Evaluating Heuristic Optimization Phase Order Search Algorithms

by
Prasad A. Kulkarni
David B. Whalley *Gary S. Tyson* *Jack W. Davidson*

Computer Science Department, Florida State University, Tallahassee, Florida

Computer Science Department, University of Virginia, Charlottesville, Virginia

2007 International Symposium on Code Generation and Optimization (CGO)

Compiler Optimizations

- Optimization phases require enabling conditions
  - need specific patterns in the code
  - many also need available registers
- Phases interact with each other
- Applying optimizations in different orders generates different code

Phase Ordering Problem

- To find *an* ordering of optimization phases that produces *optimal* code with respect to possible phase orderings
- Evaluating each sequence involves compiling, assembling, linking, execution and verifying results
- Best optimization phase ordering depends on
  - source application
  - target platform
  - implementation of optimization phases
- Long standing problem in compiler optimization!!

Addressing Phase Ordering

- Exhaustive phase order space evaluation [CGO '06, LCTES '06]
  - possible for most functions
  - long search times for larger functions
- Heuristic approaches
  - commonly employed, extensively studied
  - allow faster searches
  - no guarantees on solution quality

Survey of Heuristic Algorithms

- Cost and performance comparison
  - with optimal
  - with other heuristic searches
- Analyze phase order space properties
  - sequence length
  - leaf sequences
- Improving heuristic search algorithms
  - propose new algorithms

Outline

- Experimental setup
- Local search techniques
  - distribution of local minima
  - local search algorithms
- Exploiting properties of leaf sequences
  - genetic search algorithm
- Conclusions

Outline

- Experimental setup
- Local search techniques
  - distribution of local minima
  - local search algorithms
- Exploiting properties of leaf sequences
  - genetic search algorithm
- Conclusions

Experimental Framework

- We used the VPO compilation system
  - established compiler framework, started development in 1988
  - comparable performance to gcc –O2
- VPO performs all transformations on a single representation (RTLs), so it is possible to perform most phases in an arbitrary order
- Experiments use all the 15 re-orderable optimization phases in VPO
- Target architecture was the StrongARM SA-100 processor

VPO Optimization Phases

ID	Optimization Phase	ID	Optimization Phase
b	branch chaining	l	loop transformations
c	common subexpr. elim.	n	code abstraction
d	remv. unreachable code	o	eval. order determin.
g	loop unrolling	q	strength reduction
h	dead assignment elim.	r	reverse branches
i	block reordering	s	instruction selection
j	minimize loop jumps	u	remv. useless jumps
k	register allocation		

Benchmarks

- 12 MiBench benchmarks; 88 functions

Category	Program	Description
auto	bitcount	test processor bit manipulation abilities
	qsort	sort strings using quicksort sorting algorithm
network	dijkstra	Dijkstra’s shortest path algorithm
	patricia	construct patricia trie for IP traffic
telecomm	fft	fast fourier transform
	adpcm	compress 16-bit linear PCM samples to 4-bit
consumer	jpeg	image compression and decompression
	tiff2bw	convert color .tiff image to b&w image
security	sha	secure hash algorithm
	blowfish	symmetric block cipher with variable length key
office	stringsearch	searches for given words in phrases
	ispell	fast spelling checker

Terminology

- *Active* phase – an optimization phase that modifies the function representation
- *Dormant* phase – a phase that is unable to find any opportunity to change the function
- *Function instance* – any semantically, syntactically, and functionally correct representation of the source function (that can be produced by our compiler)

Terminology (cont...)

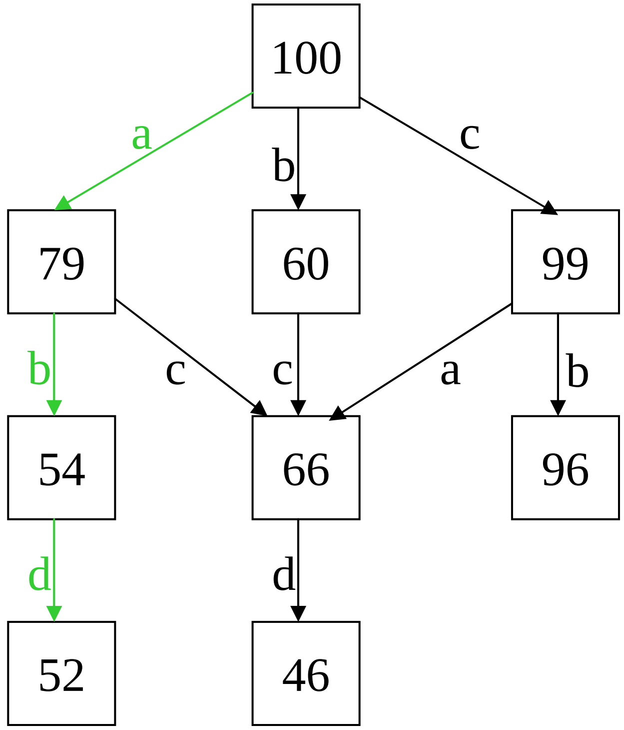
- Attempted sequence – phase sequence comprising of both active and dormant phases
- Active sequence – phase sequence only comprising active phases
- Batch sequence – active sequence applied by the default (*batch*) compiler

Setup for Analyzing Search Algorithms

- Exhaustively evaluate optimization phase order space
  - represent phase order space as DAG
- For each search algorithm
  - use algorithm to generate next optimization phase sequence
  - lookup performance in DAG

Phase Order Search Space DAG

- Performance evaluation of each phase order is traversal in the DAG
  - a-b-d = 52



