Using Distance Estimates In Heuristic Search

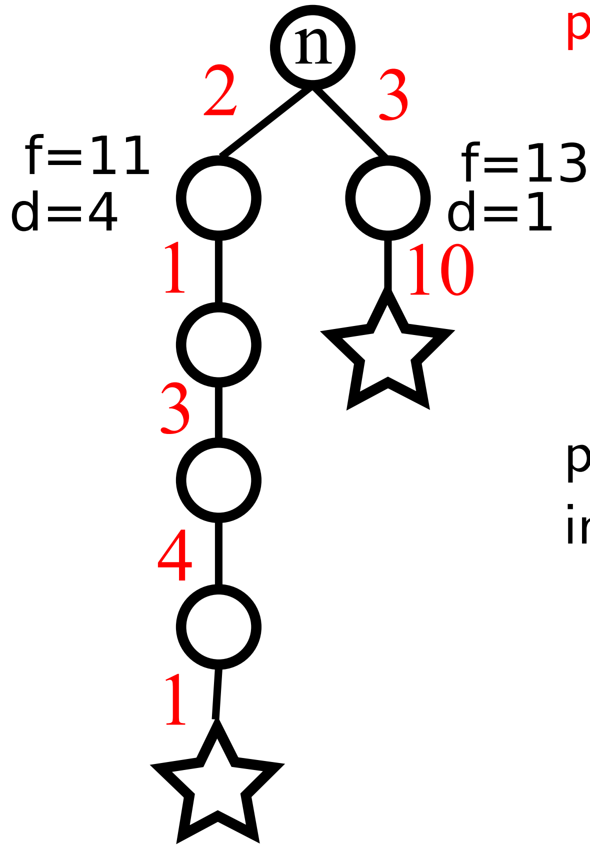
Jordan T. Thayer and Wheeler Ruml

UNIVERSITY *of* NEW HAMPSHIRE

`jtd7`, `ruml` at `cs.unh.edu`

slides at: http://www.cs.unh.edu/~jtd7/papers/

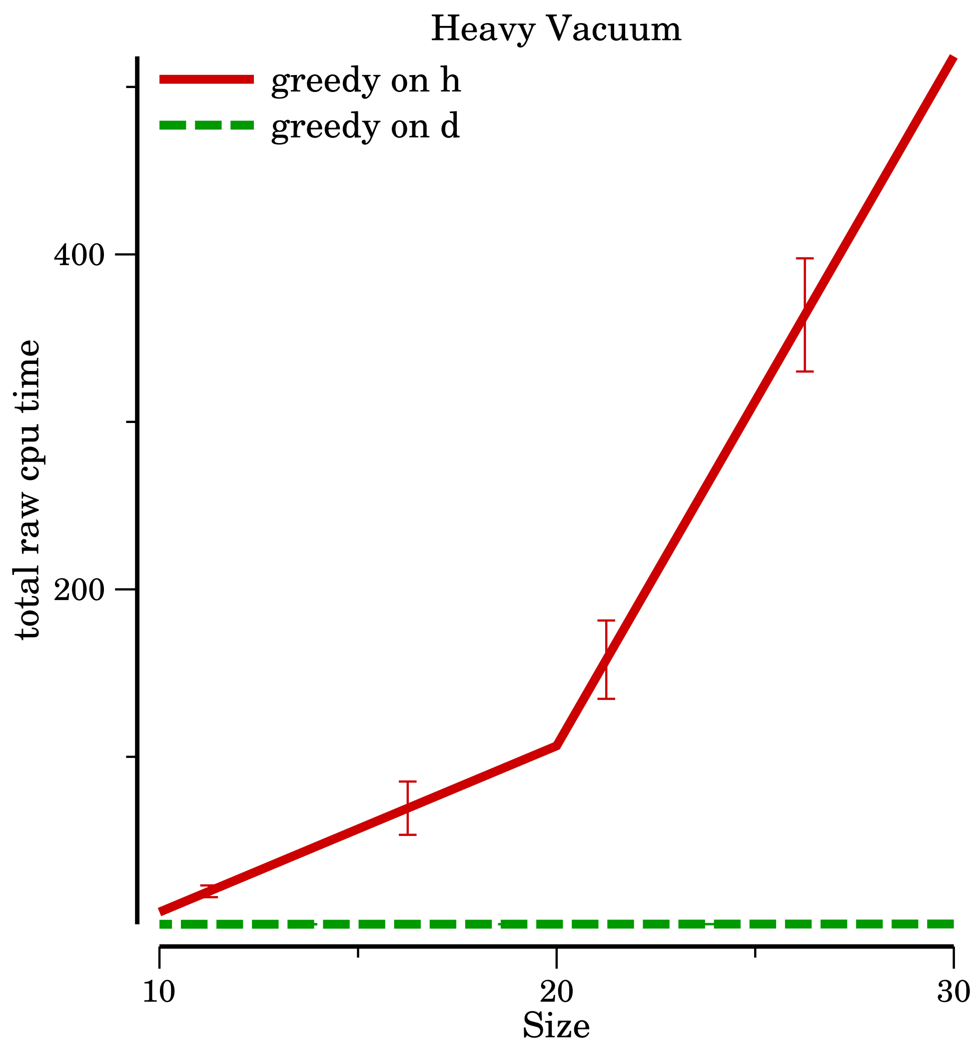
Use Distance Estimates



when actions have varying costs, plan cost and plan length can differ

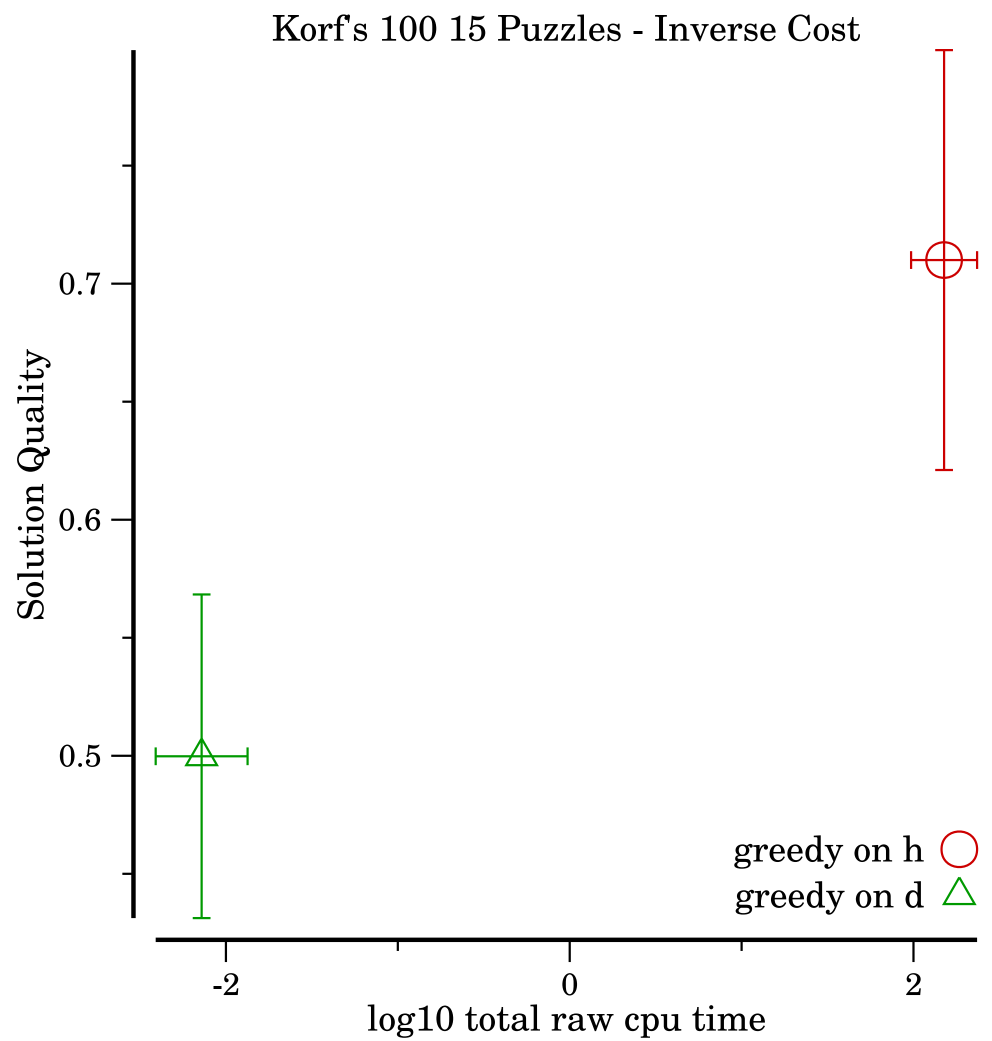
paying attention to the difference can improve search performance dramatically

