Heuristic Search for Large Problems with Real Costs

Matthew Hatem, Ethan Burns and Wheeler Ruml

University of New Hampshire

with support from NSF (grant IIS-0812141) and DARPA (grant N10AP20029)

Motivation

- Problem: A* runs out of memory in $\sim$10 minutes

Motivation

- Problem: A* runs out of memory in $\sim$10 minutes
- Problem: IDA* fails with many duplicates and real costs

Motivation

- Problem: A* runs out of memory in ∼10 minutes
- Problem: IDA* fails with many duplicates and real costs
- Solution: A* on disk (“external memory”)

Motivation

- Problem: A* runs out of memory in $\sim$10 minutes
- Problem: IDA* fails with many duplicates and real costs
- Solution: A* on disk ("external memory")

- Problem: Previous methods assume integer costs

Motivation

- Problem: A* runs out of memory in $\sim$10 minutes
- Problem: IDA* fails with many duplicates and real costs
- Solution: A* on disk ("external memory")

- Problem: Previous methods assume integer costs
- Problem: Most previous methods use breadth-first search

Motivation

- Problem: A* runs out of memory in ∼10 minutes
- Problem: IDA* fails with many duplicates and real costs
- Solution: A* on disk ("external memory")

- Problem: Previous methods assume integer costs
- Problem: Most previous methods use breadth-first search
- Solution: PEDAL (this paper)
  - Best-first search order
  - Real costs
  - Provably I/O efficient
  - Exploits parallelism

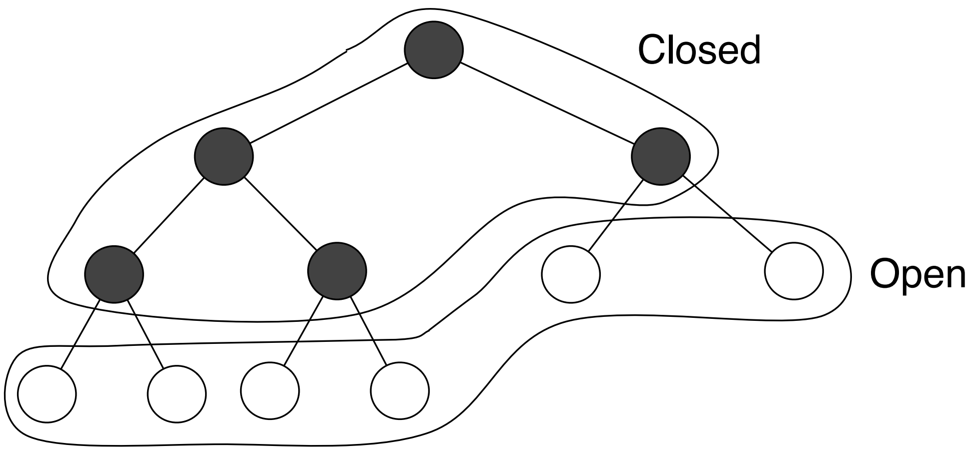
Outline

- External memory search
- The problem with real costs
- PEDAL
- Results

External Memory Search

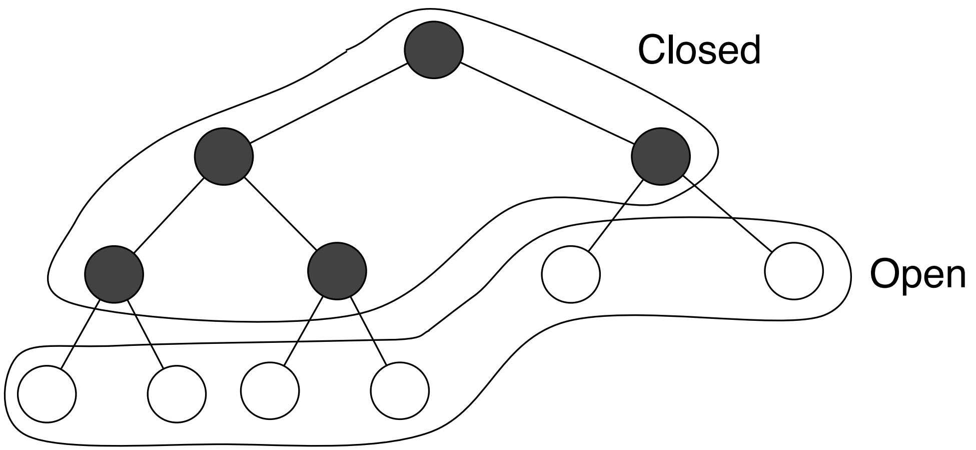
A* Search

- Open List: ordered set $f(n) = g(n) + h(n)$
- Closed List: random access!
- How to put this on disk?



A* Search

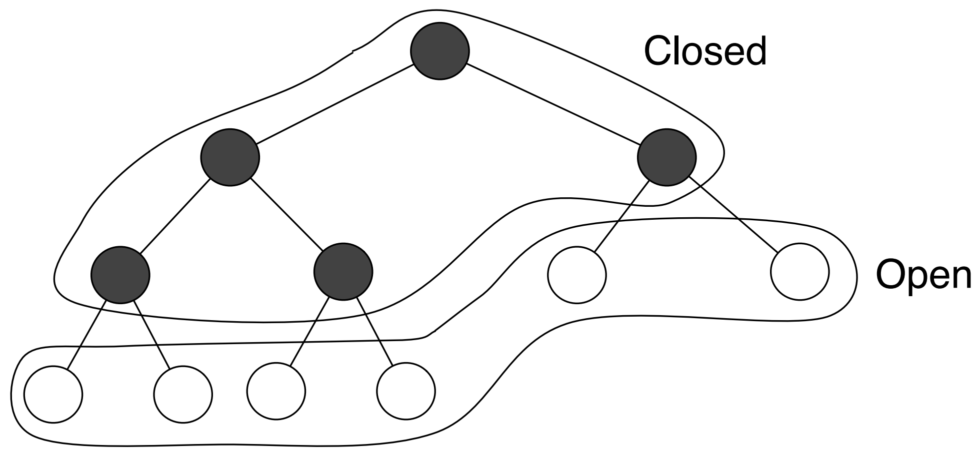
- Open List: ordered set $f(n) = g(n) + h(n)$
- Closed List: random access!
- How to put this on disk?



- Two ideas: Buckets and Layers

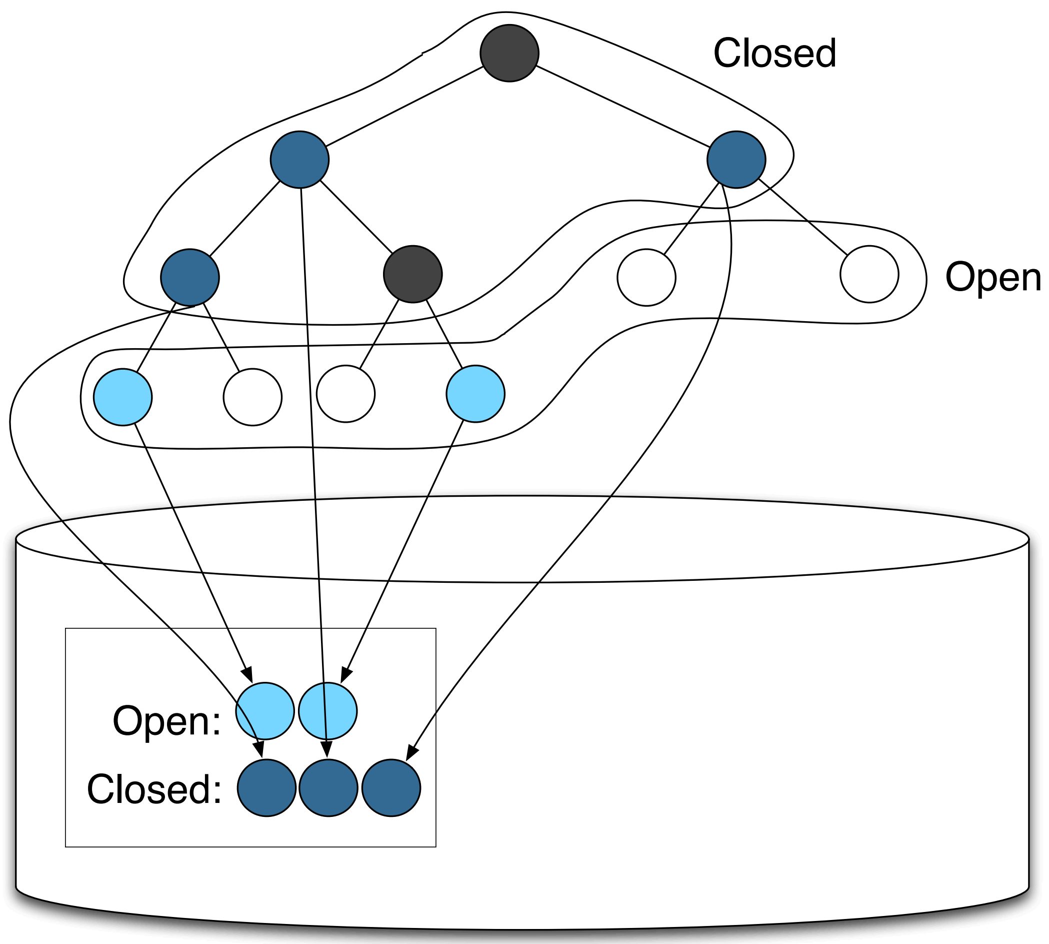
Buckets

- Use a hash function to partition the space
- Duplicate nodes will be in the same bucket