Outline

- Experimental setup
- Local search techniques
  - distribution of local minima
  - local search algorithms
- Exploiting properties of leaf sequences
  - genetic search algorithm
- Conclusions

Local Search Techniques

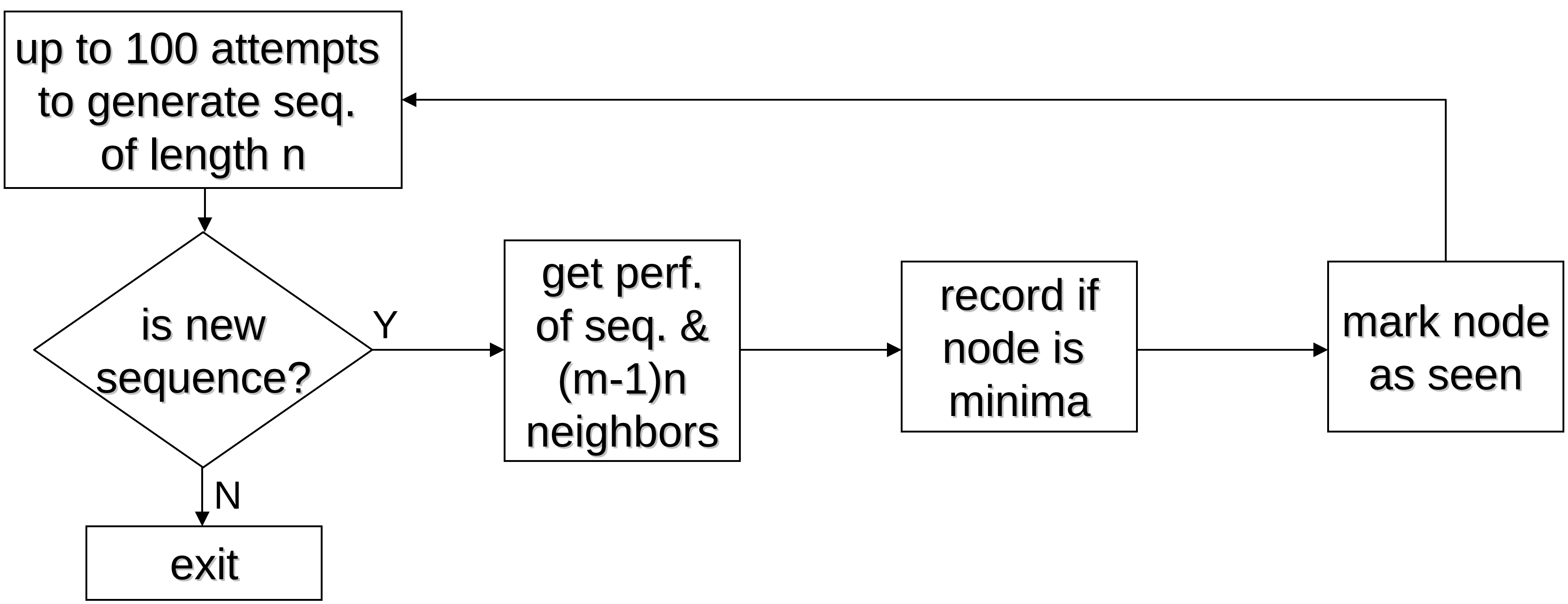
- Consecutive sequences differ in only one position

- if *m* phases and a sequence length of *n,* then will have *n(m-1)* neighbors

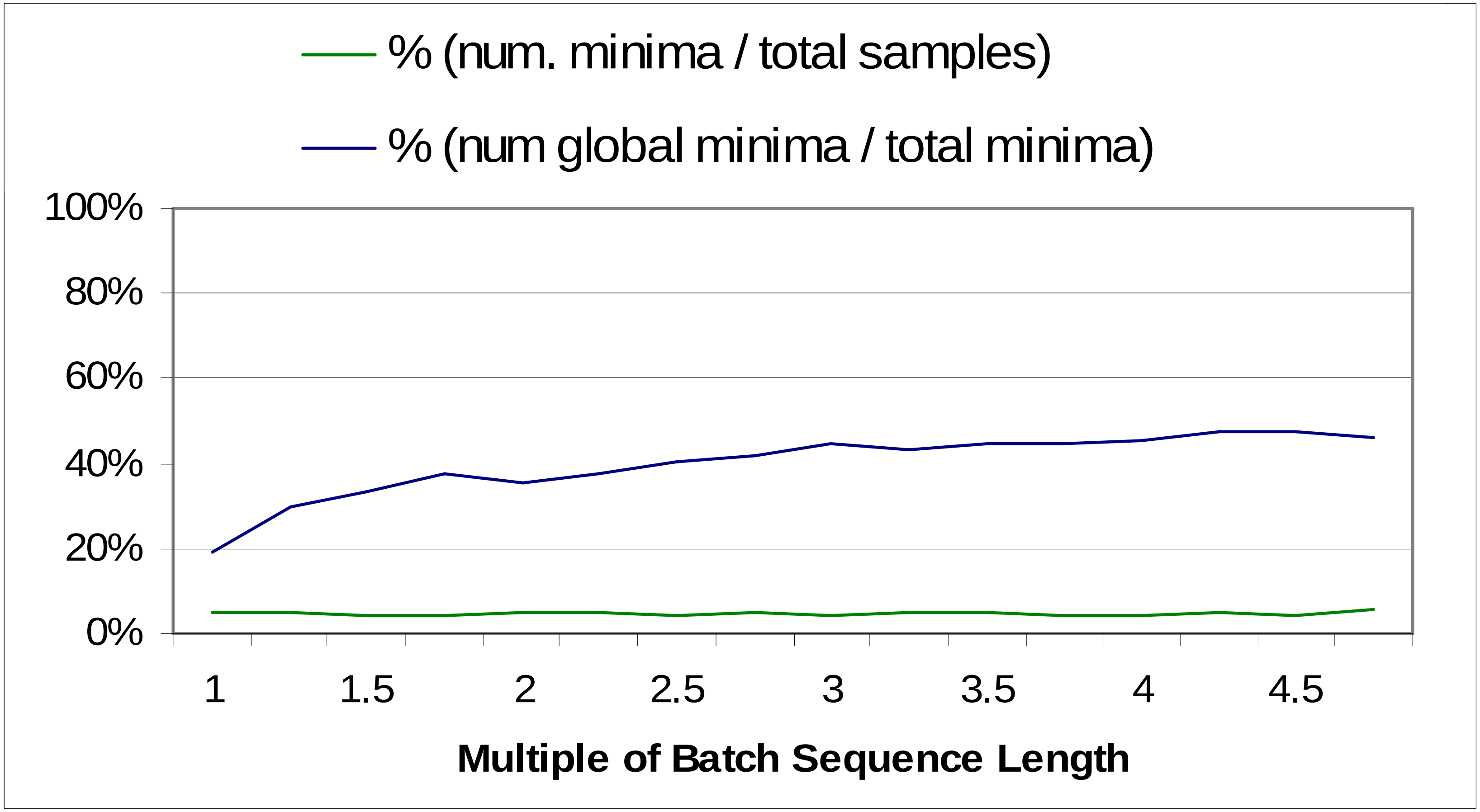
bseq	**neighbors**							
a	**b**	**c**	a	a	a	a	a	a
b	b	b	**a**	**c**	b	b	b	b
c	c	c	c	c	**a**	**b**	c	c
a	a	a	a	a	a	a	**b**	**c**