Use Distance Estimates



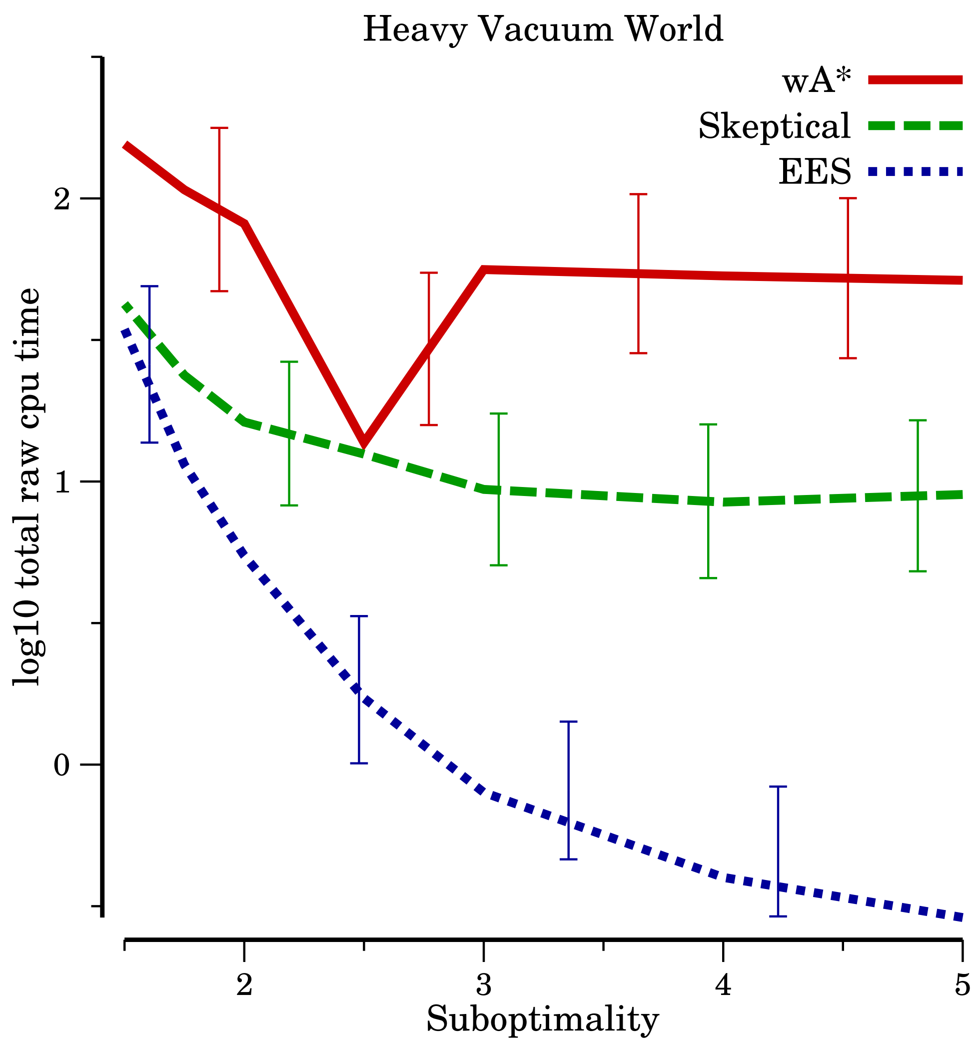
search on d scales better

Use Distance Estimates



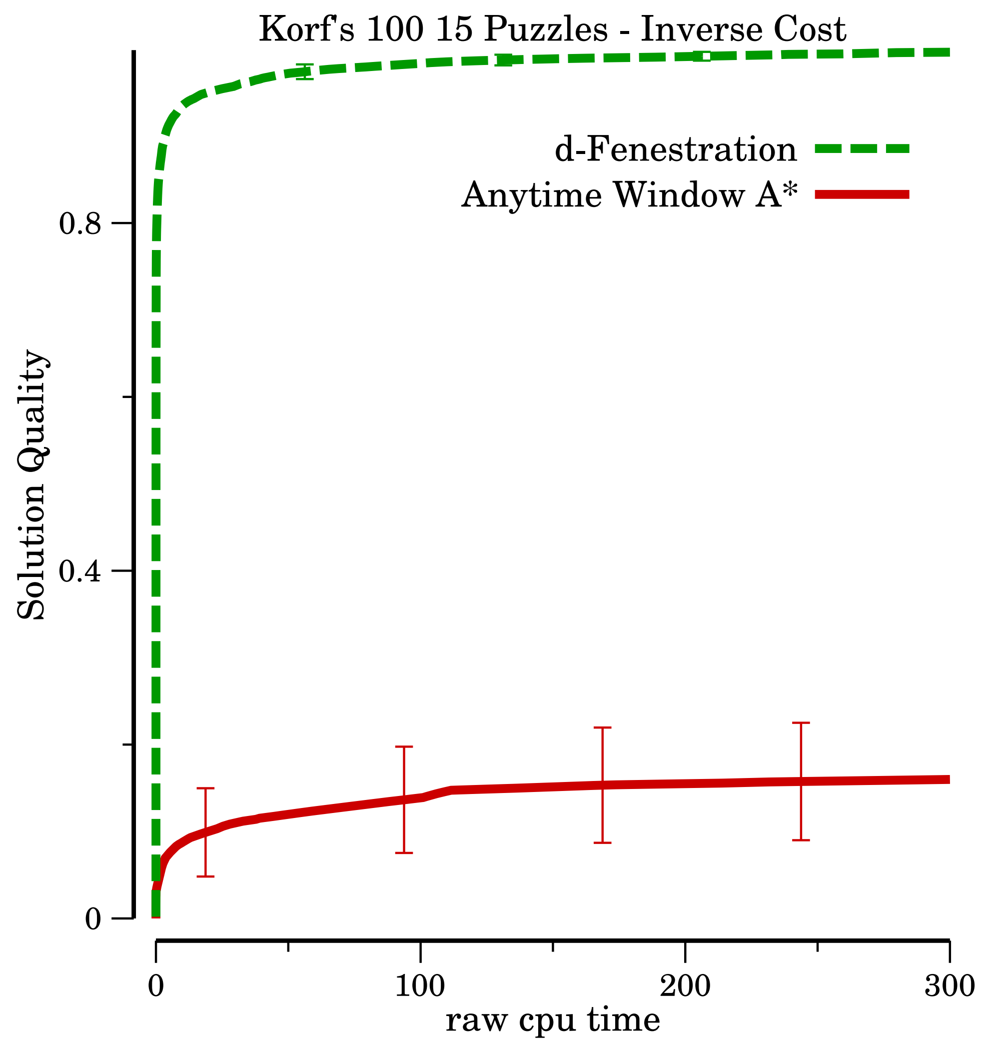
search on d can be several orders of magnitude faster

Use Distance Estimates



d can help in bounded suboptimal search

Use Distance Estimates



d improves the performance of anytime search as well

About the Tutorial

- Jordan and I will alternate
- bibliography at the end
- the pseudo code
  - not presented during talk
  - included for later review
- not discussed
  - optimal search strategies
  - bounded-depth tree search
  - local search strategies

Outline

- distance estimates

  differentiating, computing $d_{nearest}$ / $d_{cheapest}$

- $d(n)$ in suboptimal search

  best-first search on d, alternating, beam search on d

- $d(n)$ in bounded suboptimal search

  skeptical, A^*_ϵ, explicit estimation search

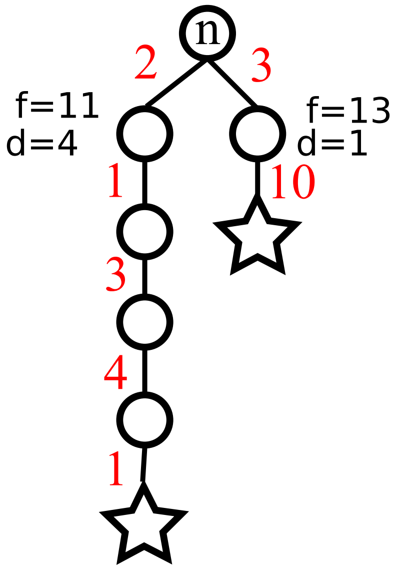
- $d(n)$ in anytime search

  d-fenestration, size-cost search, anytime frameworks

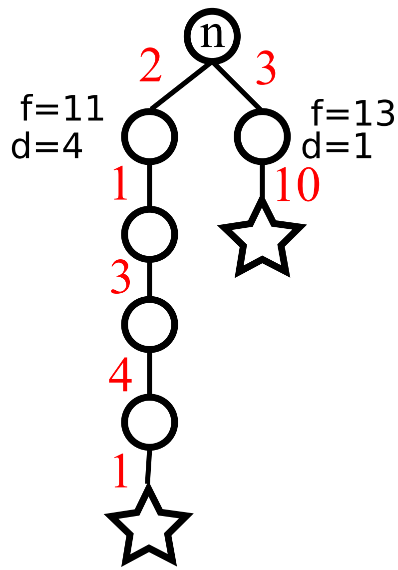
- summary and conclusions

$d(n)$

Near and Cheap



$$h^*(n) = 11$$
$$d^*(n) = ?$$



$$h^*(n) = 11$$
$$d^*_{cheapest}(n) = 5$$
$$d^*_{nearest}(n) = 2$$

$d_{nearest}$ is potentially independent of h,
h^* and $d^*_{cheapest}$ are related

h^* and $d^*_{cheapest}$ can be estimated simultaneously, saving effort
compute $d_{nearest}$ by ignoring cost information
d estimates don't have to be admissible

