of time that the systems have used
- The complexity inherent in a feedback-driven adaptive system

Addressed Problems



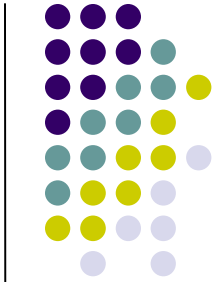
- Phase ordering problem
  - Prasad A. Kulkarni, David B. Whalley, Gary S. Tyson, Jack W. Davidson. Evaluating Heuristic Optimization Phase Order Search Algorithms, In CGO 2007.
- Determining good optimization settings
  - John Cavazos, Grigori Fursin, Felix Agakov, Edwin Bonilla, Michael F.P. O'Boyle, and Olivier Temam. Rapidly Selecting Good Compiler Optimizations using Performance Counters, In CGO 2007.

Phase Ordering Problem



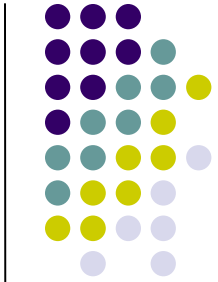
- Each optimization phase may create or destroy specific conditions of other phases.
- Finding the best sequence of optimization phases to apply is known as the phase ordering problem.

Obstacles

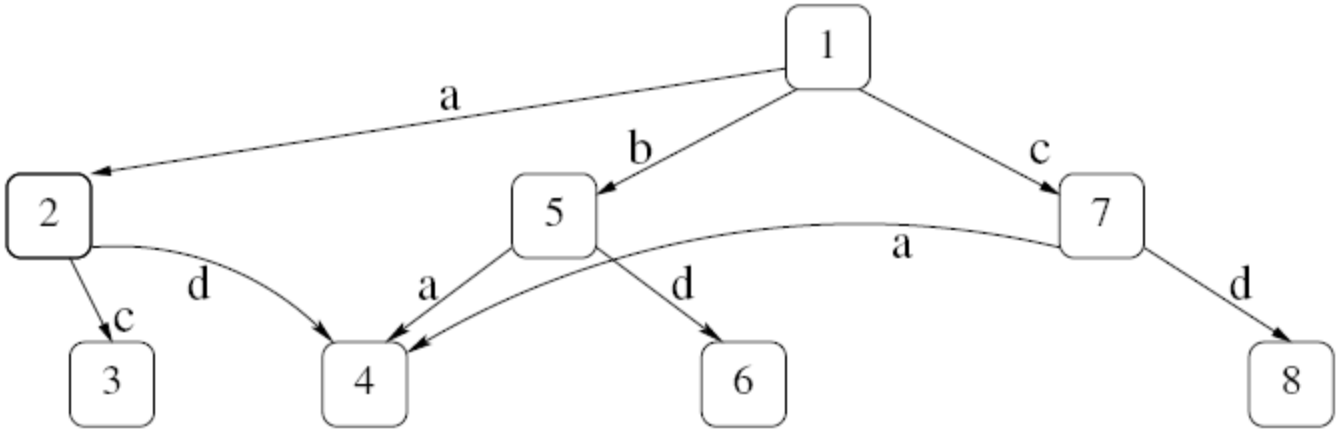


- The relationship and interactions between optimization phases remain ill-understood.
- The space of all possible orderings of optimization phases is huge since
  - Numerous different optimization phases
  - Different sequence lengths are allowed
  - Repeating phases is allowed