Local Search Space Properties

- Analyze local search space to
  – study distribution of local and global minima
  – study importance of sequence length

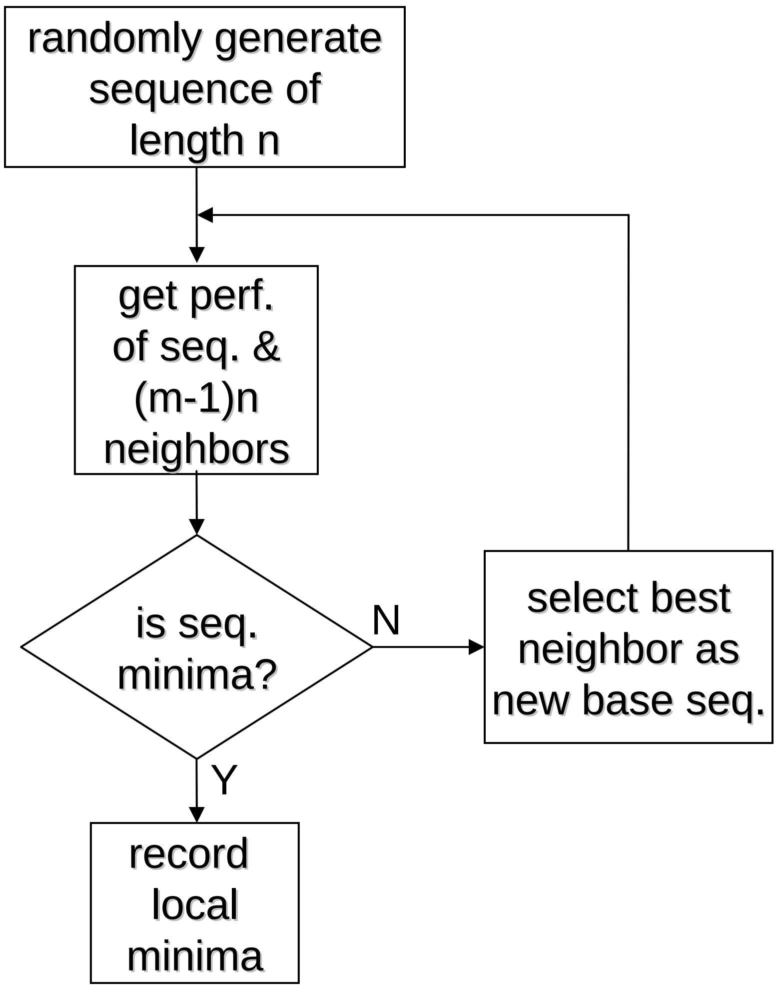


Distribution of Minima

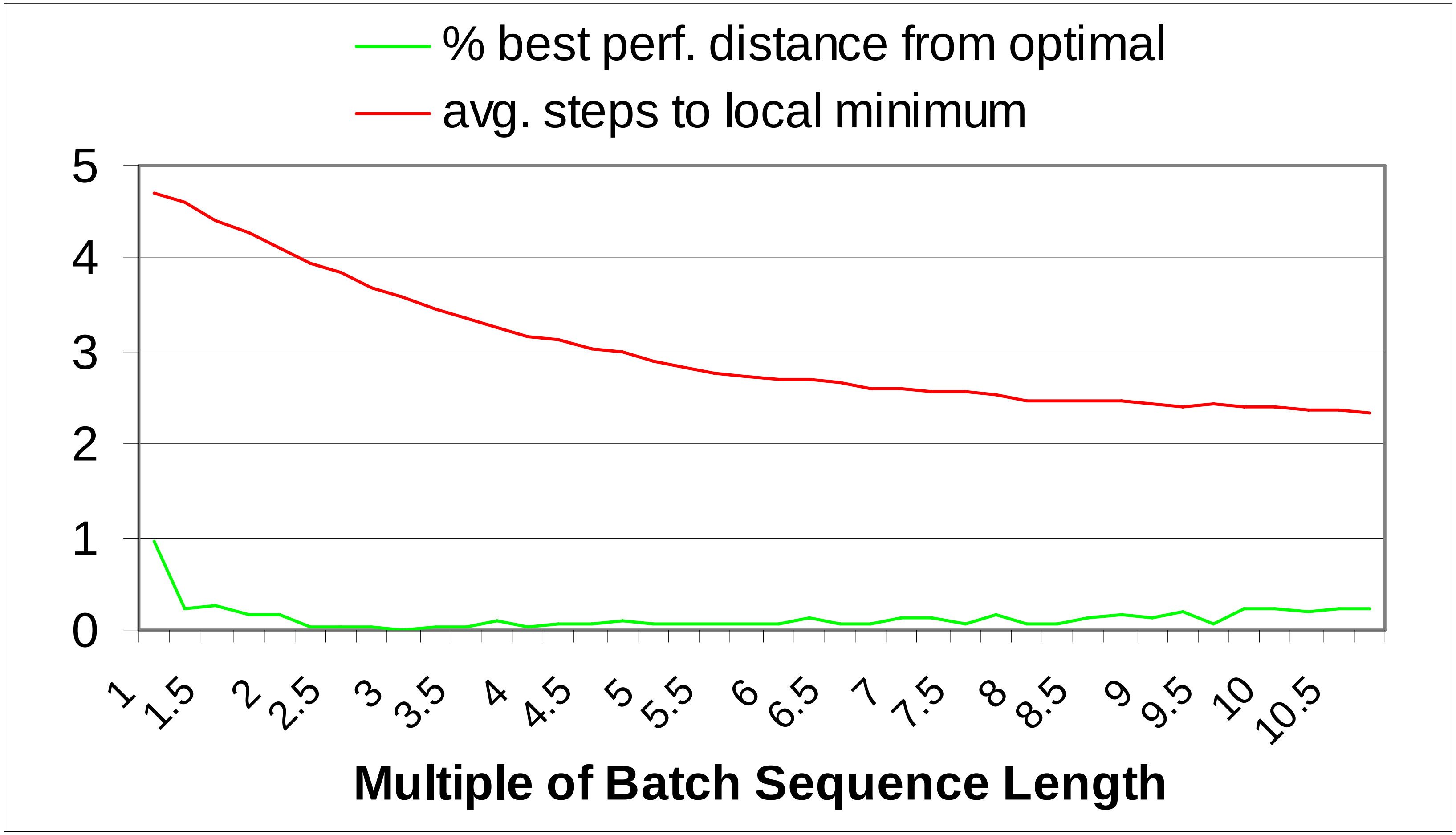


Hill Climbing

- Steepest decent
  - compare all successors of base
  - exit on local minima



Hill Climbing Results (cont...)



Local Search Conclusions

- Phase order space consists of few minima, but significant percentage of local minima can be optimal
- Selecting appropriate sequence length is important
  - smaller length results in bad performance
  - larger length is expensive