Using $d(n)$ for Suboptimal Search

Outline: Distance Estimates In Suboptimal Search

- best-first search on d: speedy search

    sorts on $d_{cheapest}$ to preserve some cost information

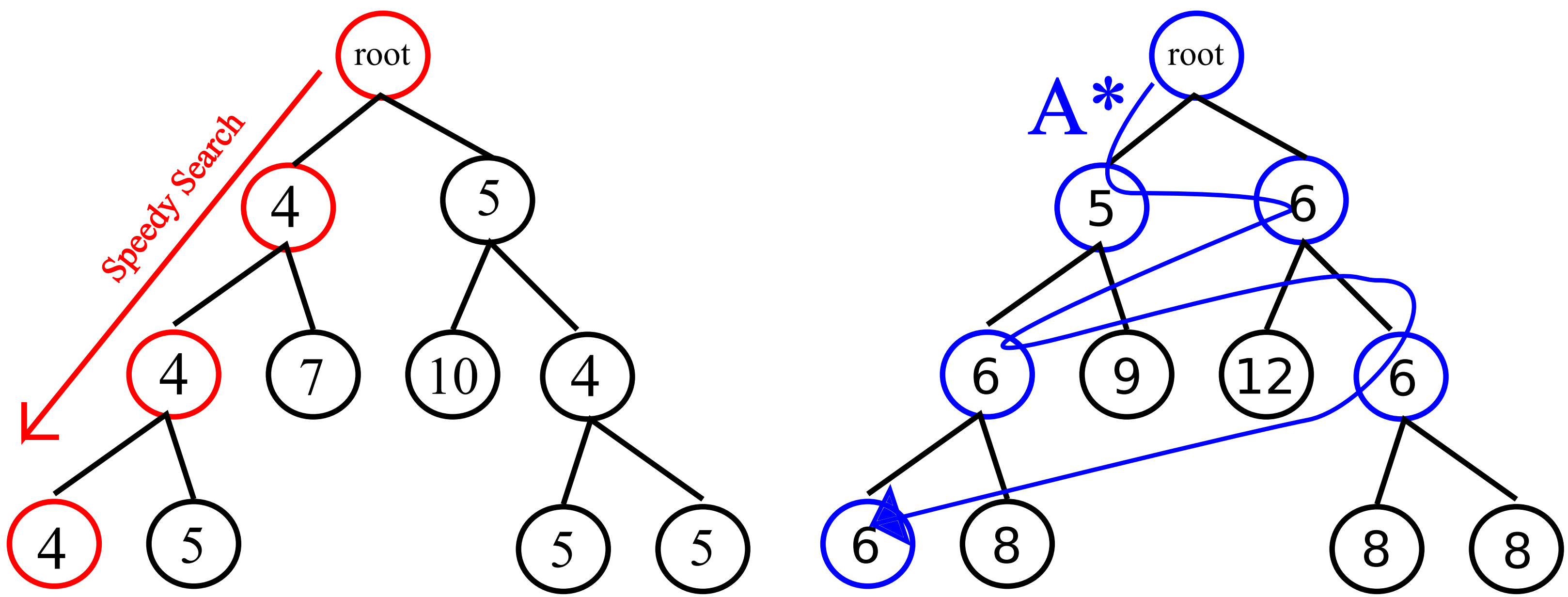
- interleaving search on d and h: alternation

    alternates between nodes sorted on h and $d_{cheapest}$

- beam search on d

    breadth-first beam search on $d_{cheapest}$

1. best-first search on distance-to-go estimate, $d(n)$.

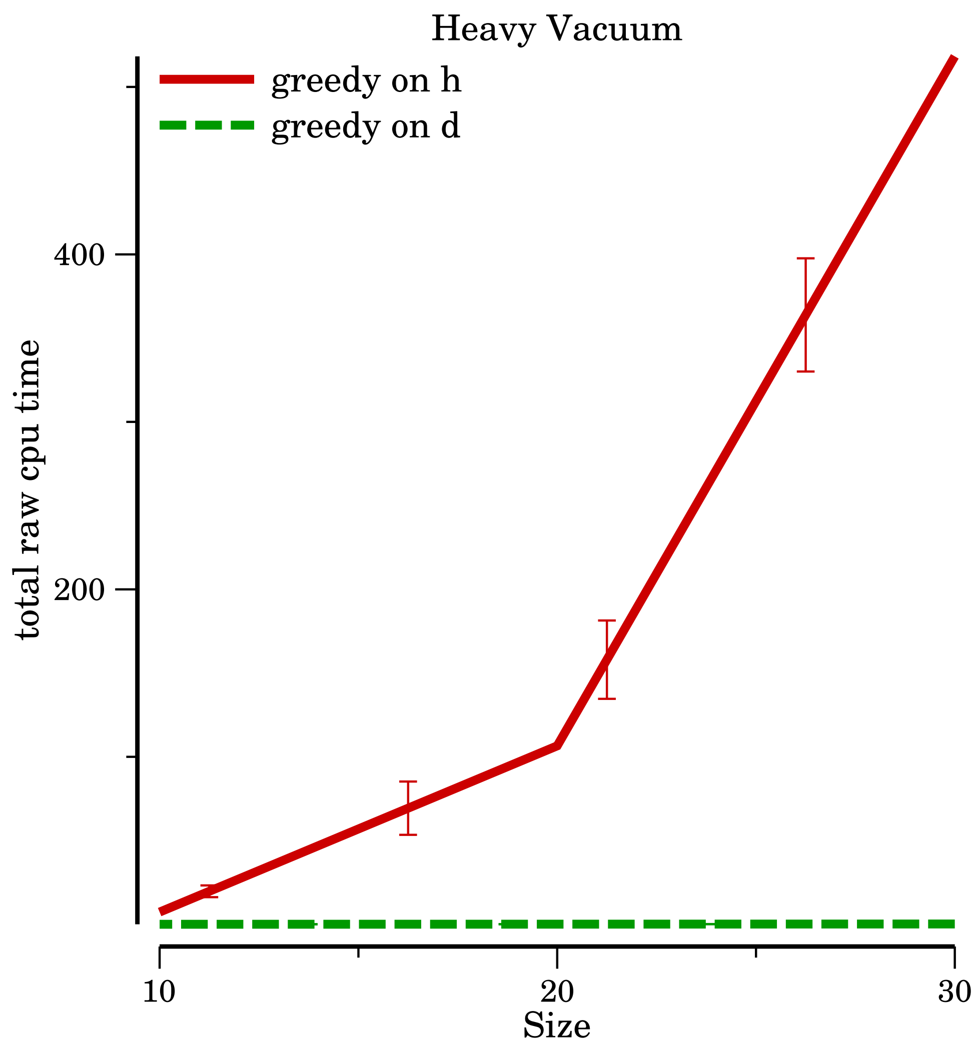


compare to greedy search, Doran and Michie 1966

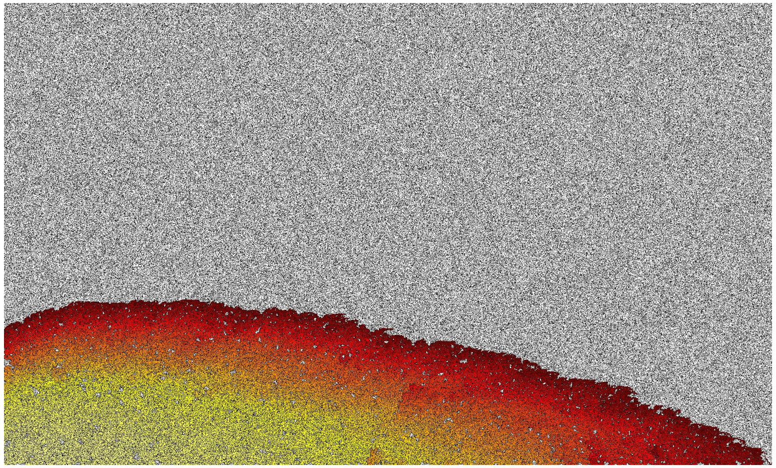
$h \propto d$, however $d = d$

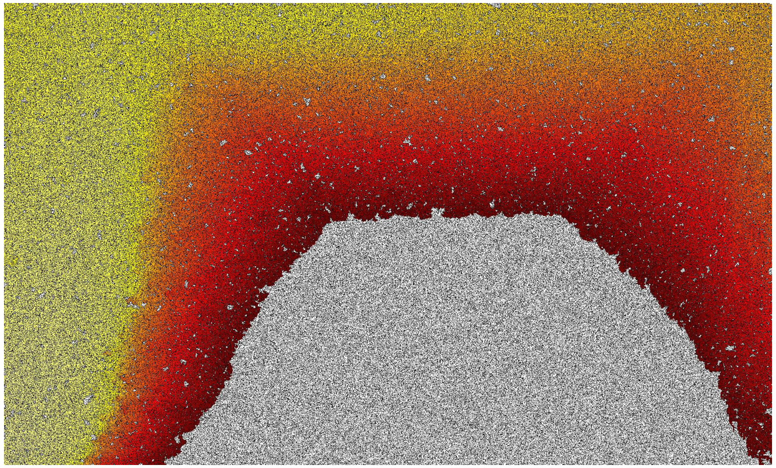
d drops consistently, resulting in low vacillation