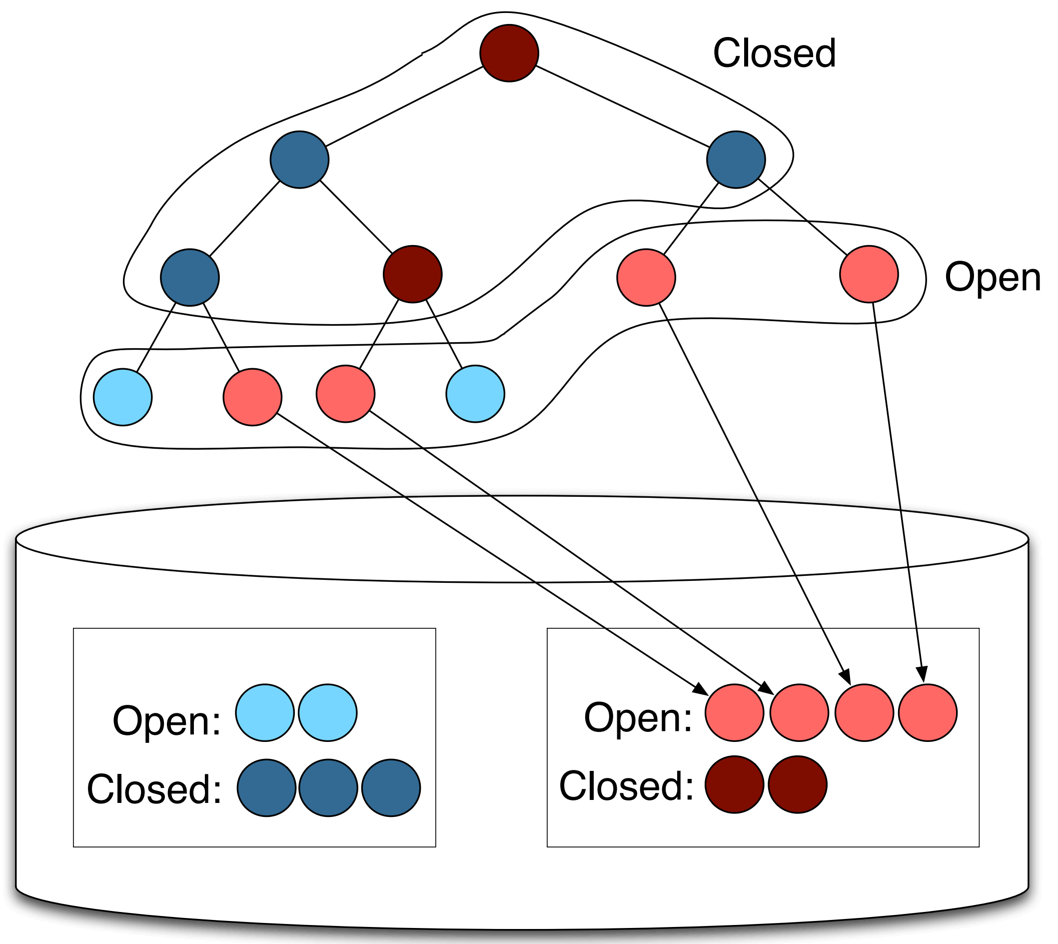
Buckets

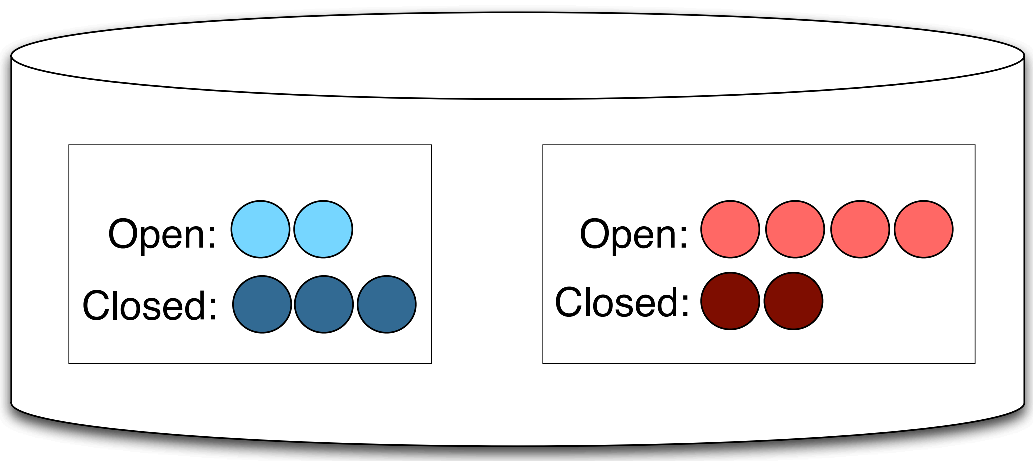
- Use a hash function to partition the space
- Duplicate nodes will be in the same bucket



Buckets

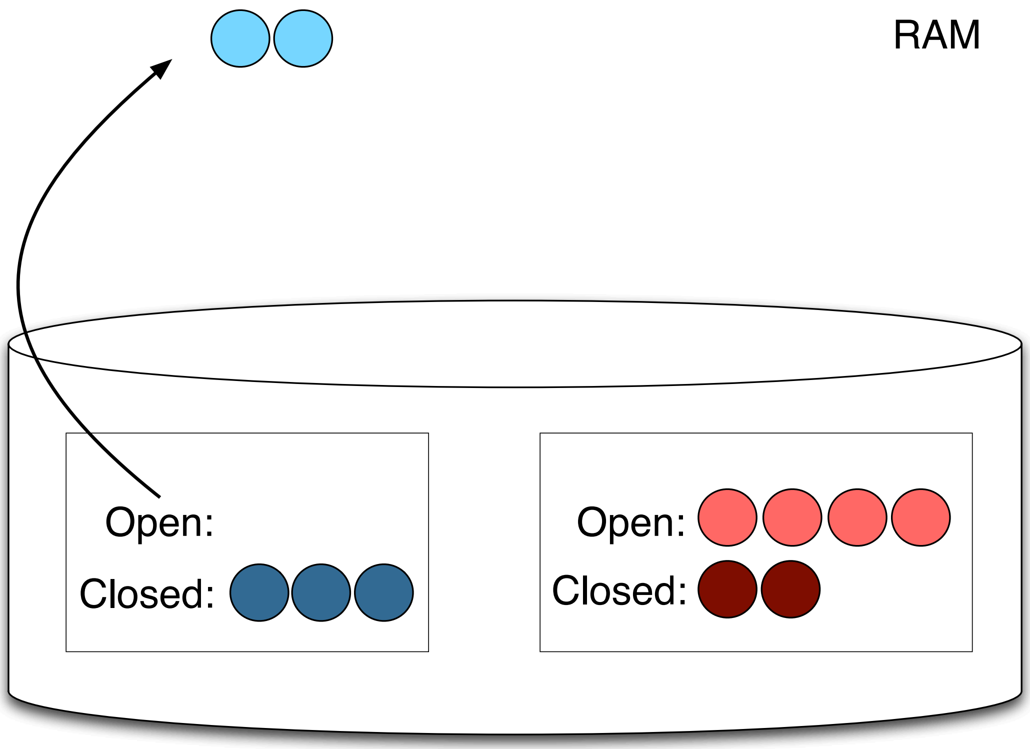
- Use a hash function to partition the space
- Duplicate nodes will be in the same bucket





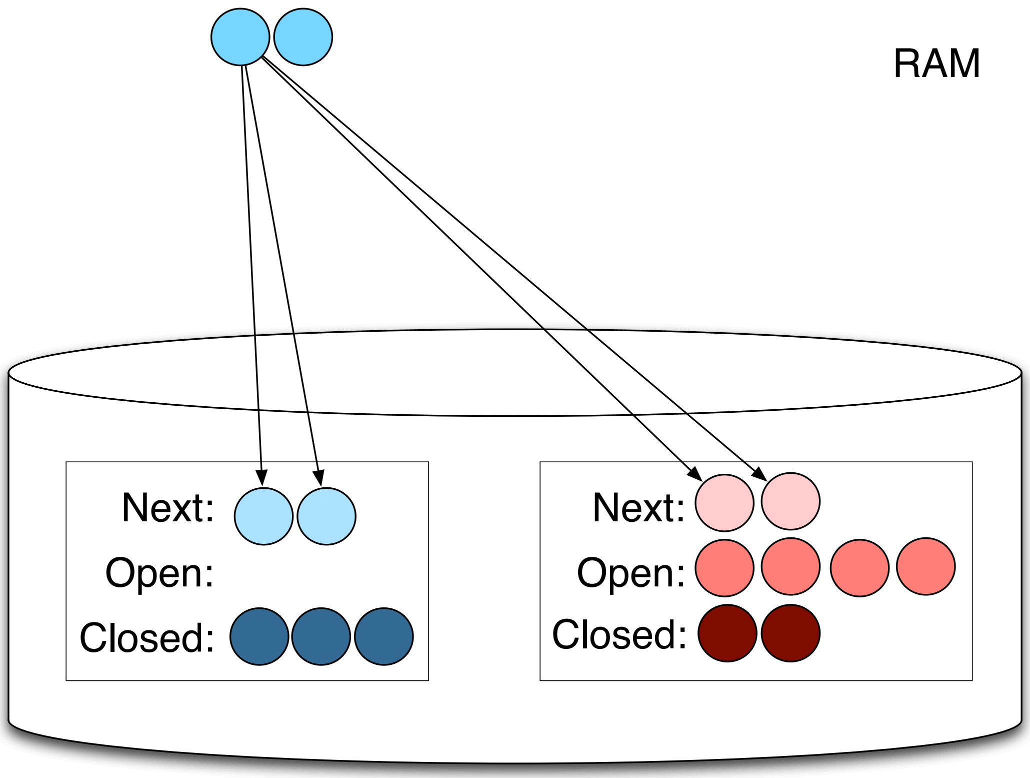
Only one bucket need fit in RAM to expand