Exhaustive Exploration

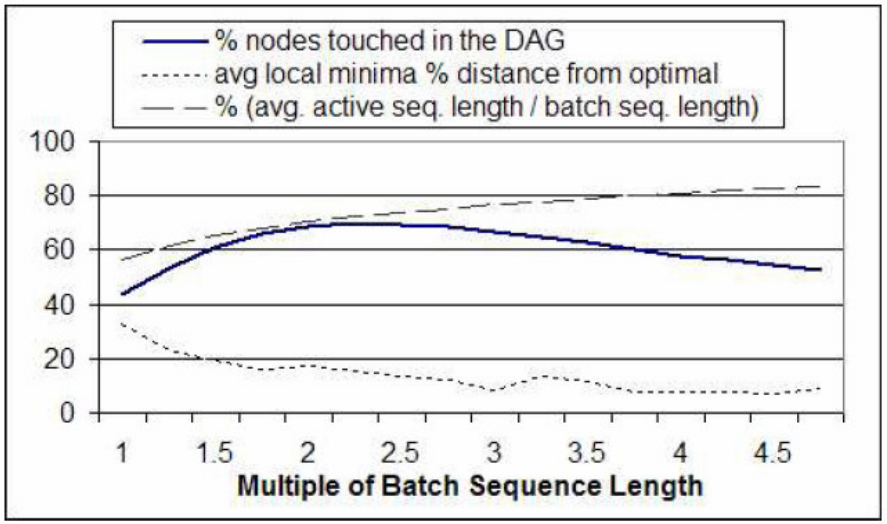


- Nodes represent distinct function instances
- Edges represent transitions on application of an optimization phase