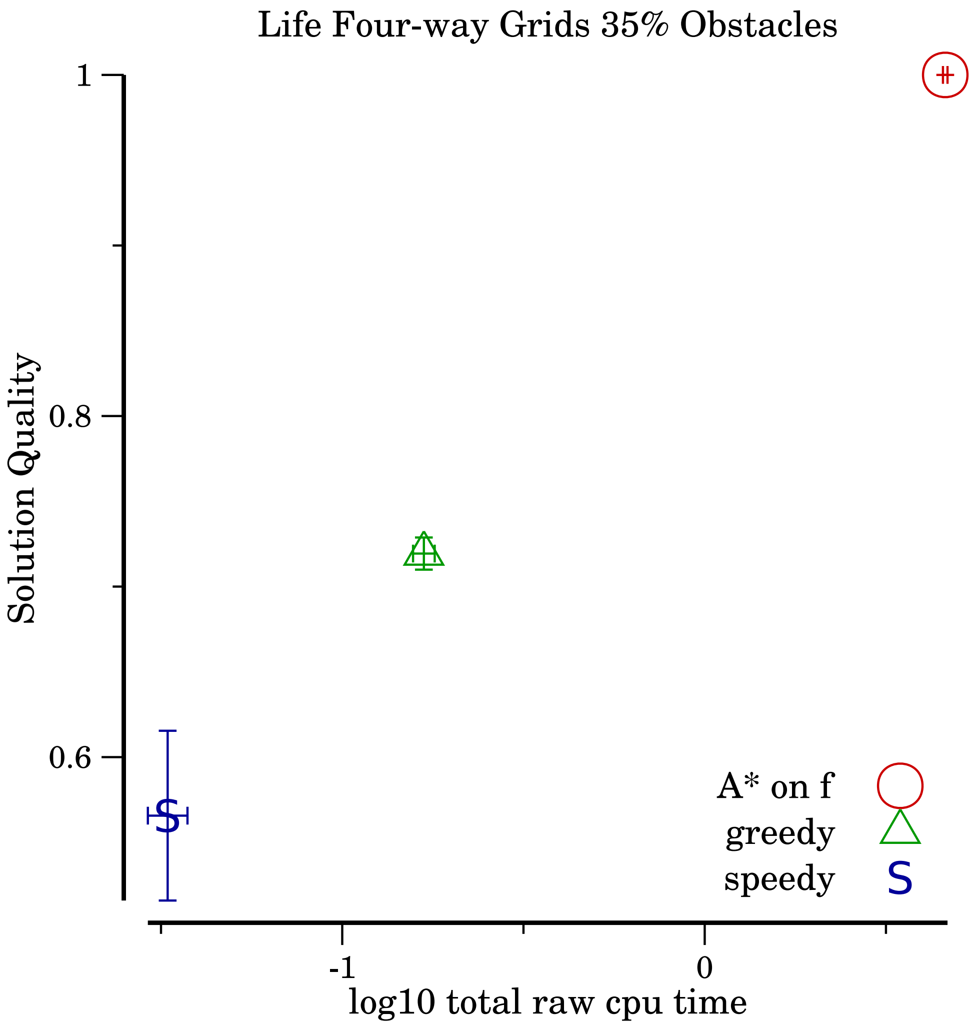
1. while *open* has nodes
2. remove n from *open* with minimum $d(n)$
3. if n is a goal then return n
4. otherwise expand n, inserting its children into *open*
5. return failure



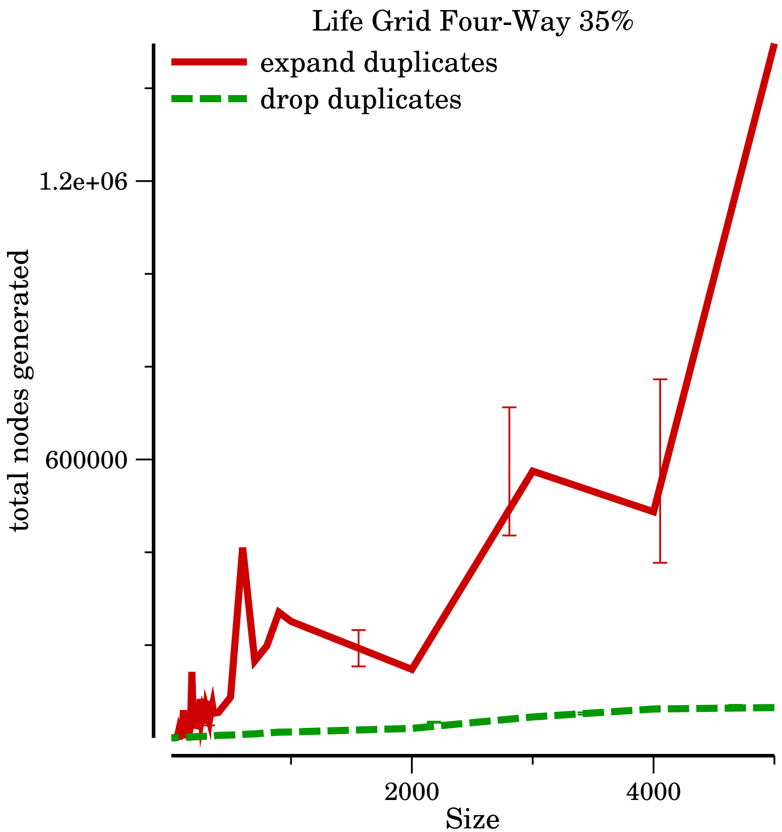
speedy tends to scale better than greedy

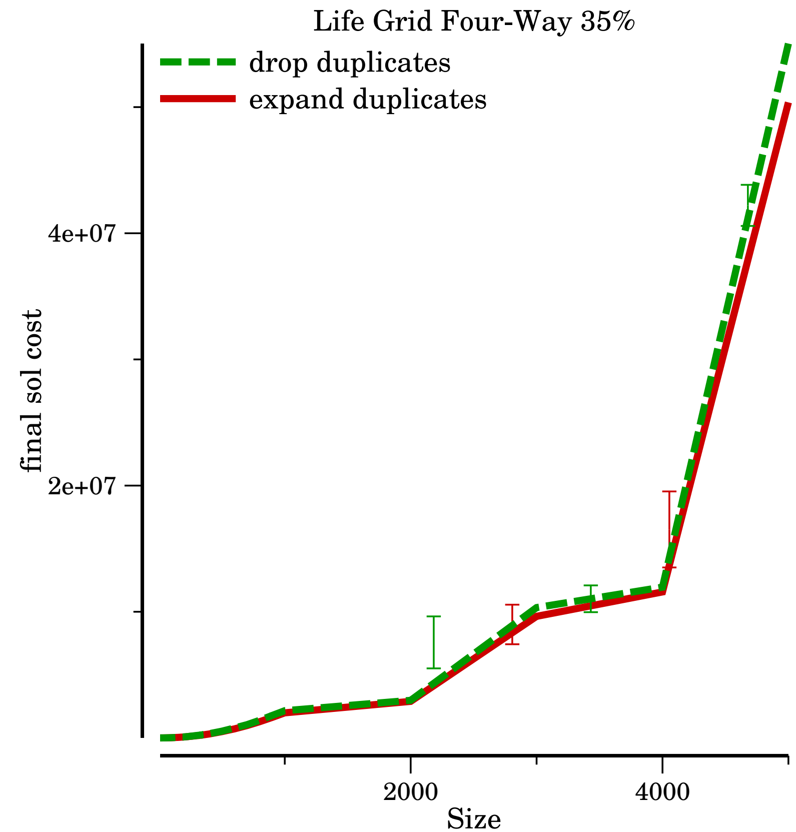






finds solutions faster at cost of quality





dropping duplicates improves speed, barely harms quality

Alternating Between h and d

Why not try alternating between h and d?