Expand Phase



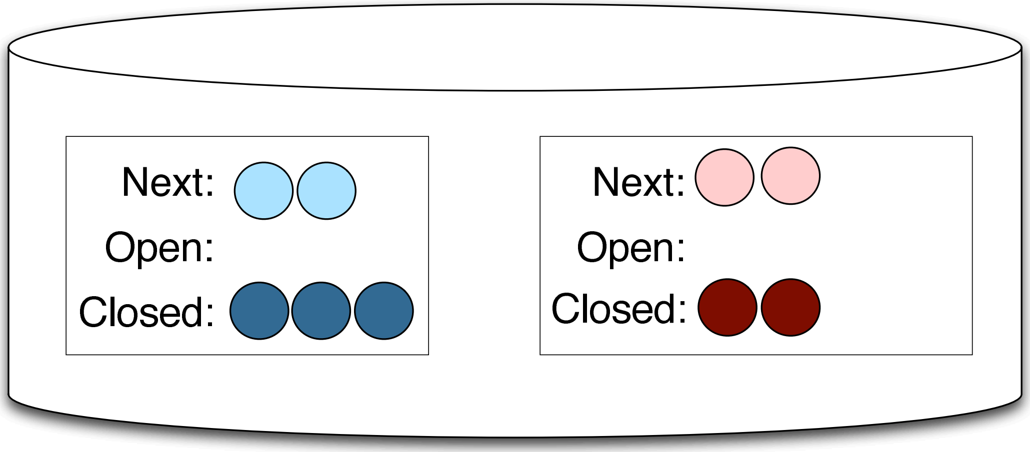
Only one bucket need fit in RAM to expand

Expand Phase

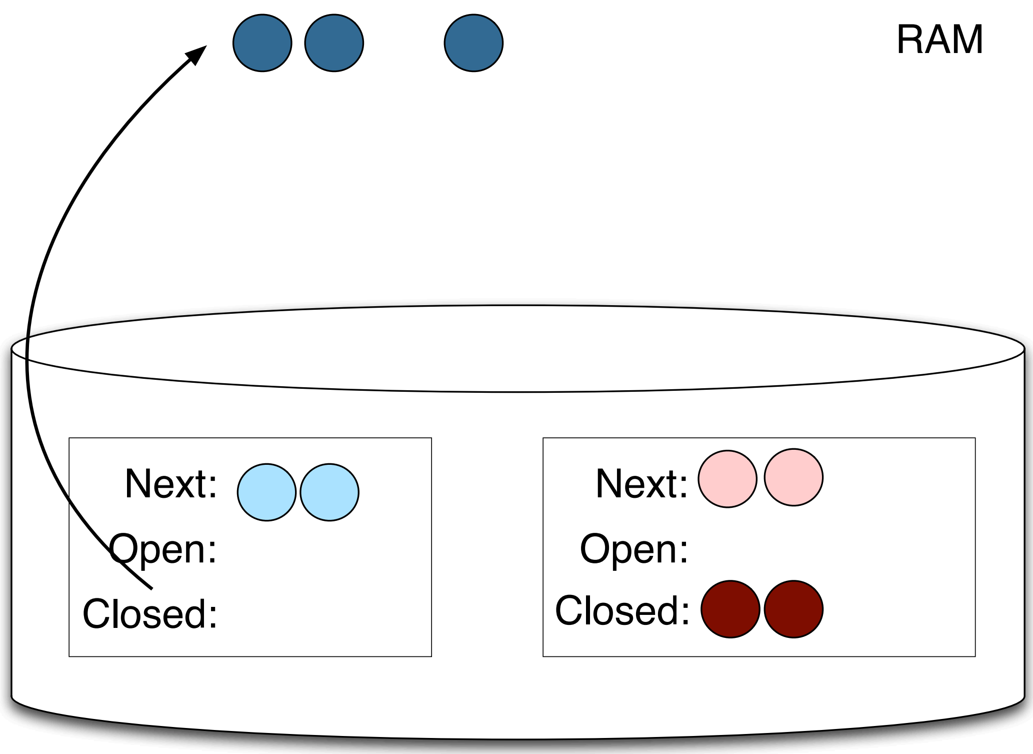


Only one bucket need fit in RAM to expand

Merge Phase

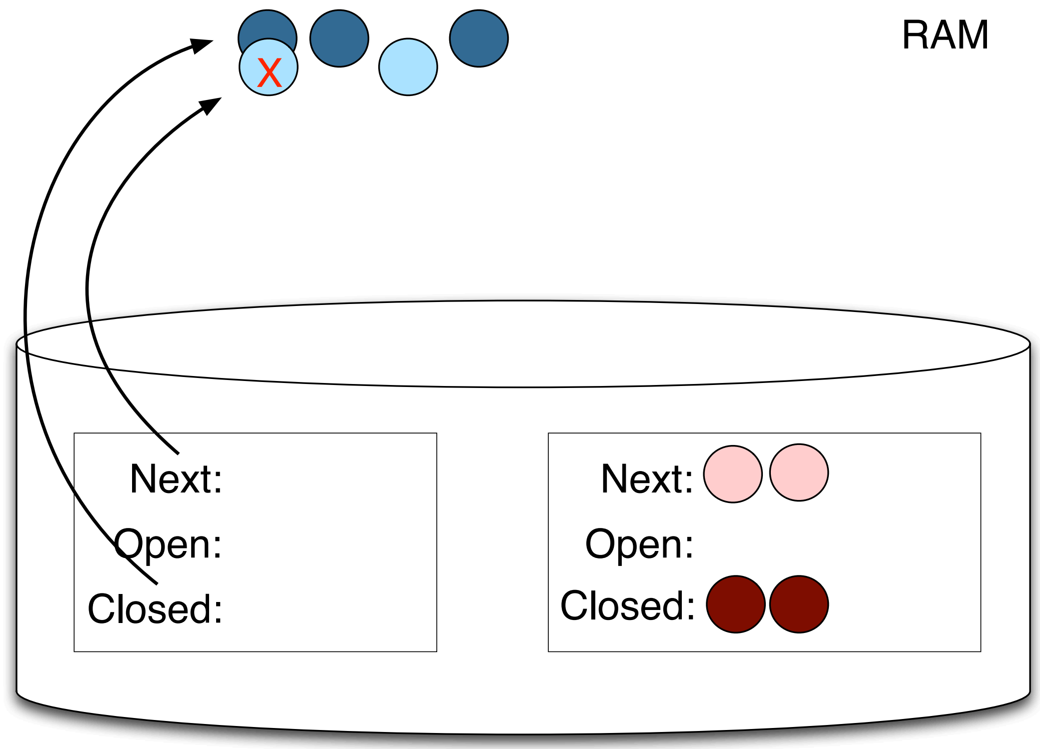


Only one bucket need fit in RAM to merge duplicates



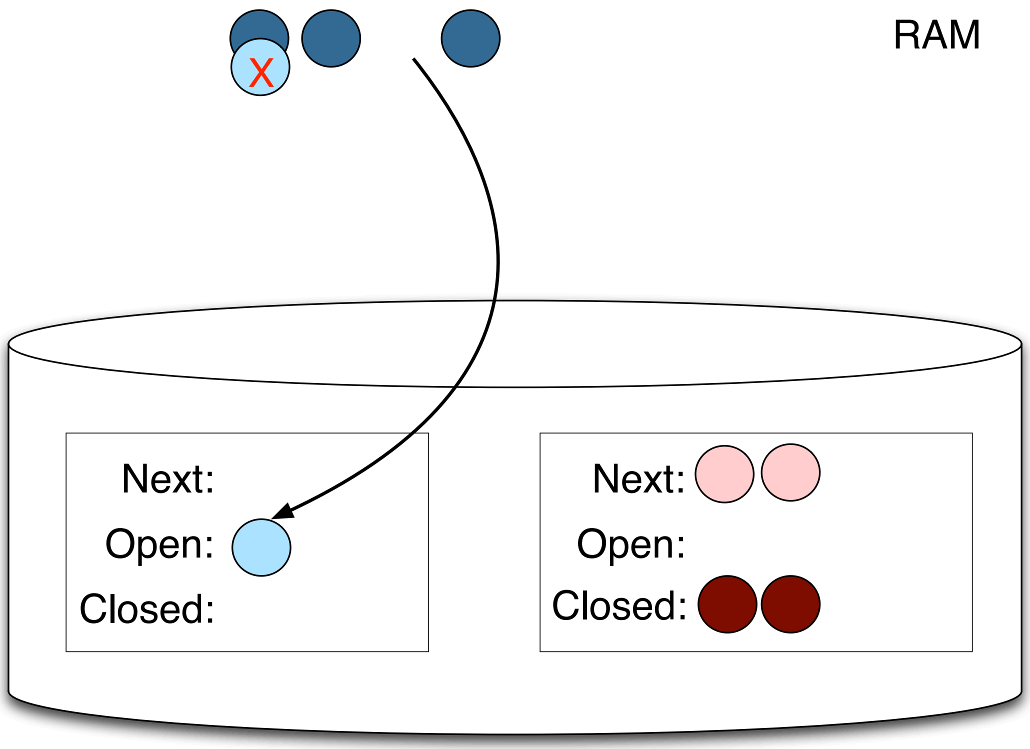
Only one bucket need fit in RAM to merge duplicates