Outline

- Experimental setup
- Local search techniques
  - distribution of local minima
  - local search algorithms
- Exploiting properties of leaf sequences
  - genetic search algorithm
- Conclusions

Leaf Sequence Properties

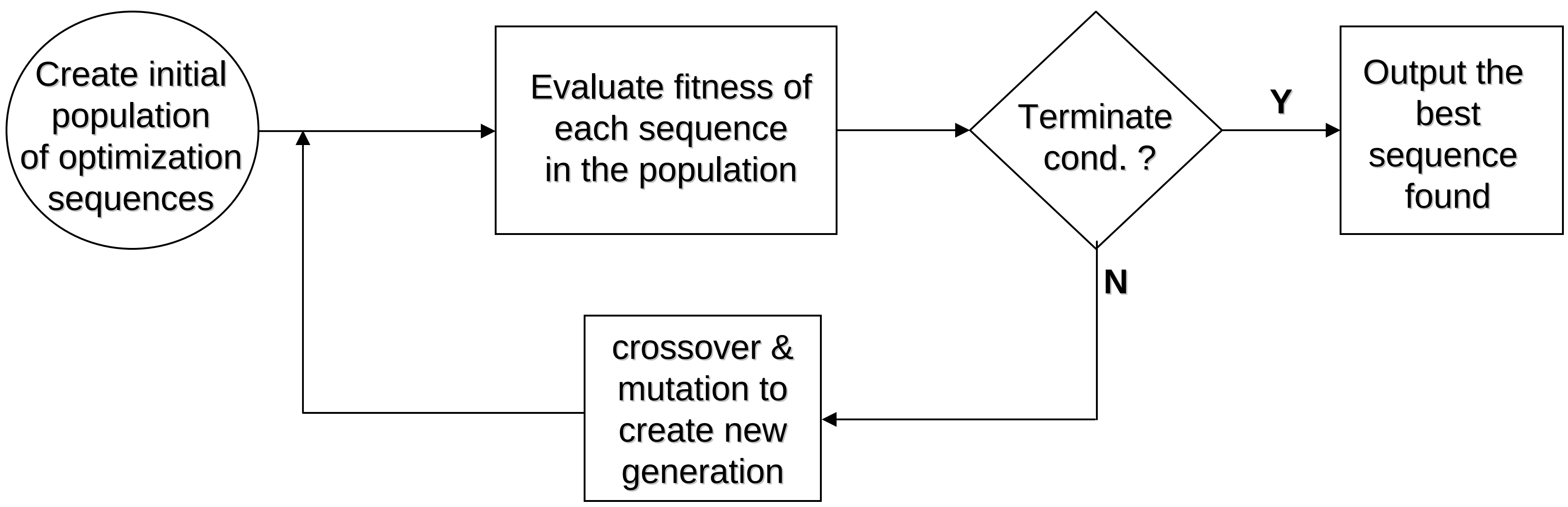
- *Leaf function instances* are generated when no additional phases can be successfully applied
  - sequences leading to leaf function instances are *leaf sequences*
- Leaf sequences result in *good* performance
  - at least one leaf instance represents an *optimal* phase ordering for over 86% of functions
  - significant percentage of leaf instances among optimal