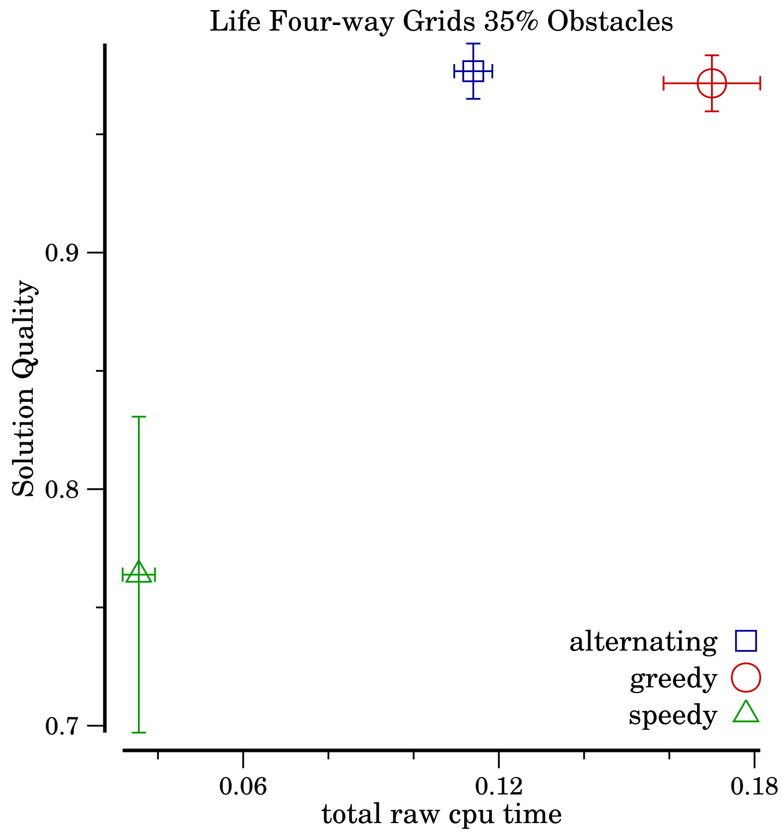
(Helmert and Röger ICAPS-10)

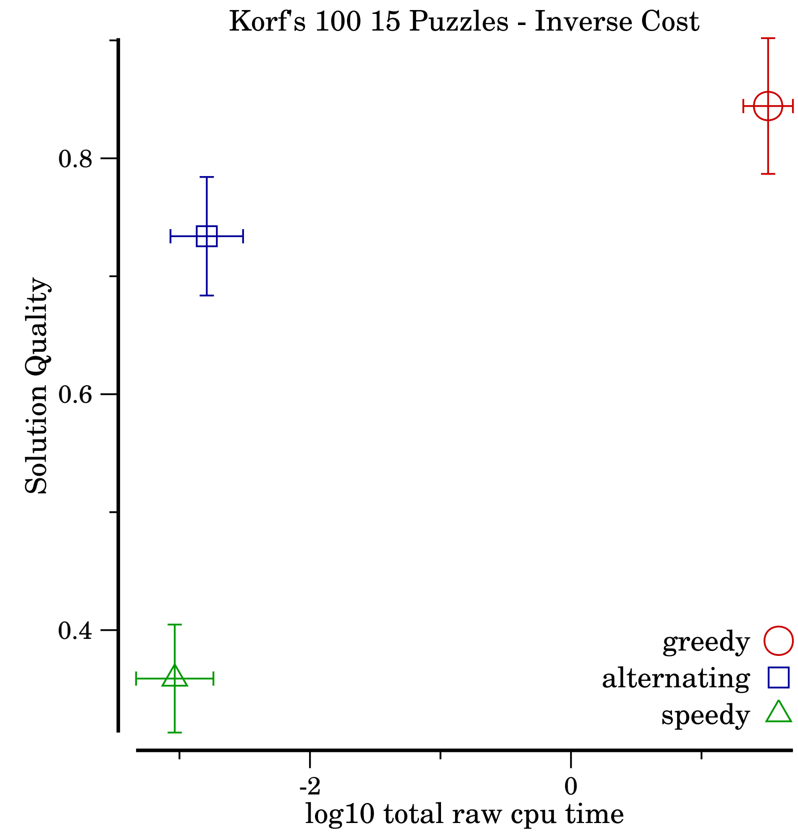
1. maintain one best-first queue on distance-to-go estimate, $d(n)$.
2. maintain another on cost-to-go estimate, $h(n)$
3. expand nodes from both, alternating between them

Alternating Between h and d

1. $from_d =$ true
2. while *open*$_d$ has nodes
3. if $from_d$
4. then $n \leftarrow best_d$
5. else $n \leftarrow best_f$
6. remove n from *open*$_d$ and *open*$_h$
7. if n is a goal then return n
8. otherwise expand n, inserting its children into *open*$_d$ and *open*$_h$
9. toggle $from_d$
10. return failure

Alternating Between h and d

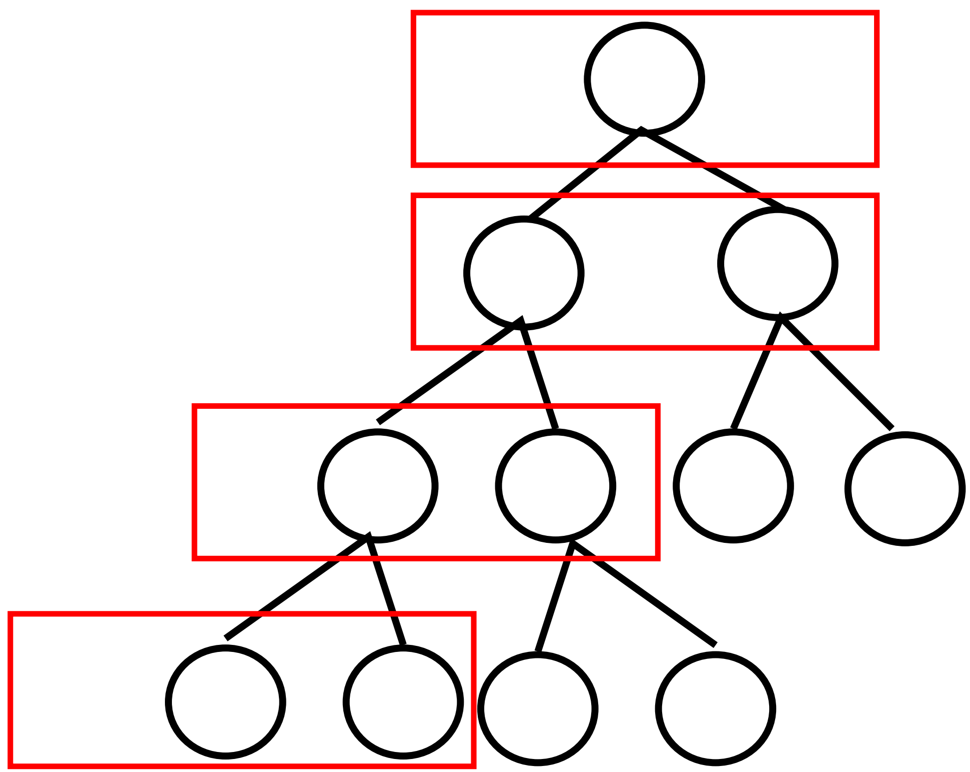




alternating provides a middle ground between greedy and speedy

Breadth-First Beam Search

1. run breadth-first search with a fixed sized open list
2. filter out nodes with high $d(n)$



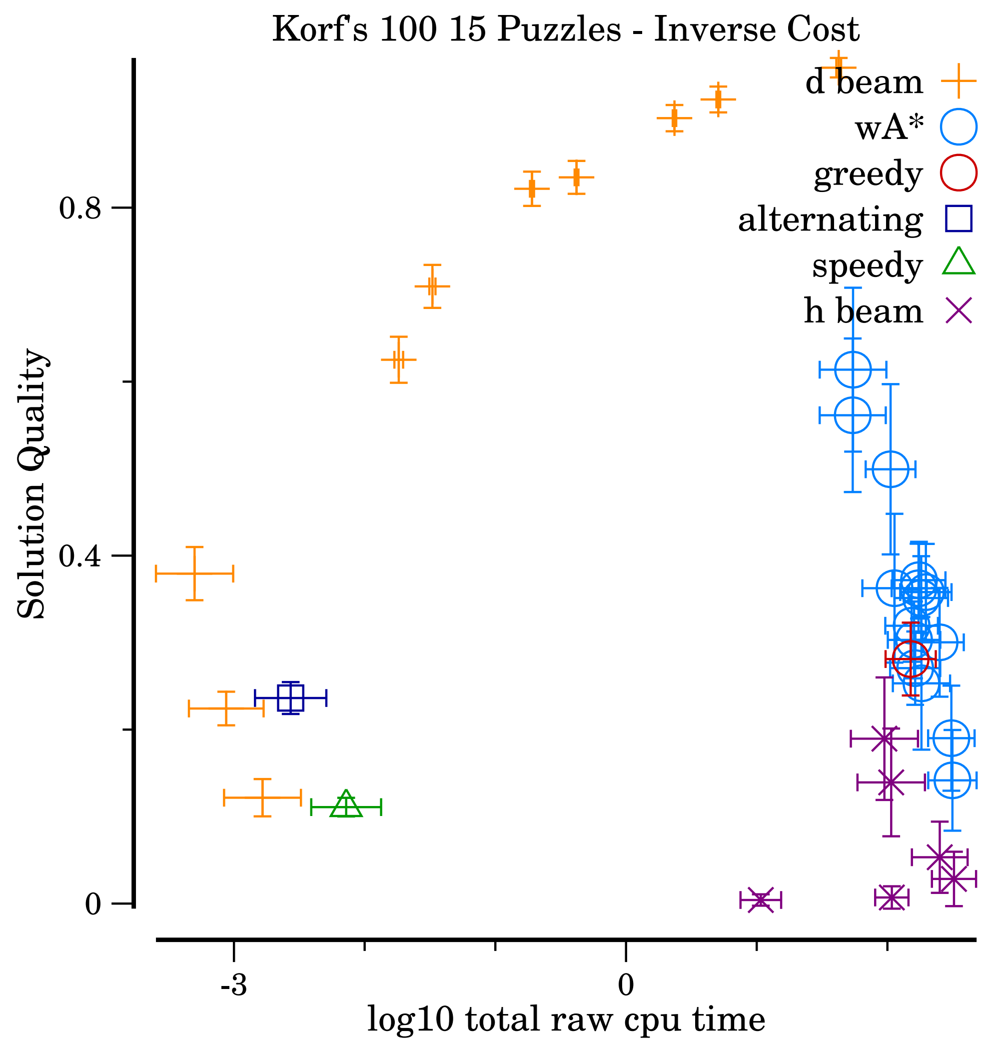
Beam Search On d

1. while *open* has nodes
2. for each $n \in$ *open*
3. if n is a goal, return n
4. otherwise expand n, adding to *children*
5. *open* becomes best *width* children in *children*