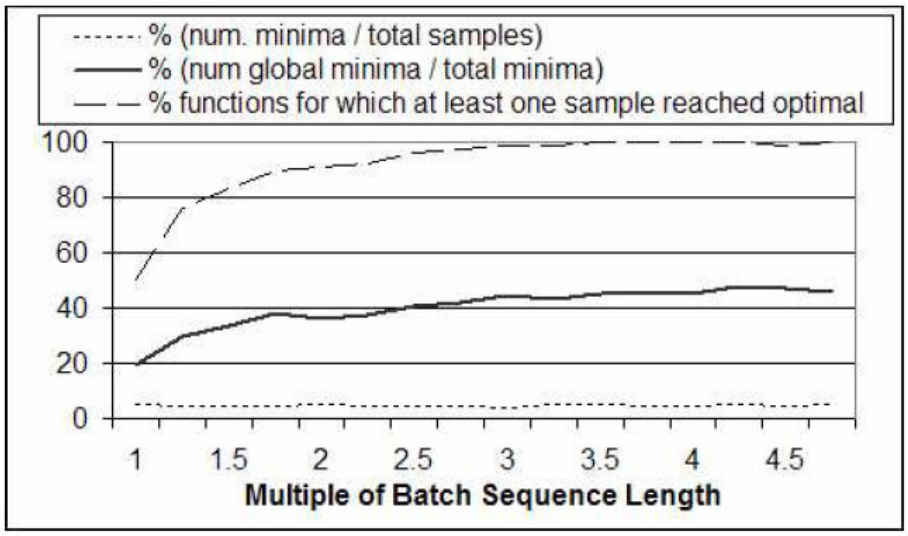


(b) Depth–first Traversal

Search Space Properties

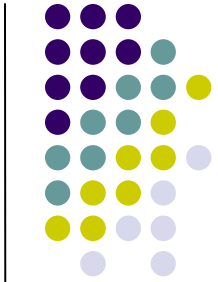


(a) Local Minima Information



(b) Global Minima Information

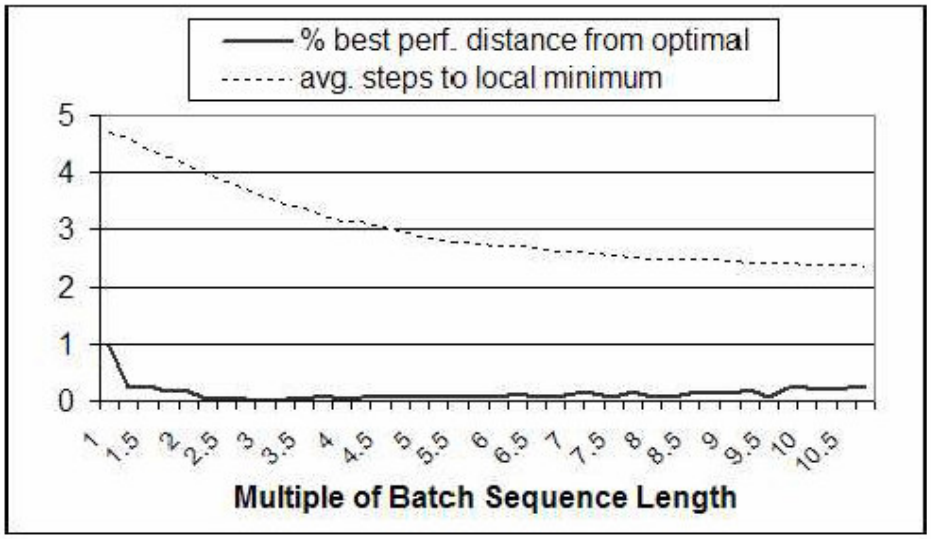
N-Lookahead



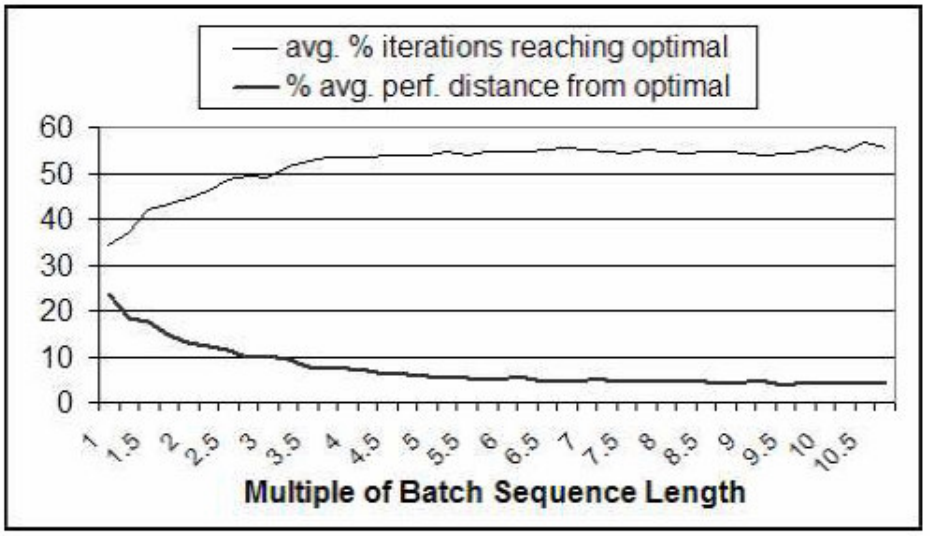
- The algorithm scans N levels to select the phase that leads to the best result
- The result shows the unpredictable nature of phase interactions