Focusing on Leaf Sequences

- Modify phase order search algorithms to only produce leaf sequences
  - no need to guess appropriate sequence length
  - likely to result in optimal or close to optimal performance
  - leaf function instances comprise only 4.2% of the total instances

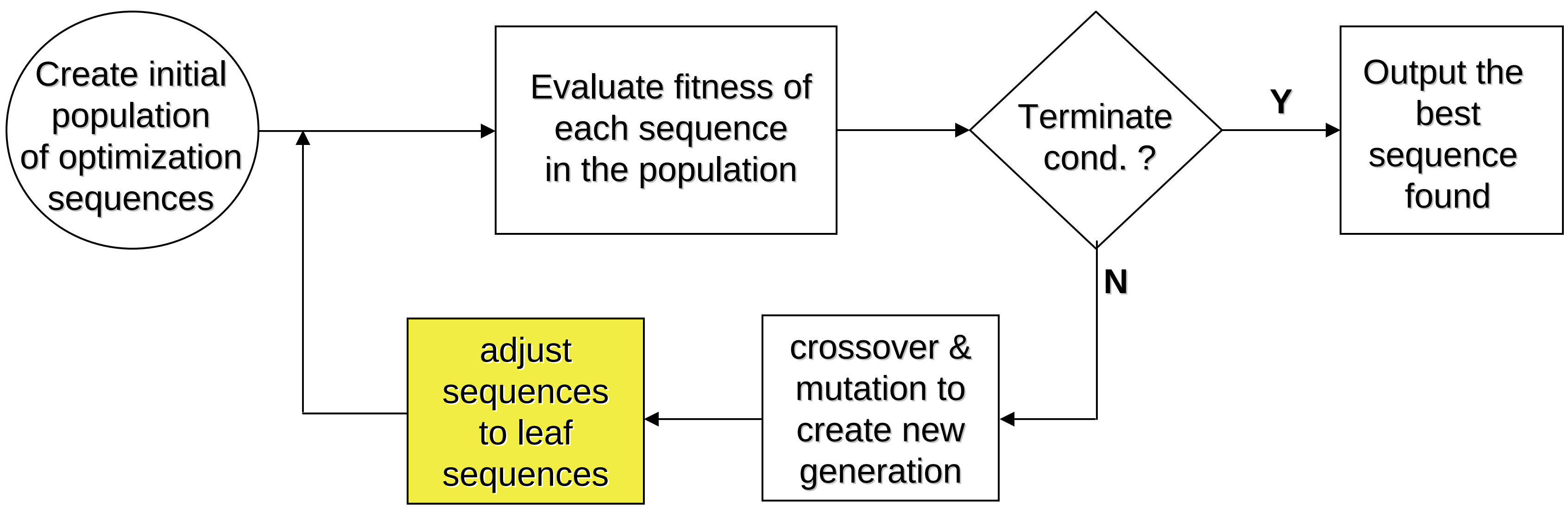
Genetic Algorithm

- A biased sampling search method
  - evolves solutions by merging parts of different solutions