5a. best according to $f(n) = g(n) + h(n)$

6. return failure

```
1.   while open has nodes
2.     for each n ∈ open
3.       if n is a goal, return n
4.       otherwise expand n, adding to children
5.     open becomes best width children in children
5a.      best according to d(n)
6.   return failure
```



ignoring cost information can improve performance

Summary: d in Suboptimal Search

- when minimizing solving time, search on d
- distance estimates easy to use in suboptimal search

  no bounds to worry about

  generally, just swap d for h, $Depth$ for g
- some cost information can be retained by using $d_{cheapest}$

  but search on $d_{nearest}$ likely faster
- solution quality tends to suffer

  but speed and coverage improve

Bounded Suboptimal Search

- proving bounds (for those who just came in)
- revised dynamically weighted A*

  scales w according to $d_{cheapest}$
- skeptical search

  use d to learn better h

  full talk wednesday 10:30
- A^*_ϵ

  use d to find solution within bound fast
- explicit estimation search

  uses inadmissible heuristics to correct flaw in A^*_ϵ

  full talk at IJCAI-11 on Friday July 22

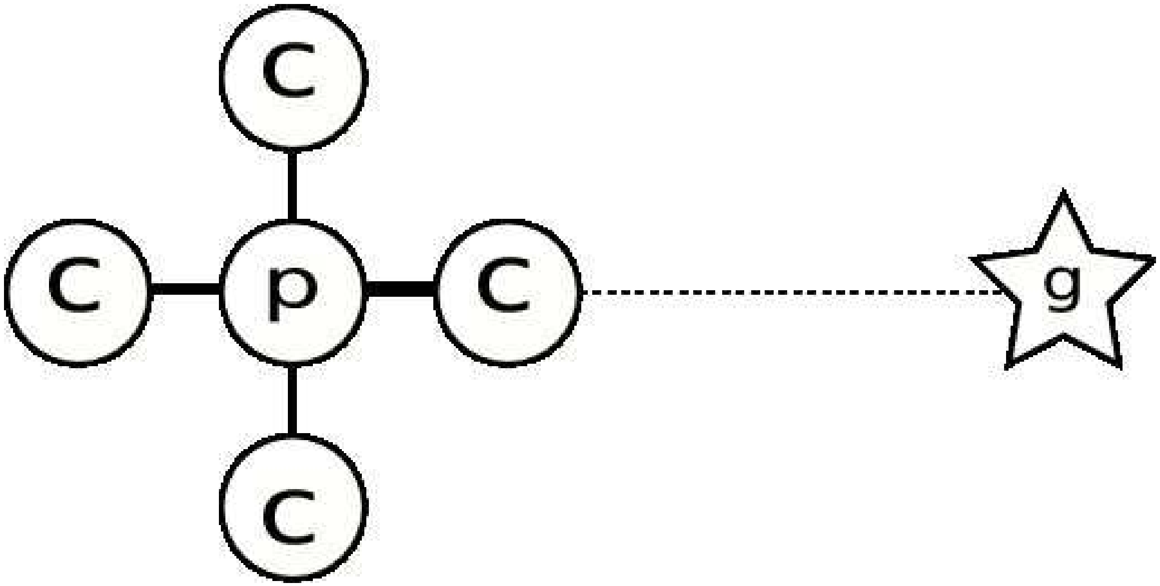
uninteresting alone, but useful in anytime frameworks

1. run weighted A*
2. as search progresses, decrease weight

never weighted more than weighted A*, so same bound holds

$$f_{dwA*}(n) = g(n) + w \cdot (1 - \frac{Depth(n)}{d(root)}) \cdot h(n)$$
$$f_{wA*}(n) = g(n) + w \cdot h(n)$$

uninteresting alone, but useful in anytime frameworks

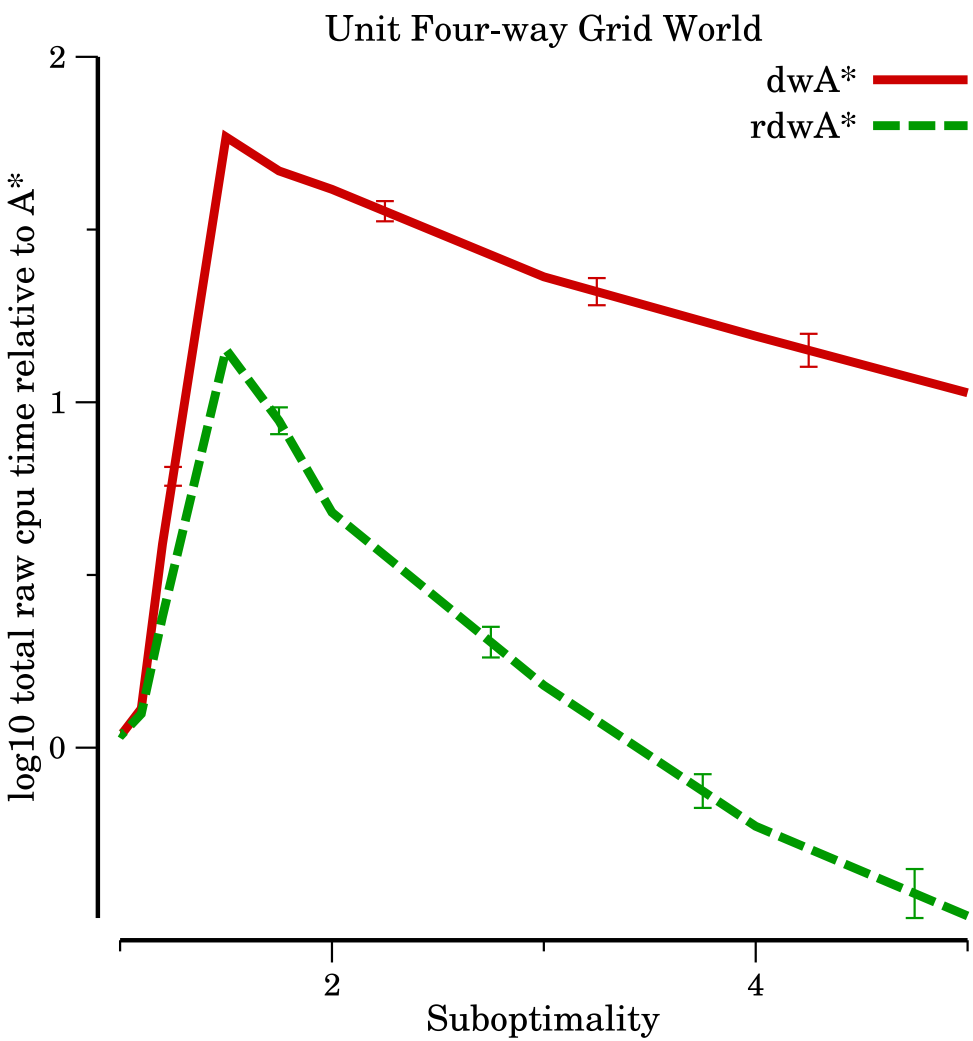


moving away from root $\neq$ moving towards goal!