	Lookahead		
	1	2	3
% Performance	22.90	14.64	5.35

Hill Climbing

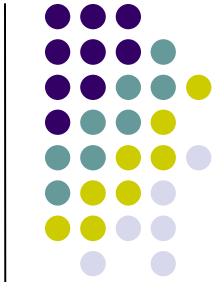


(a) Local Minima Information

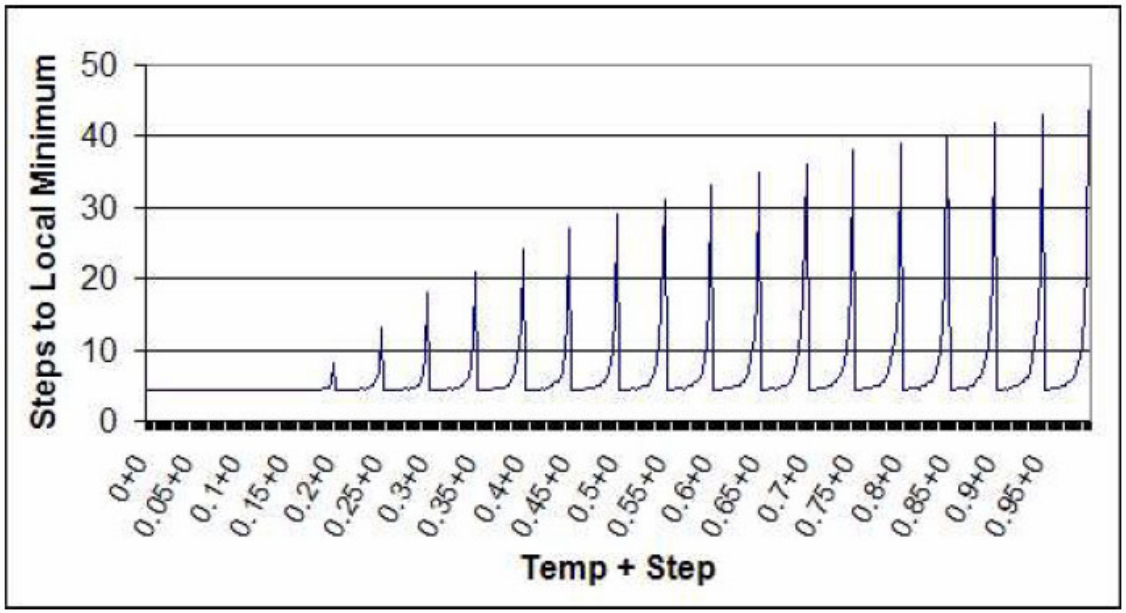


(b) Global Minima Information

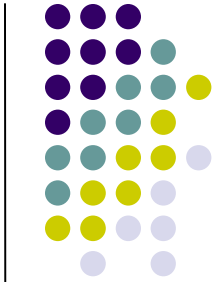
Simulated Annealing



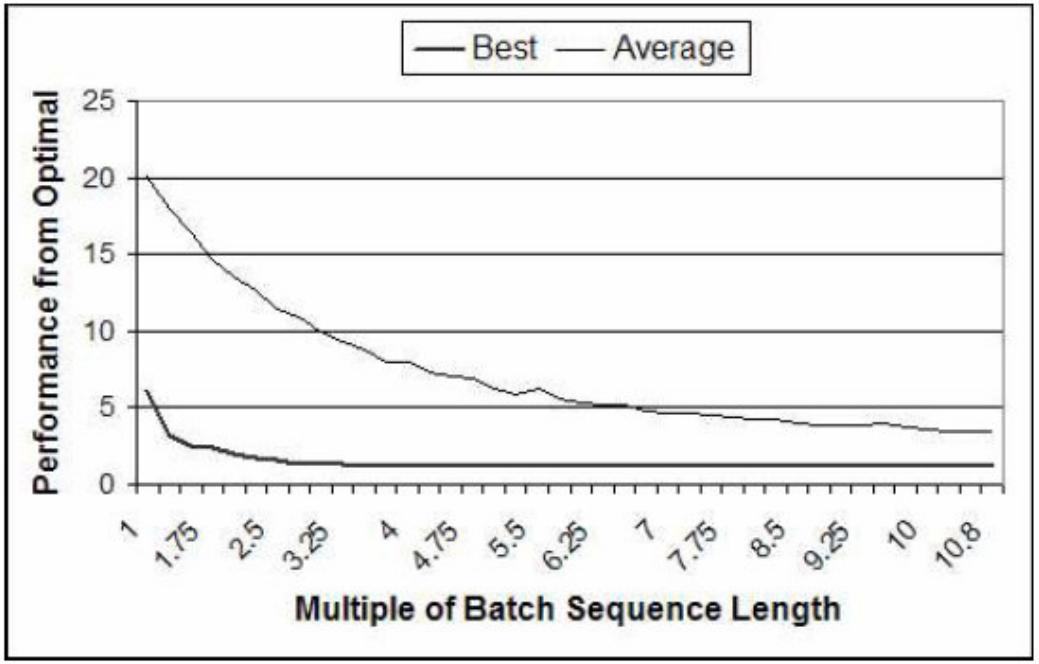
- The increase in the number of steps to local optimal does not translate into any significant performance improvement