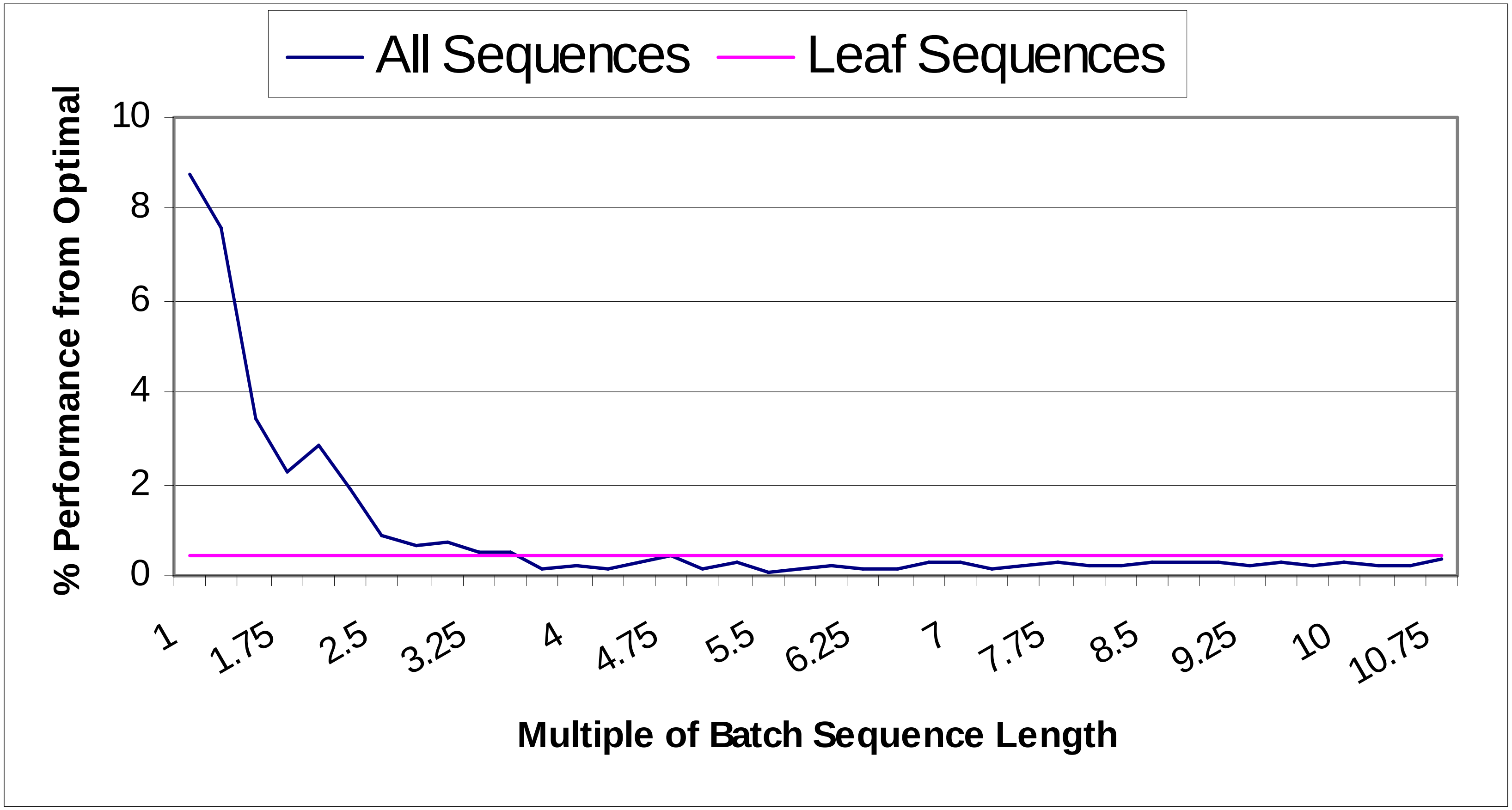


Modified Genetic Algorithm

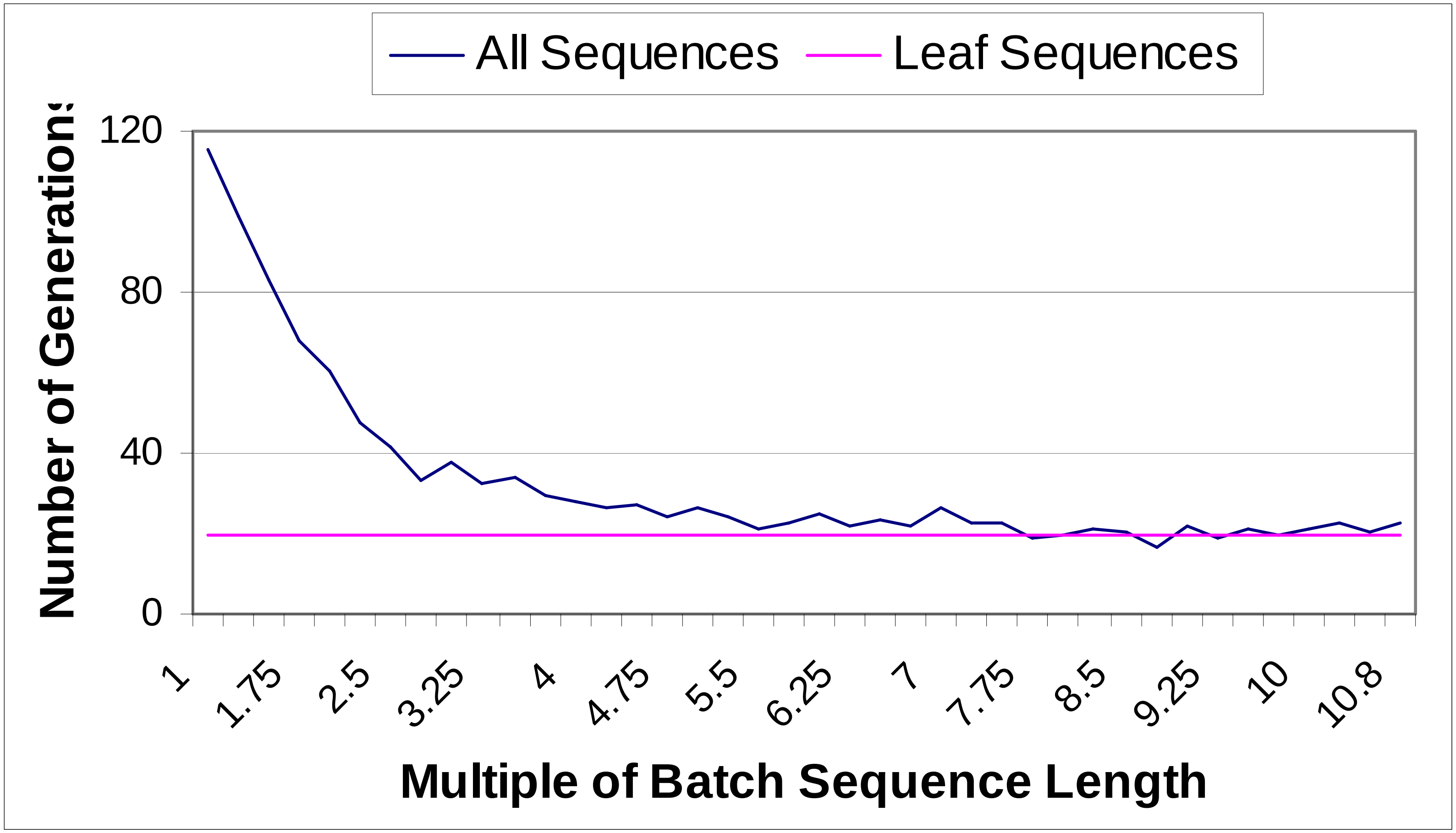
- Only generate leaf sequences



Genetic Algorithm – Performance



Genetic Algorithm – Cost



Leaf Search Conclusion

- Benefits of restricting searches to leaf sequences
  - no need for apriori knowledge of appropriate sequence length
  - near-optimal performance
  - low cost