1. run weighted A*
2. as search progresses, decrease weight

$$f_{rdwA*}(n) = g(n) + h(n) + w \cdot \frac{d(n)}{d(root)} \cdot h(n)$$

Revised Dynamically Weighted A* Thayer and Ruml ICAPS-09

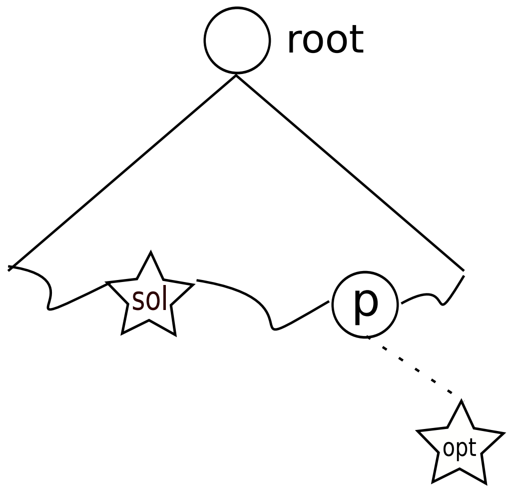


reward progress, not effort!

$$f_{dwA*} = g(n) + h(n) \cdot w \cdot max(1, min(0, (1 - \frac{Depth(n)}{d(root)})))$$

$$f_{rdwA*} = g(n) + h(n) \cdot max(1, min(w, w \cdot \frac{d(n)}{d(root)}))$$

Two Ways To Provide Bounds



$$f(n) = g(n) + h(n)$$
$$f'(n) = g(n) + w \cdot h(n)$$

- p is the deepest node on an optimal path to opt.

$$g(sol)$$
$$f'(sol) \leq f'(p)$$
$$g(p) + w \cdot h(p) \leq w \cdot (g(p) + h(p))$$
$$w \cdot f(p) \leq w \cdot f(opt)$$
$$w \cdot g(opt)$$

1. works for any $f'(p) \leq w \cdot f(p)$
2. $g(p) + w \cdot h(p) \ll w(g(p) + h(p))$!

- use d to adjust cost-to-go heuristic
- use adjusted heuristic in optimistic search:

1. aggressive: run weighted A^* with an inadmissible heuristic

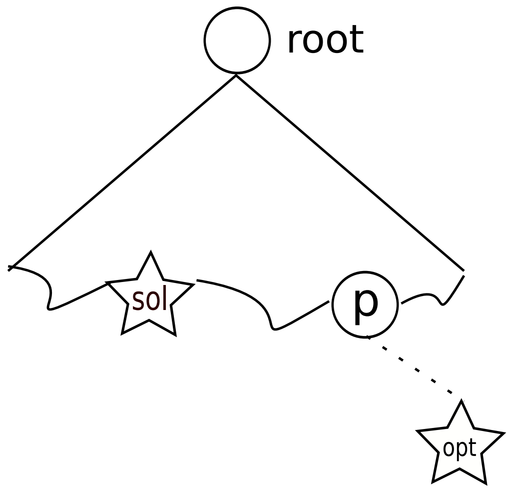
   $$f'(n) = g(n) + w \cdot \widehat{h}(n)$$

   if $\widehat{h}$ close to h^*, solution should be within bound

2. cleanup: check $w \cdot best_f > f(sol)$

   expand $best_f$ until within bound

   guarantees solution quality



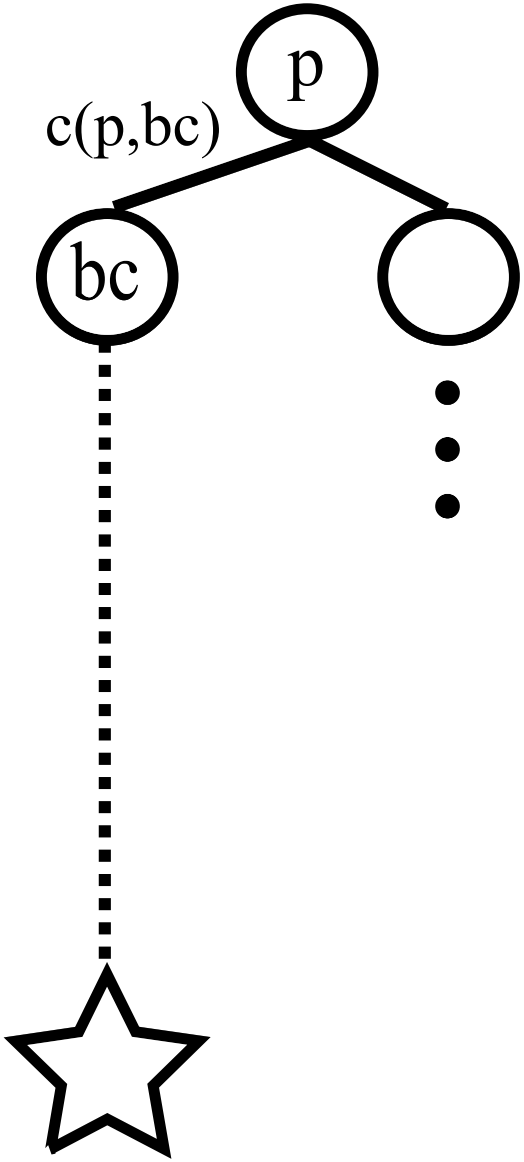
- p is the deepest node on an optimal path to opt.
- $best_f$ is the node with the smallest f value.

$$f(p) \leq f(opt)$$
$$f(best_f) \leq f(p)$$

$best_f$ provides a lower bound on solution cost

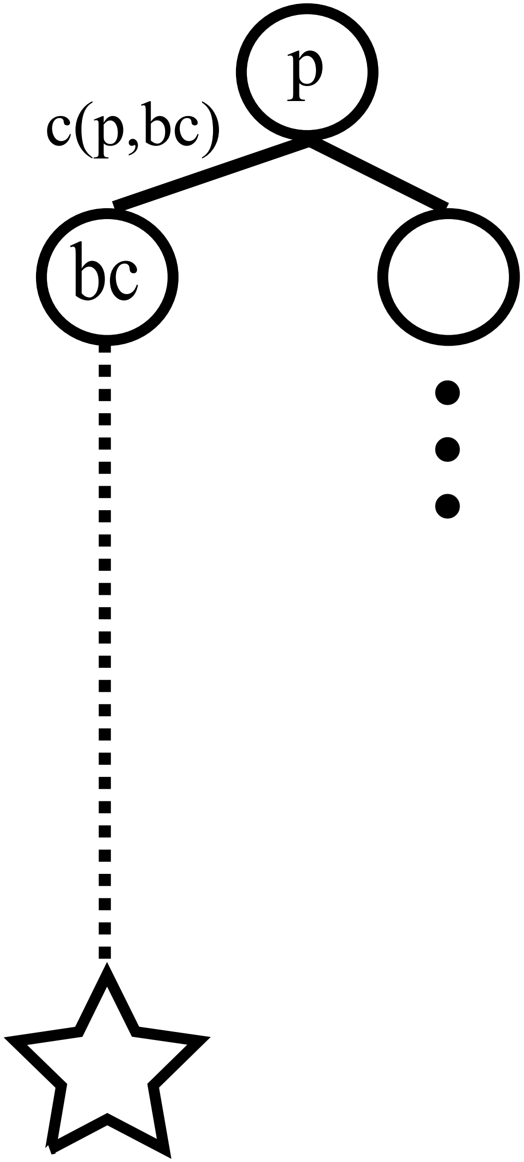
determine $best_f$ by priority queue sorted on f

Using d And Observed Error To Correct h



$f(p)$ should equal $f(bc)$

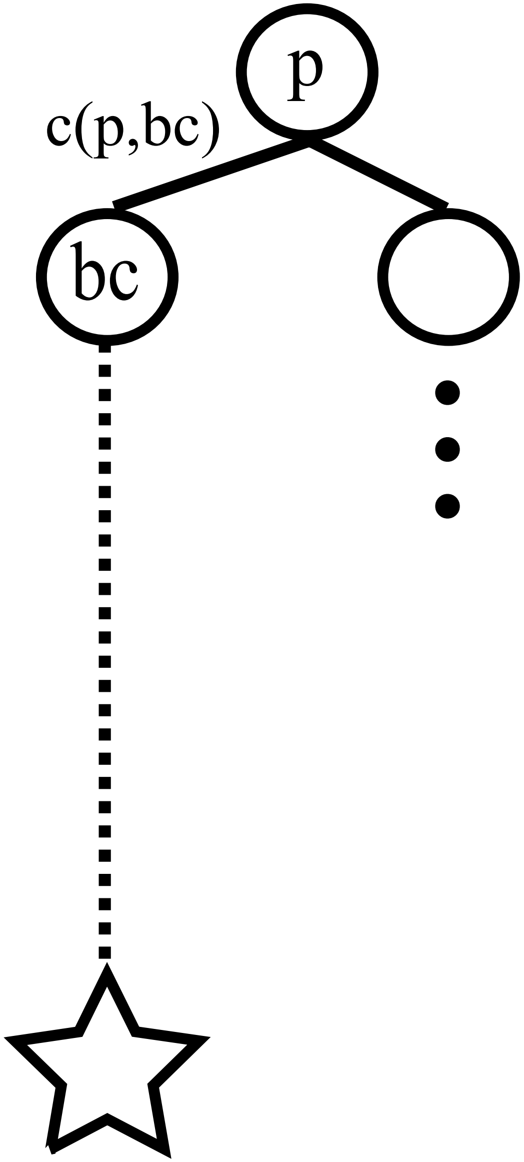
Using d And Observed Error To Correct h



$f(p)$ should equal $f(bc)$

$$
\begin{aligned}
f^*(p) &= f^*(bc) \\
g(p) + h^*(p) &= g(bc) + h^*(bc) \\
h^*(p) &= h^*(bc) + c(p, bc)
\end{aligned}
$$