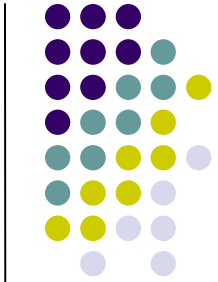
Greedy Algorithm



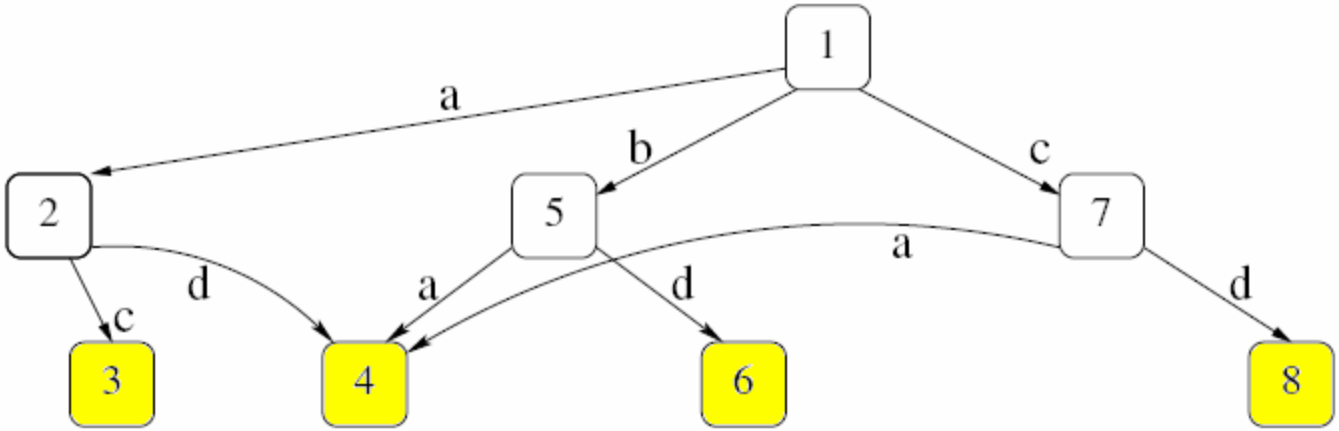
- The best achievable performance, 1.1% worse than optimal, is slightly worse than that for the hill climbing algorithm (0.02%)



Leaf Sequences

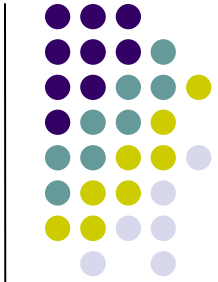


- Leaf function instances are those that cannot further modified by the application of additional optimization phases.
- The sequences leading to leaf function instances are called leaf sequences.