Outline

- Experimental setup
- Local search techniques
  - distribution of local minima
  - local search algorithms
- Exploiting properties of leaf sequences
  - genetic search algorithm
- Conclusions

Conclusions

- First study to compare heuristic search solutions with optimal orderings
- Analyzed properties of phase order search space
  - few local and global minima
- Illustrated importance of choosing the appropriate sequence length
- Demonstrated importance of leaf sequences
  - achieve near-optimal performance at low cost

Questions ?

Simulated Annealing

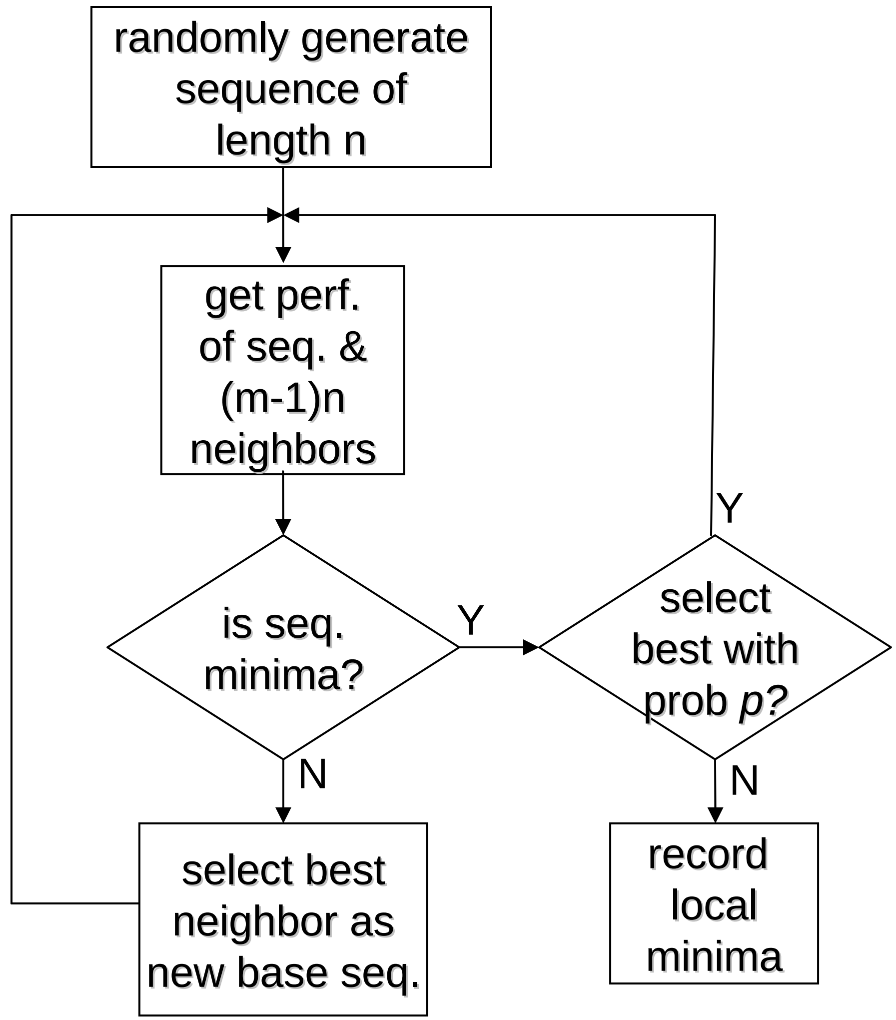
- A worse solution is accepted with

  prob = exp (-δf/T)

  – δf → diff. in perf.

  – T → current temp.