Merge Phase



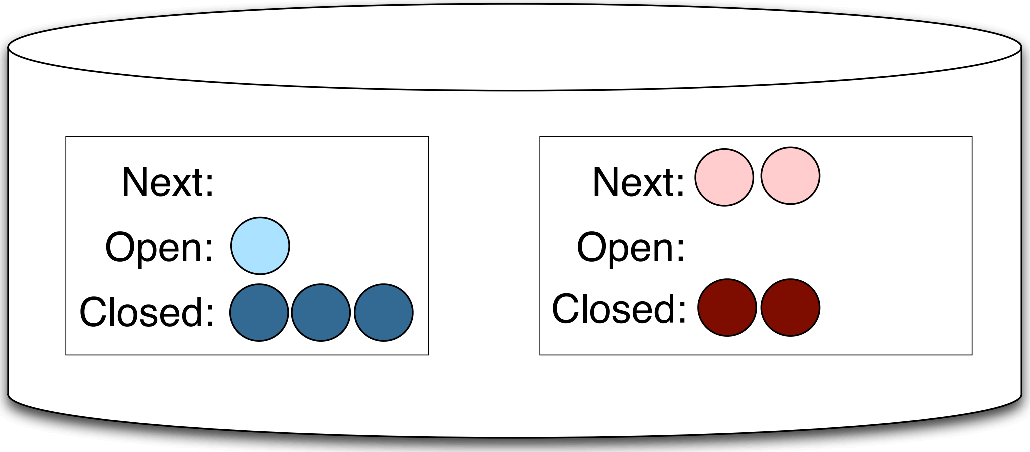
Only one bucket need fit in RAM to merge duplicates

Merge Phase



Only one bucket need fit in RAM to merge duplicates

Merge Phase



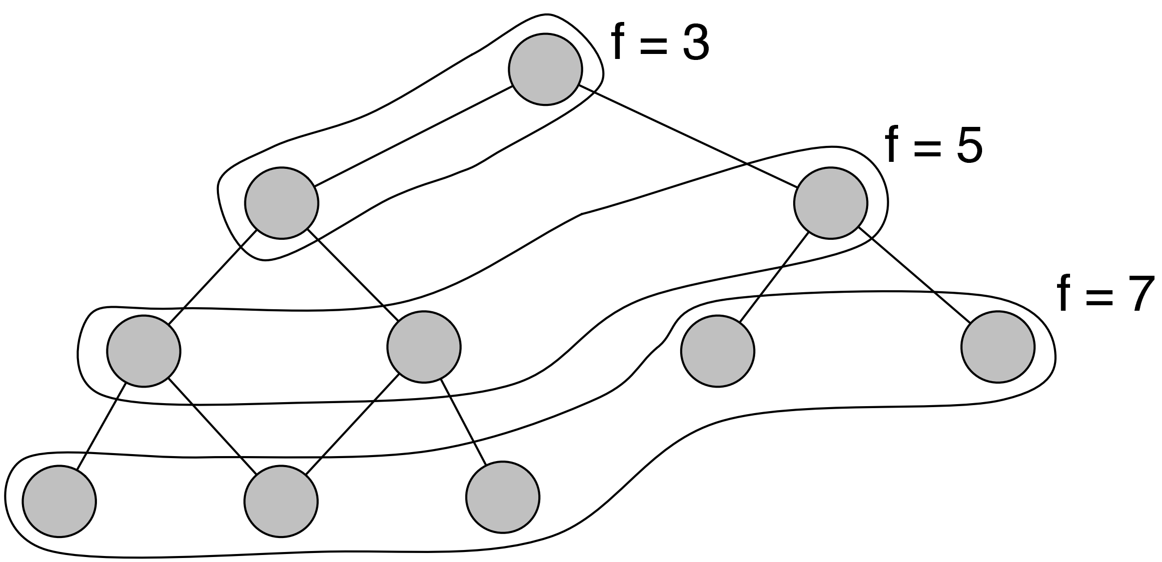
Only one bucket need fit in RAM to merge duplicates

Layers

- Depth based layers give breadth-first search
  eg., Breadth-First Heuristic Search (Zhou, Hansen 2004)
- How do we perform best-first search?

Layers

- Depth based layers give breadth-first search
  eg., Breadth-First Heuristic Search (Zhou, Hansen 2004)
- How do we perform best-first search?
  - Set an f limit and expand all nodes within the limit



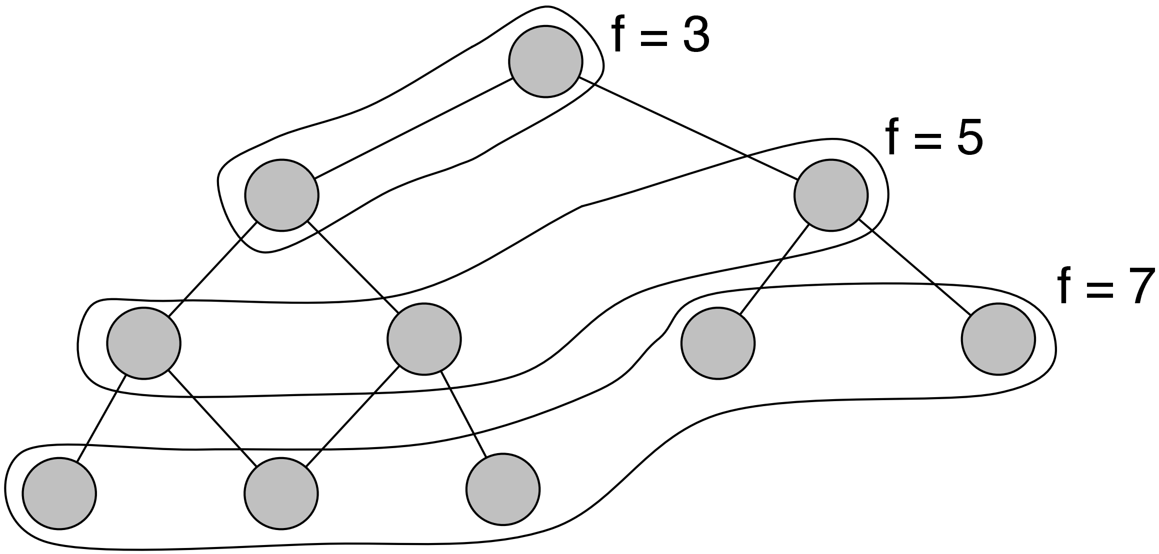
Summary

- Divide the search space into buckets
- Duplicate detection is localized per bucket
- Duplicate detection is done in a separate phase
- Layer the search by f