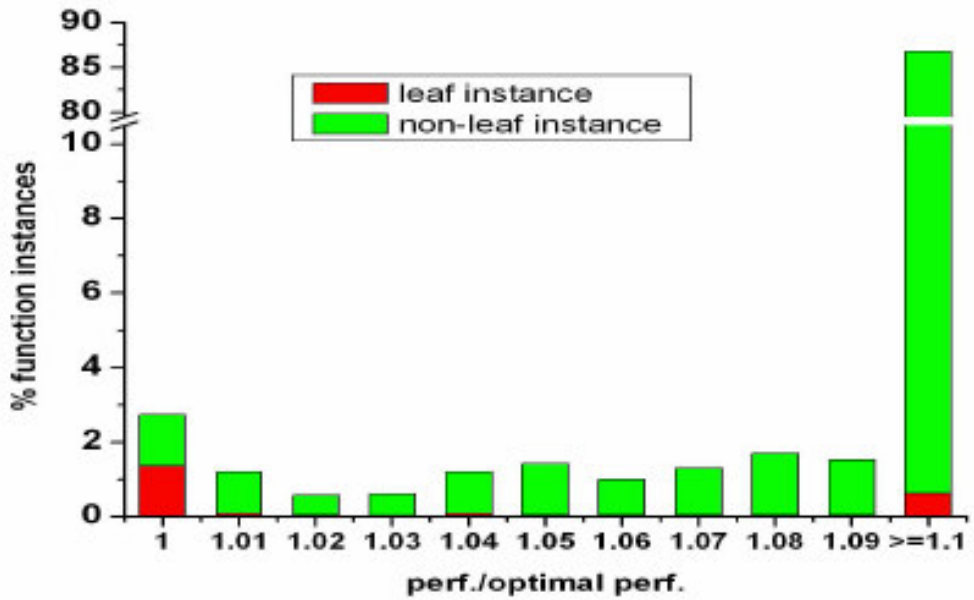


(b) Depth–first Traversal

Properties of Leaf Sequences

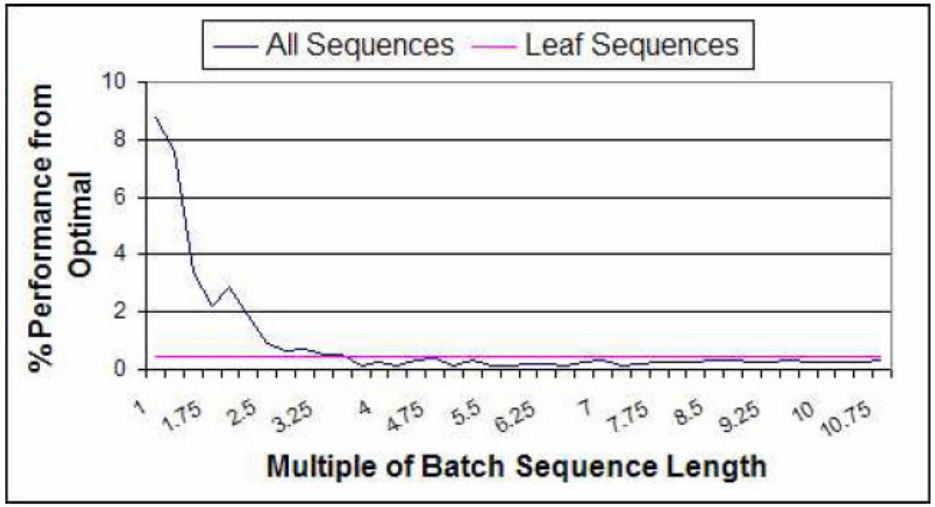


- The performance is typically closely optimal.
- The leaf instances comprise a significant portion of optimal instances and a very small portion of the total space.

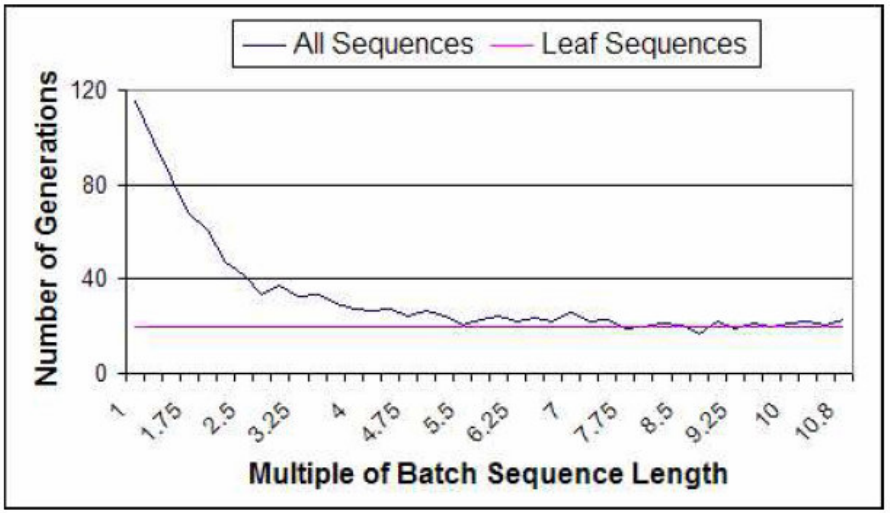


Modified Genetic Algorithm

- The modified algorithm handles the new sequences by squeezing out the dormant phases and extending it with randomly generated phases to get a leaf sequence.