- Annealing schedule
  – initial temperature
  – cooling schedule

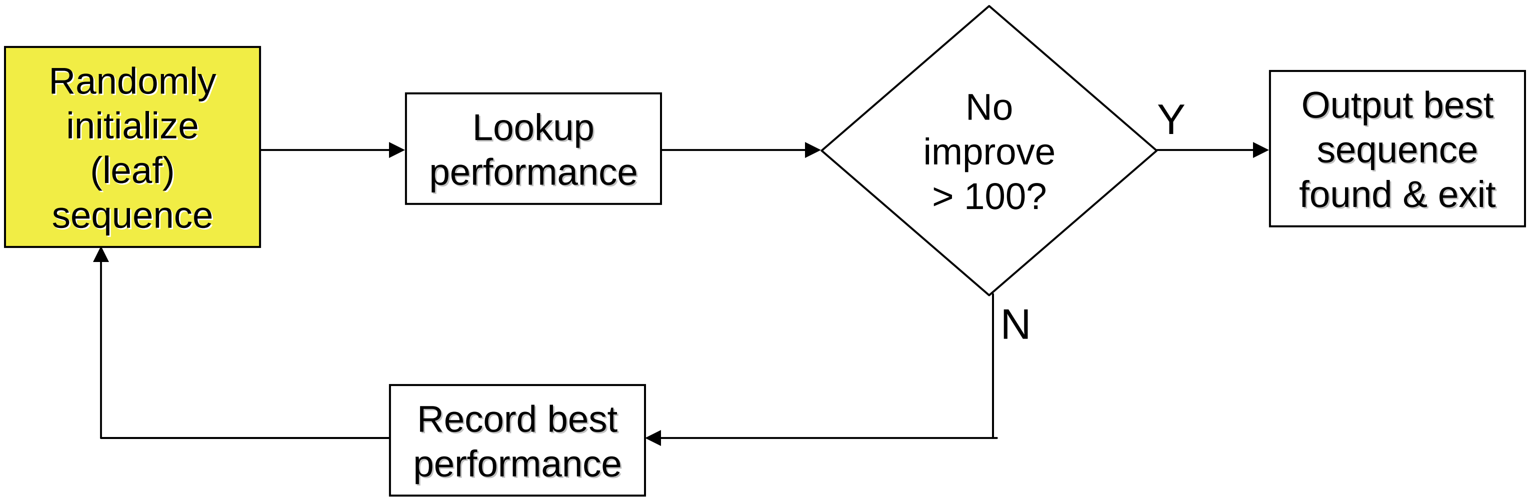


Simulated Annealing (cont...)

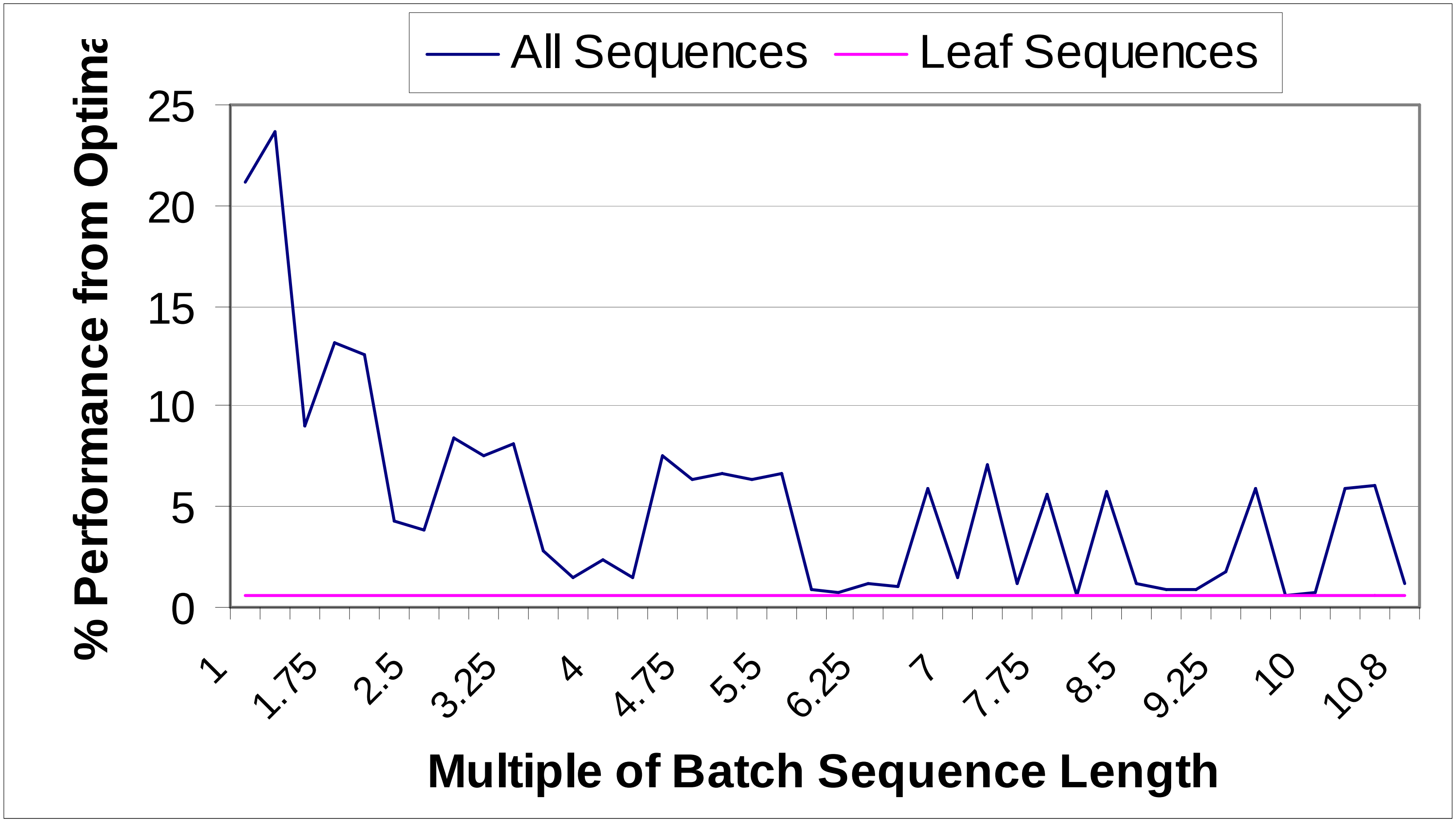
- Simulated annealing parameters
  - sequence length → 1.5 times batch length
  - initial temperature → 0.5 to 0.95
  - cooling schedule → 0.5 to 0.95, steps 0.5
- Experimental results
  - perf. 0.15% from optimal, std. dev. of 0.13%
  - avg. perf. 15.95% worse, std. dev. of 0.55%
  - 41.06% iter. reach optimal, std. dev. of 0.81%

Random Algorithm

- Random sampling used for search spaces that are discrete and sparse



Random Algorithm – Performance



Random Algorithm – Cost