PEDAL

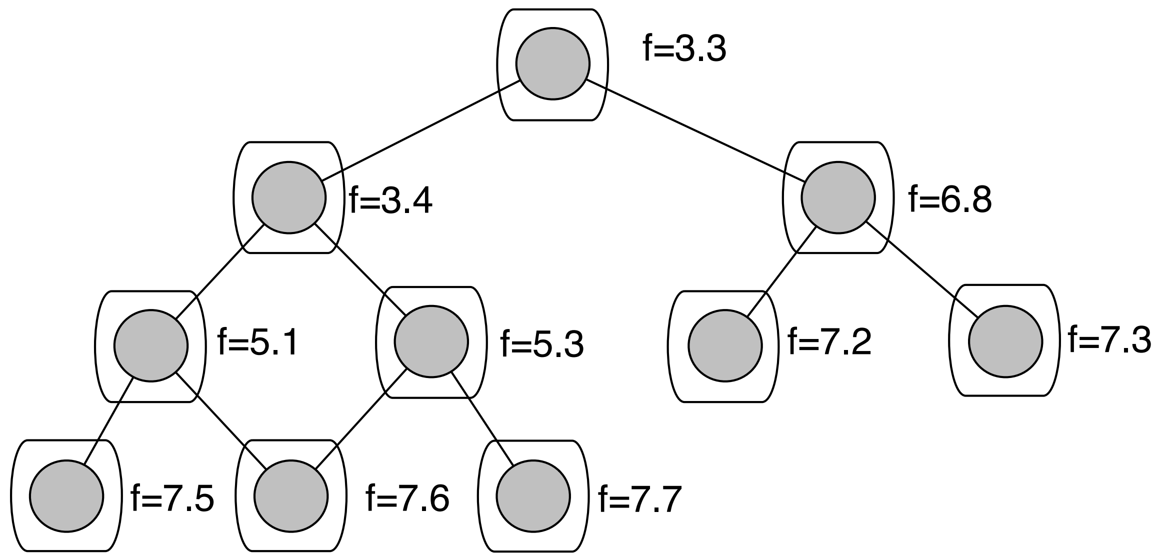
Problem with Real Costs

Unit Costs



Layers grow, many nodes per layer

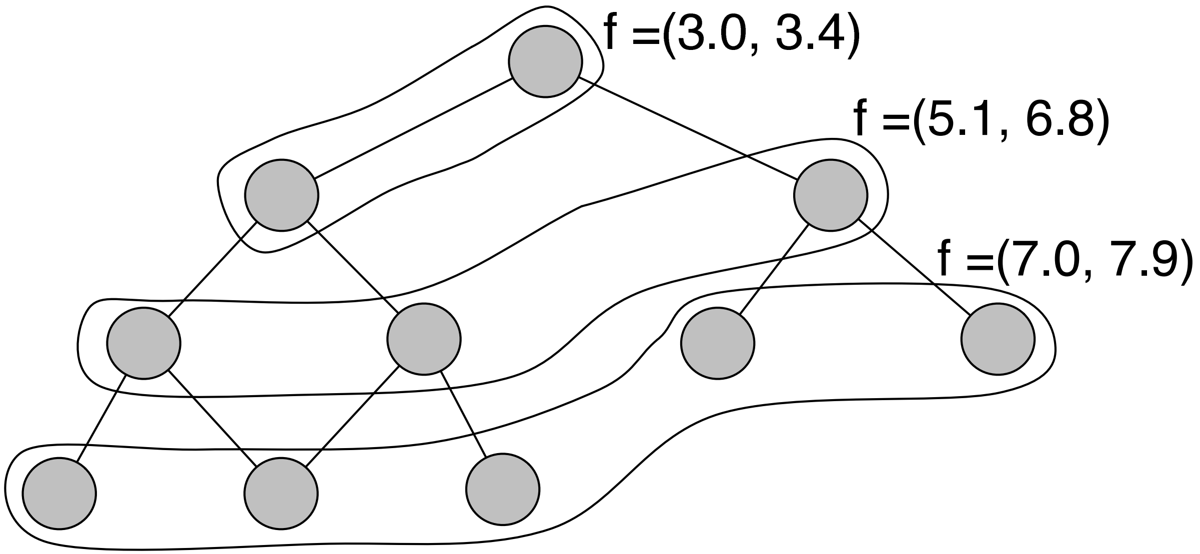
Real Costs



Many layers, too few nodes per layer

Problem with Real Costs

Real Costs



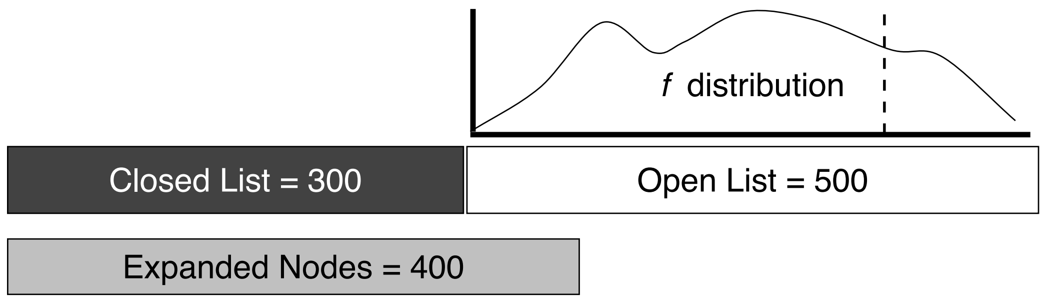
Our solution is to inflate layers
But how should we inflate layers?

Using a Histogram to Inflate Layers

Bad

Using a Histogram to Inflate Layers

Good



- Keep a distribution of all f-values on the frontier
- Choose a value in this distribution to guarantee I/O efficiency
- See paper for proof

Parallel External Dynamic A* Layering

- Best-first:
  - layer search on f
- Real Costs:
  - dynamically inflate f layers for I/O efficiency
- Exploits Parallelism (see paper)
- Recursive expansions (see paper)

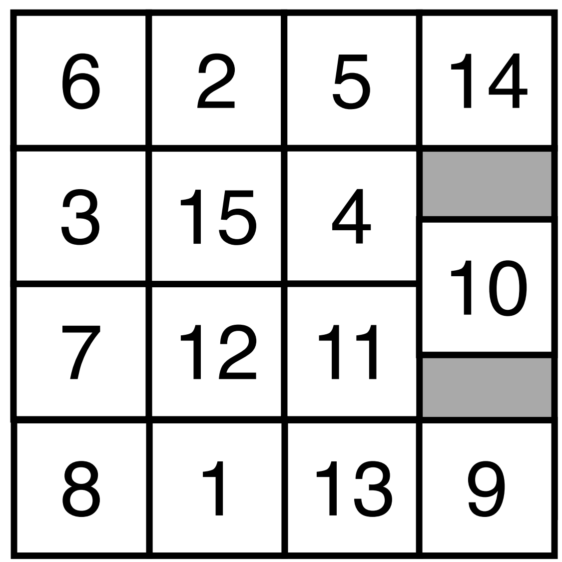
Experiments

Setup

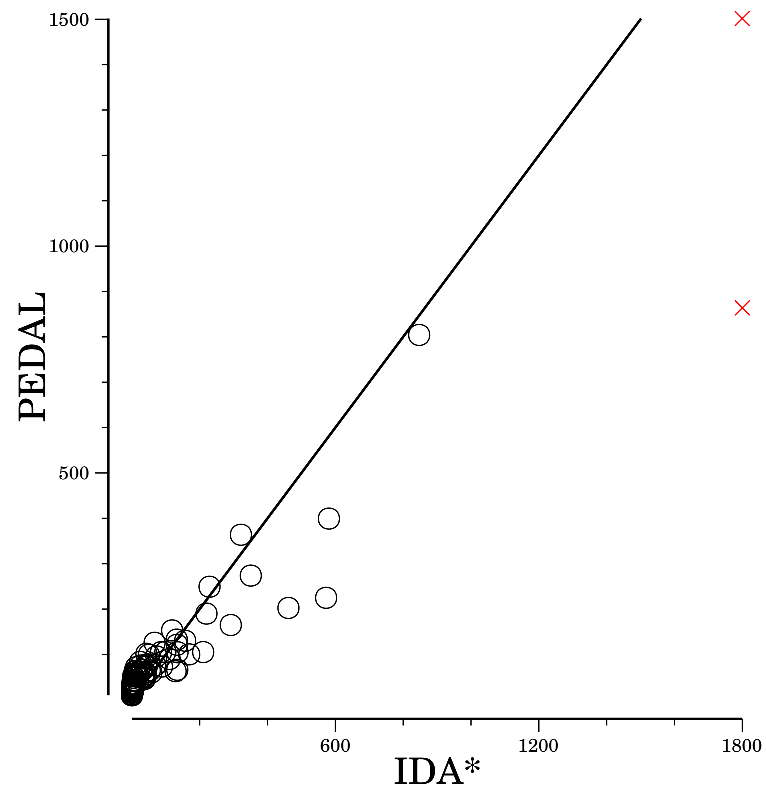
- Three domains
- Dual quad-core
- 8GB of RAM
- 7 SATA disks

15-Puzzle

- Unit cost sliding tile puzzle
- Classic benchmark
- Korf 100 (Korf 1985)
- A* is unable to solve all 100 with Manhattan Distance

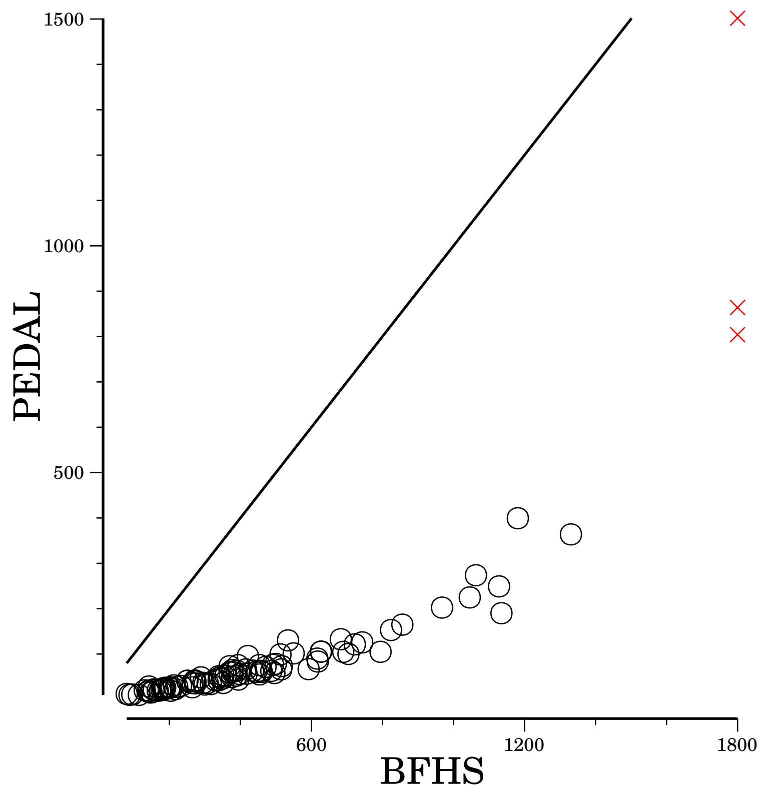


15-Puzzle:PEDAL vs. IDA* (Korf 1985)