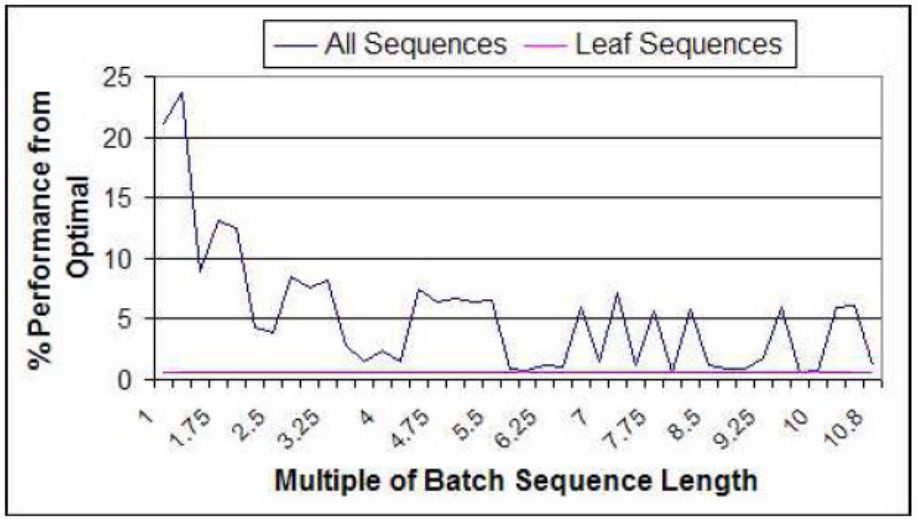
(a) Performance



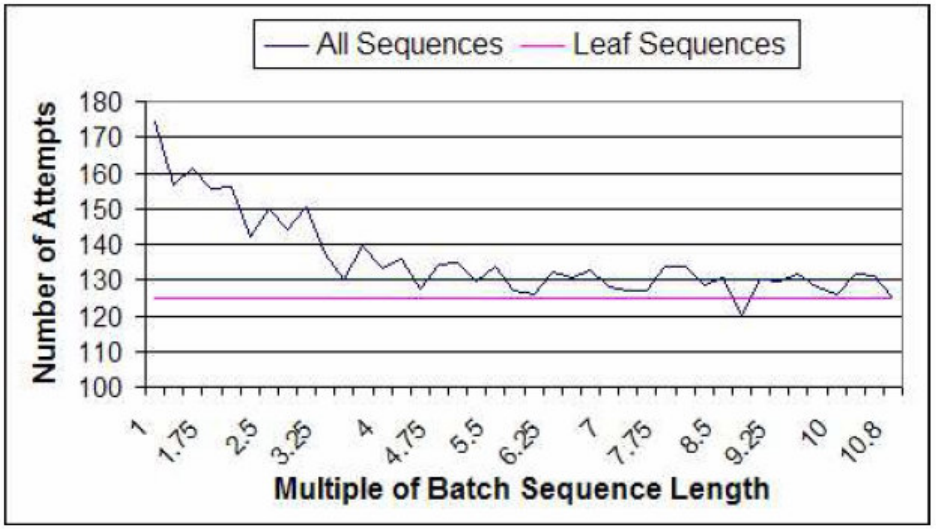
(b) Cost

Modified Random Search

- The modified algorithm only considers leaf sequences

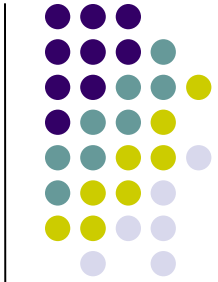


(a) Performance



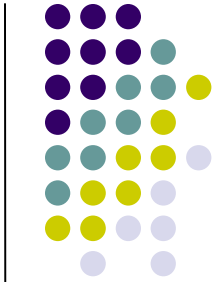
(b) Cost

Determining Good Optimization Settings

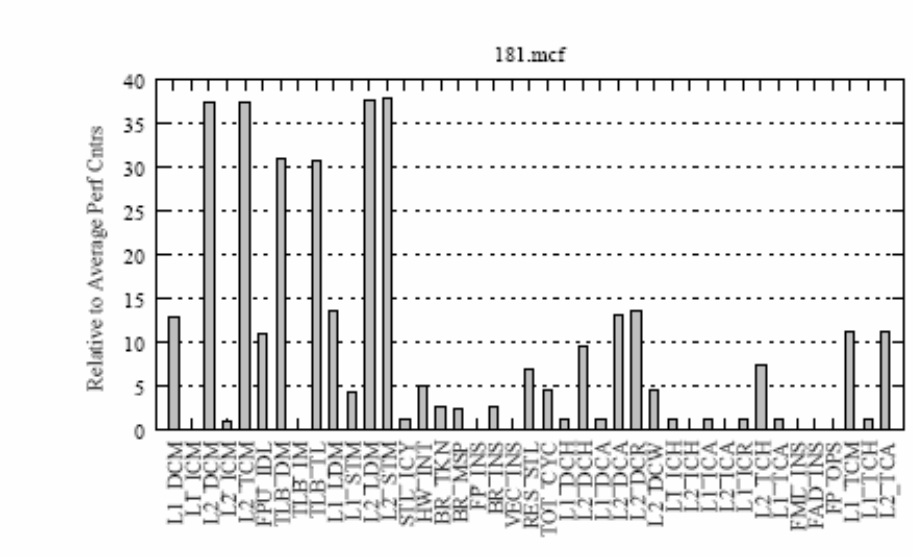


- Automatically selecting the best set of compiler optimizations for a particular program is a difficult task.
- The static code features can only characterize local code constructs.