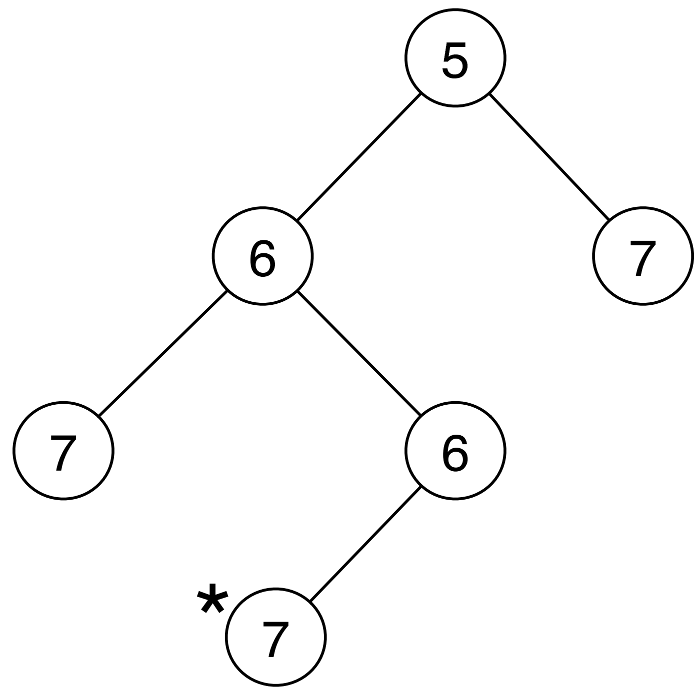
PEDAL uses disk but is faster than IDA*!

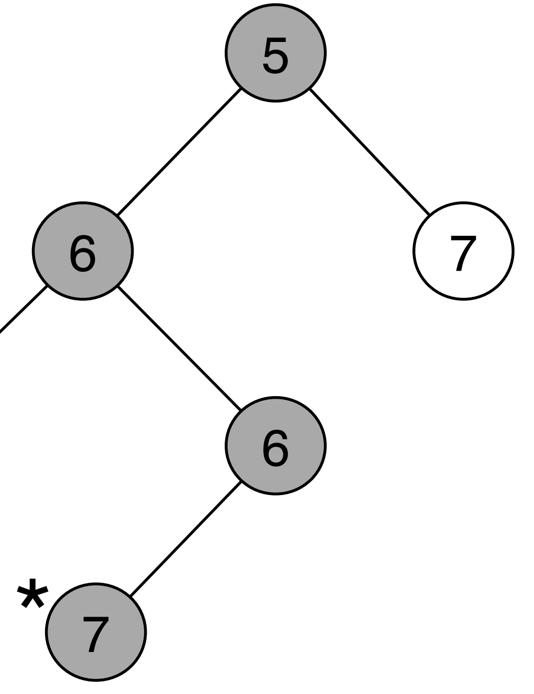
15-Puzzle: PEDAL vs. Breadth-First Heuristic Search



Best-first is better than breadth-first

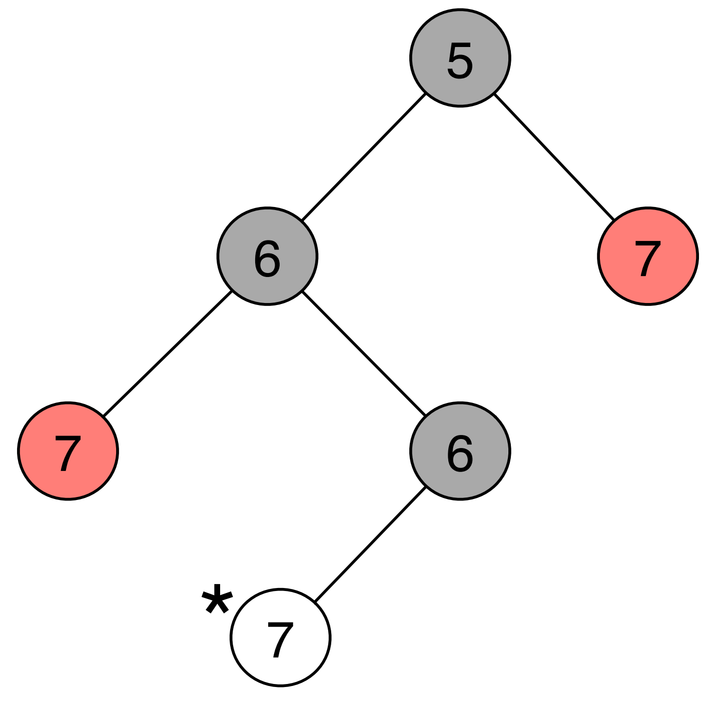
BFHS





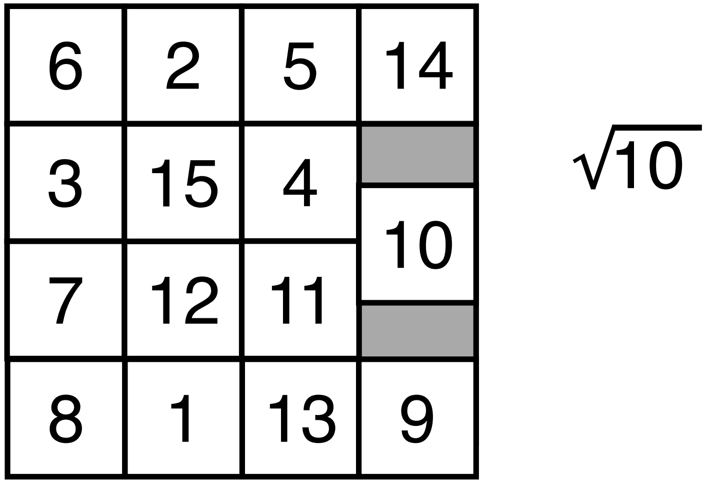
A* with proper tie breaking

BFHS



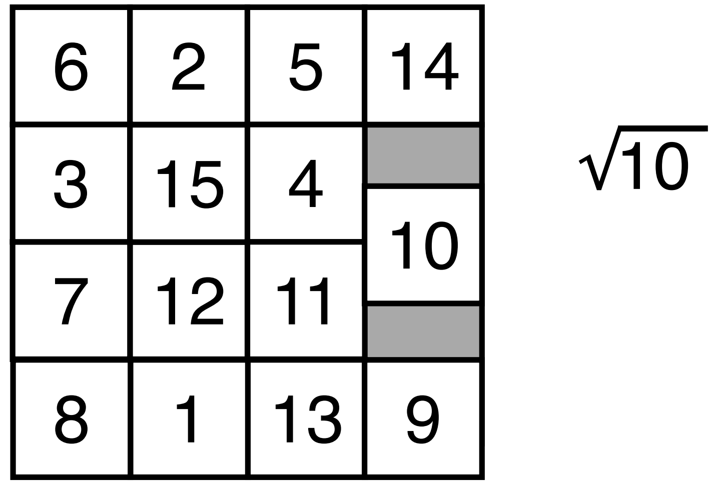
BFHS must expand more nodes

Sqrt Puzzle



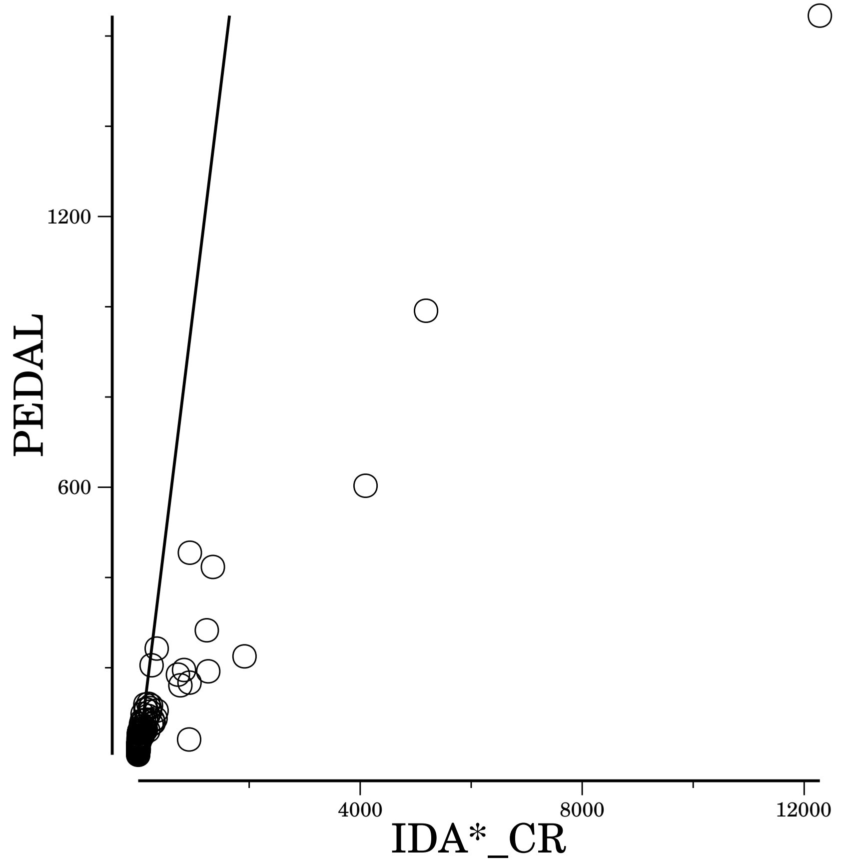
- Move costs square root of tile number
- Simple real-valued version of well understood benchmark
- Easy to reproduce