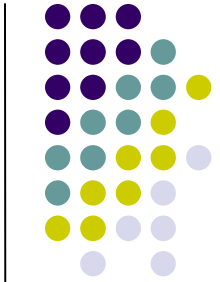
Motivation



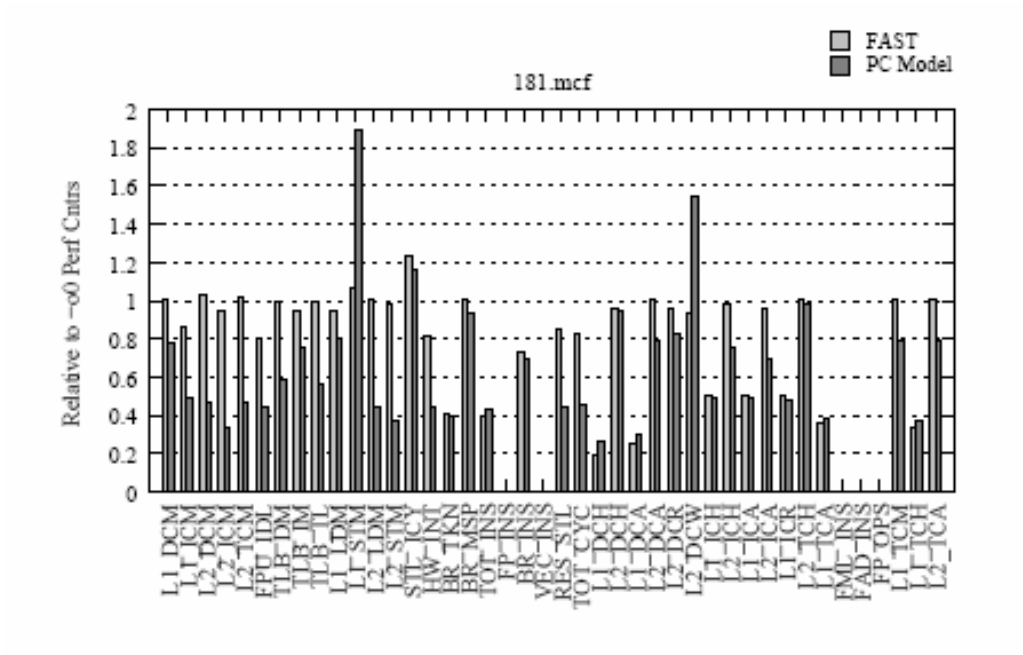
- The information obtained from performance counters is a compact summary of a program's dynamic behaviour.
- Example: The graph shows that the program has a much number of memory accesses.



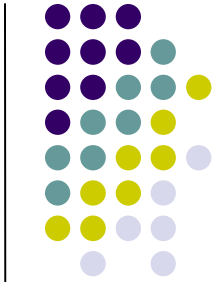
Example



- Generating 32-bit code may be only useful for a few programs which have lots of variables of the type long and/or pointers

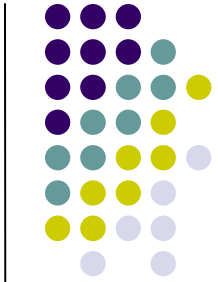


Optimization Selection Based on Performance Counters



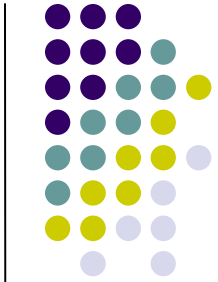
- Extract the performance counter features by running the target program.
- Feed this feature vector to the trained model, and then get the output of a probability for each optimization.
- Sample from the this probability distribution to generate the a optimization setting.

Model Construction



- The model is built using a training set.
- Use logistic regression to determine for each optimization the probability.

Performance



- This method achieves a speedup compatible to the combined elimination algorithm but with much fewer evaluations.
- The performance counters are significantly better for characterizing large programs with complex control flow.

Questions?