Sqrt Puzzle



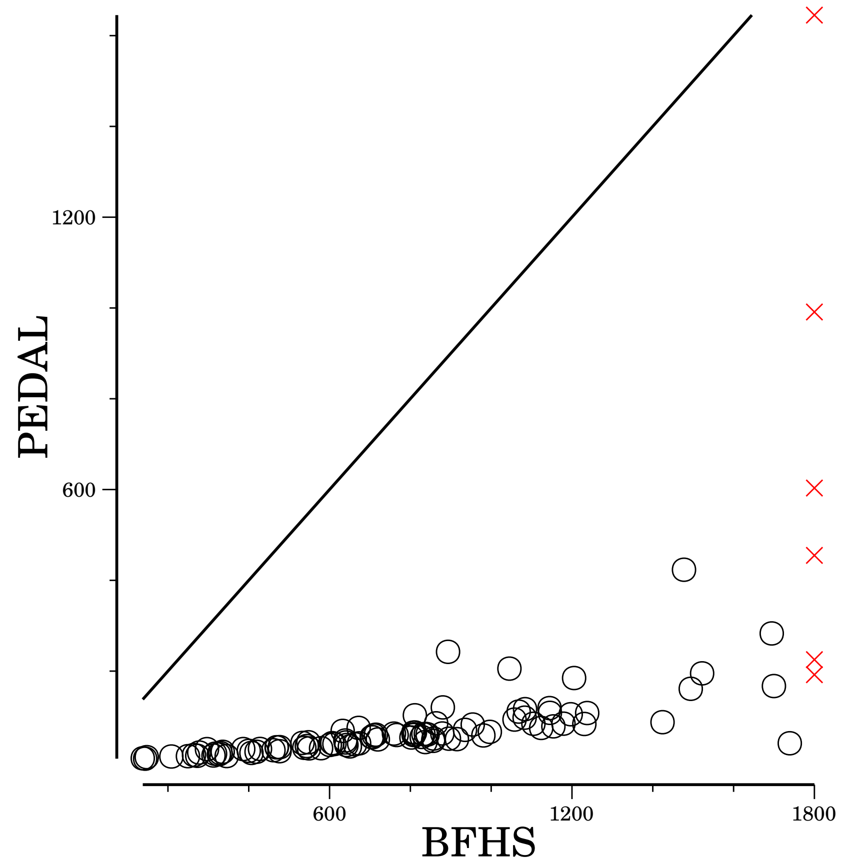
- IDA* and BFHS did not solve any instances in time limit
- We compare against IDA*$_{CR}$ and novel variant of BFHS

Sqrt. 15-Puzzle: PEDAL vs. IDA*$_{CR}$



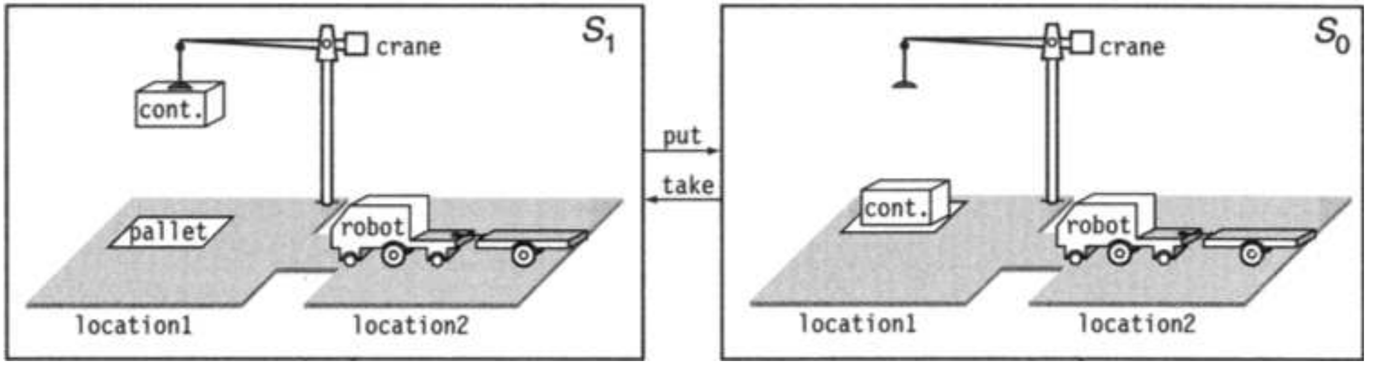
Again duplicate checking and parallelism trump disk latency

Sqrt. 15-Puzzle: PEDAL vs. BFHS



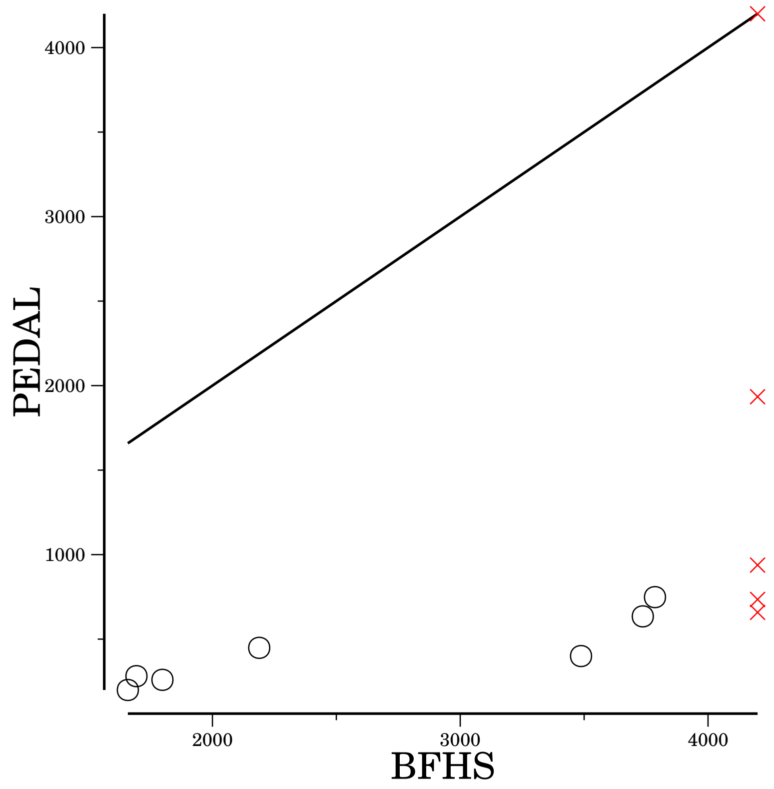
Again best-first is better than breadth-first

Dockyard Robots



- From Ghallab, Nau, Traverso (2004)
- All actions have real costs
- Many duplicate states
- IDA* and IDA*$_{CR}$ fail on all instances

Dockyard Robots: PEDAL vs. BFHS



PEDAL is the only viable alternative

Summary

- Previous external-memory search:
  - fails on domains with real costs
  - suggests that breadth-first search is preferred
- PEDAL:
  - is best-first
  - I/O efficient for real costs

Summary

- Previous external-memory search:
  - fails on domains with real costs
  - suggests that breadth-first search is preferred
- PEDAL:
  - is best-first
  - I/O efficient for real costs

External memory search doesn't have to be slow!

  - faster than IDA* on standard 15-Puzzle!

Summary

- Previous external-memory search:
  - fails on domains with real costs
  - suggests that breadth-first search is preferred
- PEDAL:
  - is best-first
  - I/O efficient for real costs

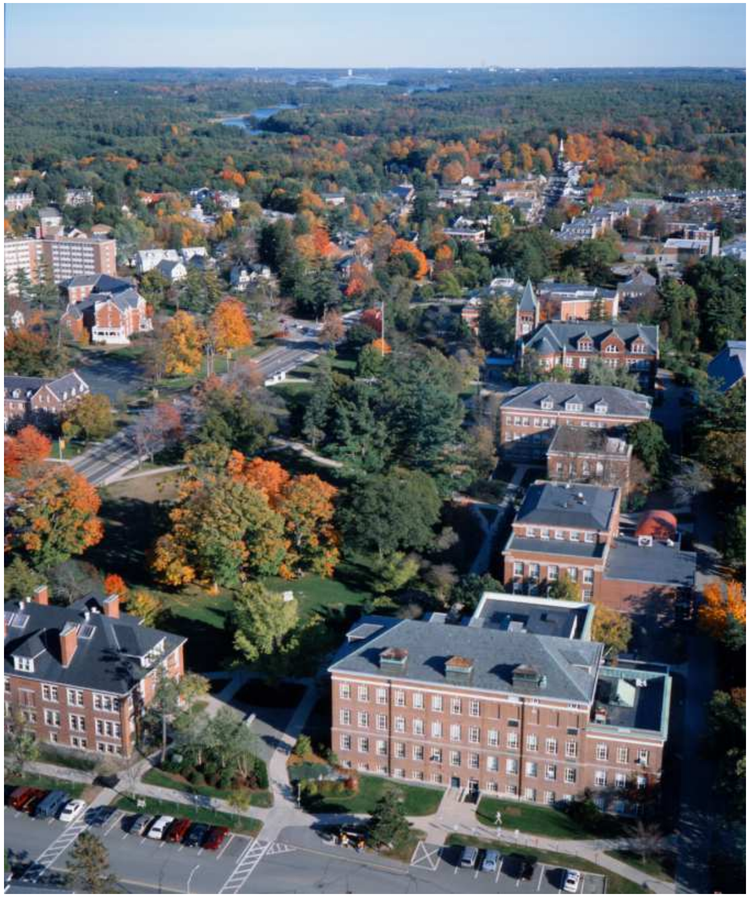
  External memory search doesn't have to be slow!

  - faster than IDA* on standard 15-Puzzle!

  General purpose best-first external memory algorithm

The University of New Hampshire

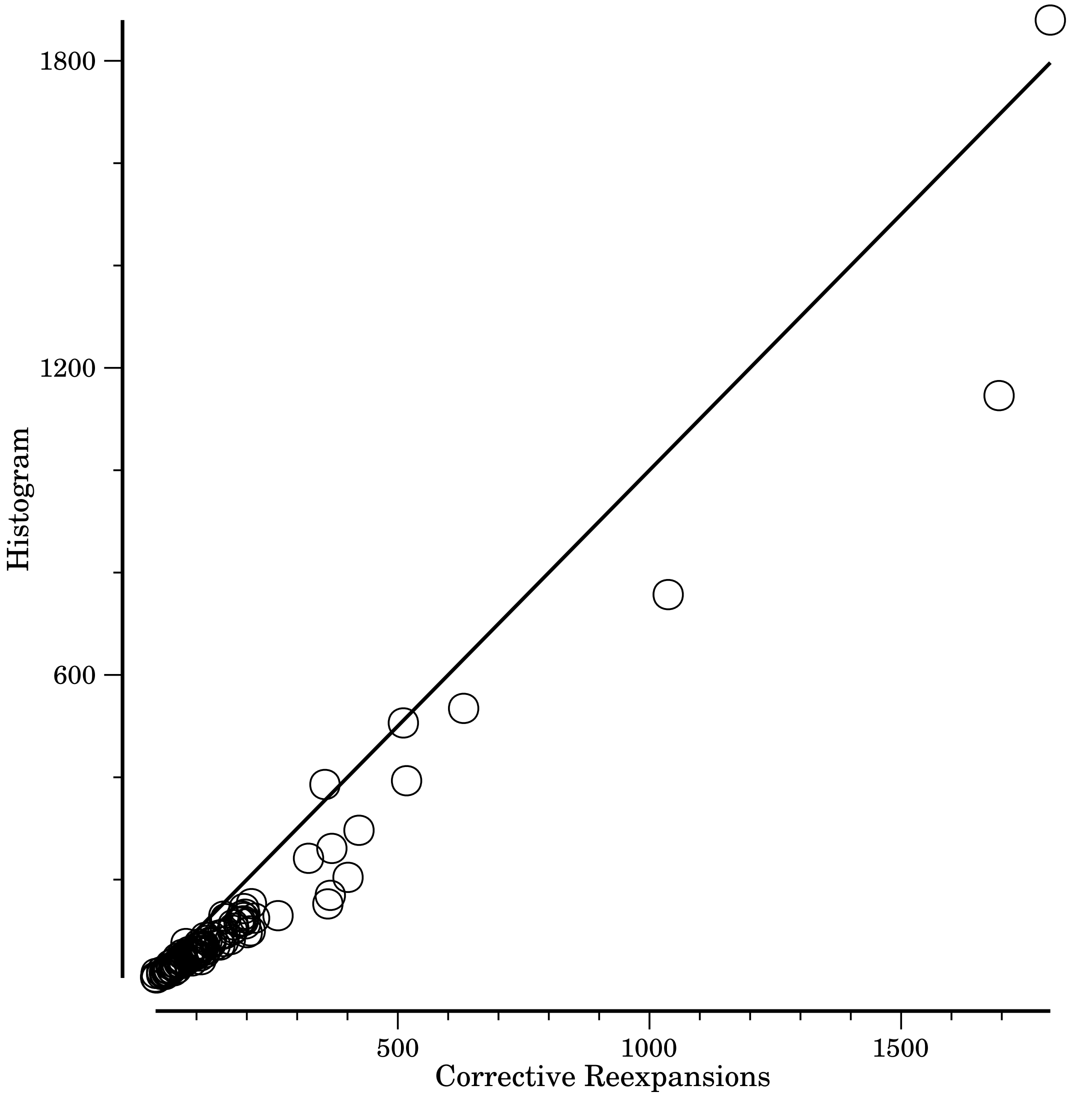
Tell your students to apply to grad school in CS at UNH!



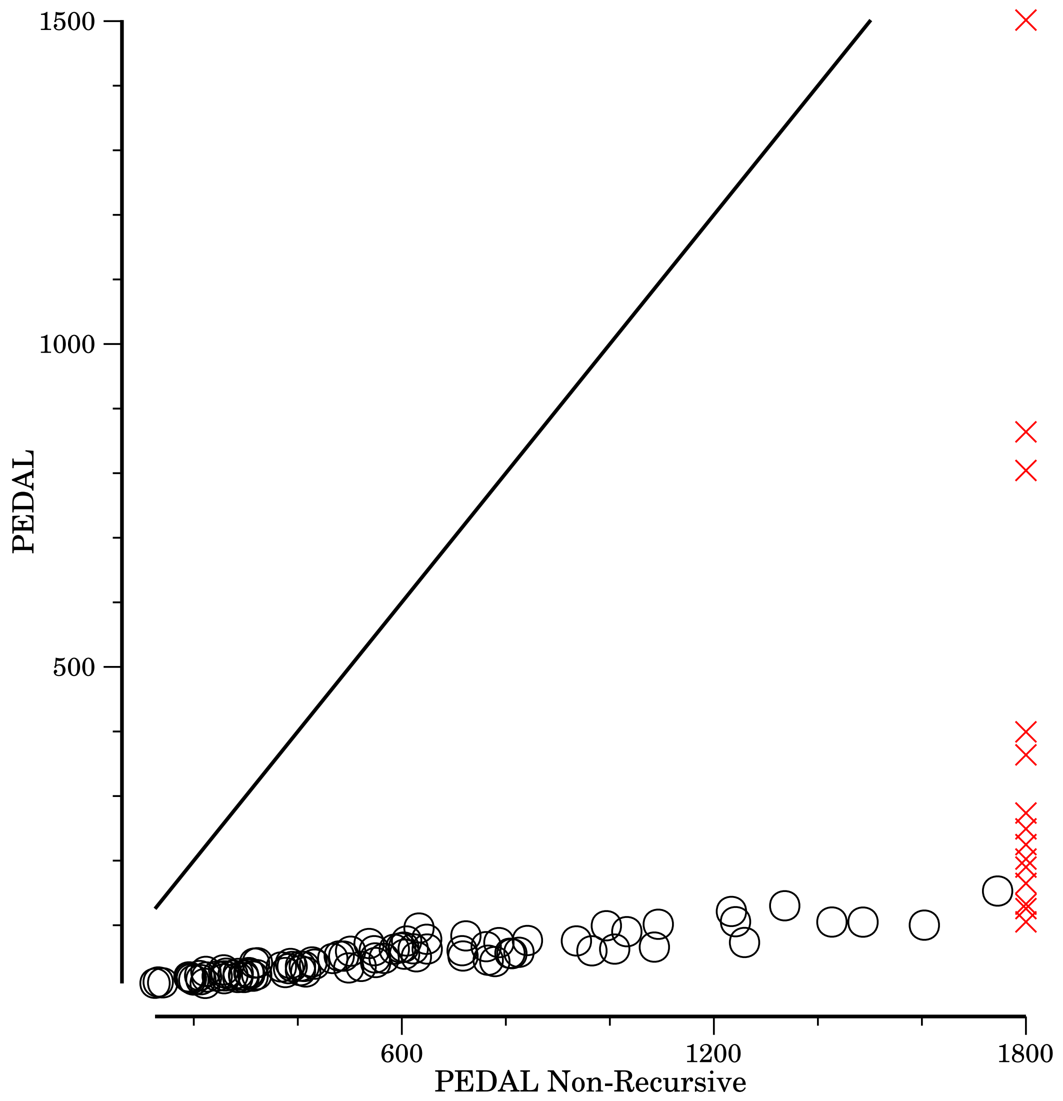
- friendly faculty
- funding
- individual attention
- beautiful campus
- low cost of living
- easy access to Boston, White Mountains
- strong in AI, infoviz, networking, systems, bioinformatics

Back-up Slides

PEDAL CR vs. Histogram



Recursive Expansions



BFHS vs. PEDAL Non-Recursive