Using d And Observed Error To Correct h

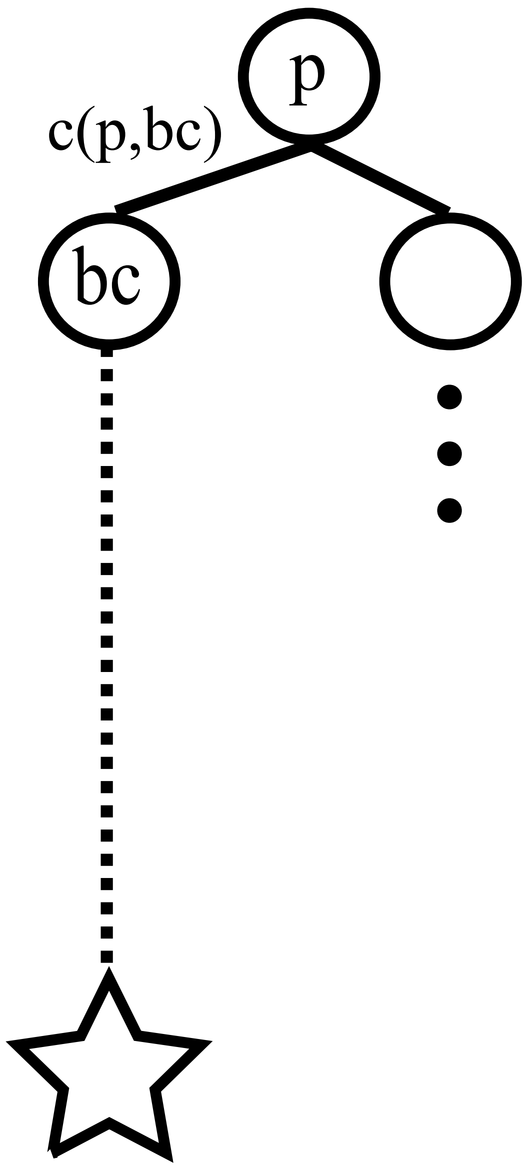


$f(p)$ should equal $f(bc)$

$$\begin{aligned} f^*(p) &= f^*(bc) \\ g(p) + h^*(p) &= g(bc) + h^*(bc) \\ h^*(p) &= h^*(bc) + c(p, bc) \end{aligned}$$

$$\begin{aligned} h(p) &= h(bc) + c(p, bc) - \epsilon_h \\ \epsilon_h &= h(bc) + c(p, bc) - h(p) \end{aligned}$$

Using d And Observed Error To Correct h



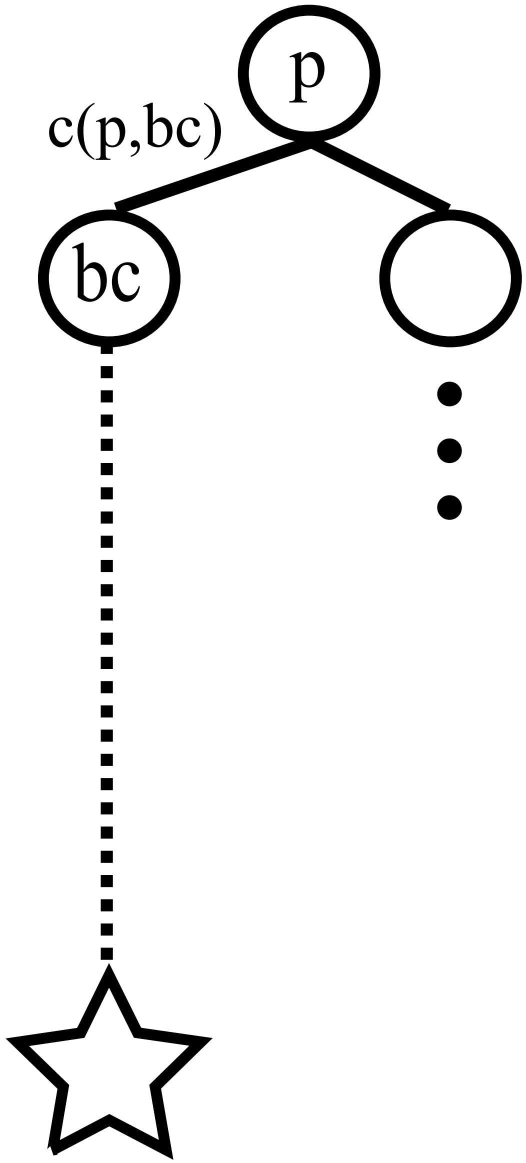
$f(p)$ should equal $f(bc)$

$$
\begin{aligned}
f^*(p) &= f^*(bc)\\
g(p) + h^*(p) &= g(bc) + h^*(bc)\\
h^*(p) &= h^*(bc) + c(p, bc)
\end{aligned}
$$

$$
\begin{aligned}
h(p) &= h(bc) + c(p, bc) - \epsilon_h\\
\epsilon_h &= h(bc) + c(p, bc) - h(p)
\end{aligned}
$$

$$
\widehat{h}(n) = h(n) + \bar{\epsilon_h} \cdot d(n)
$$

Using d And Observed Error To Correct h



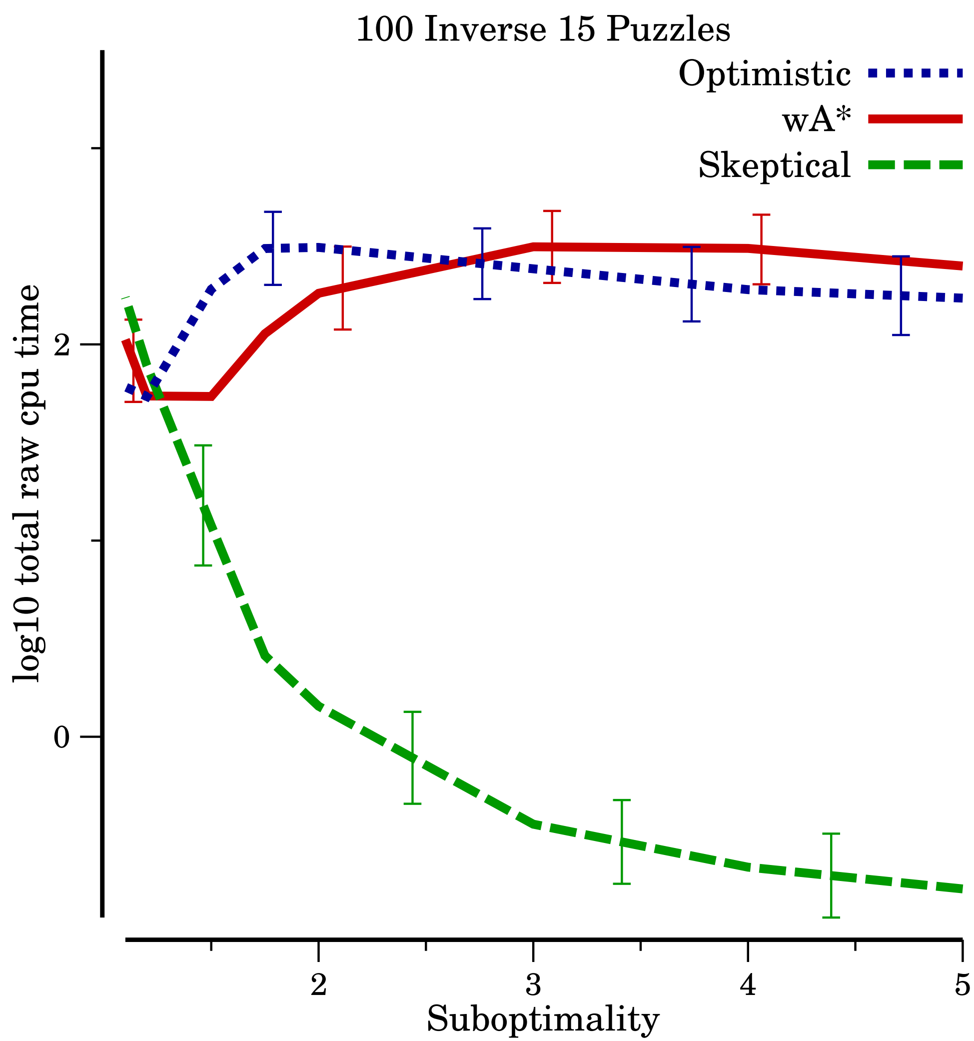
$f(p)$ should equal $f(bc)$

$$\begin{aligned} f^*(p) &= f^*(bc) \\ g(p) + h^*(p) &= g(bc) + h^*(bc) \\ h^*(p) &= h^*(bc) + c(p, bc) \end{aligned}$$

$$\begin{aligned} h(p) &= h(bc) + c(p, bc) - \epsilon_h \\ \epsilon_h &= h(bc) + c(p, bc) - h(p) \end{aligned}$$

$$\begin{aligned} \widehat{h}(n) &= h(n) + \bar{\epsilon_h} \cdot d(n) \\ \widehat{h}(n) &= h(n) + \bar{\epsilon_h} \cdot \widehat{d}(n) \end{aligned}$$

Skeptical Search Performance



every expansion provides information; use it!

intuition: of all solutions within the bound, the nearest should be the fastest to find.

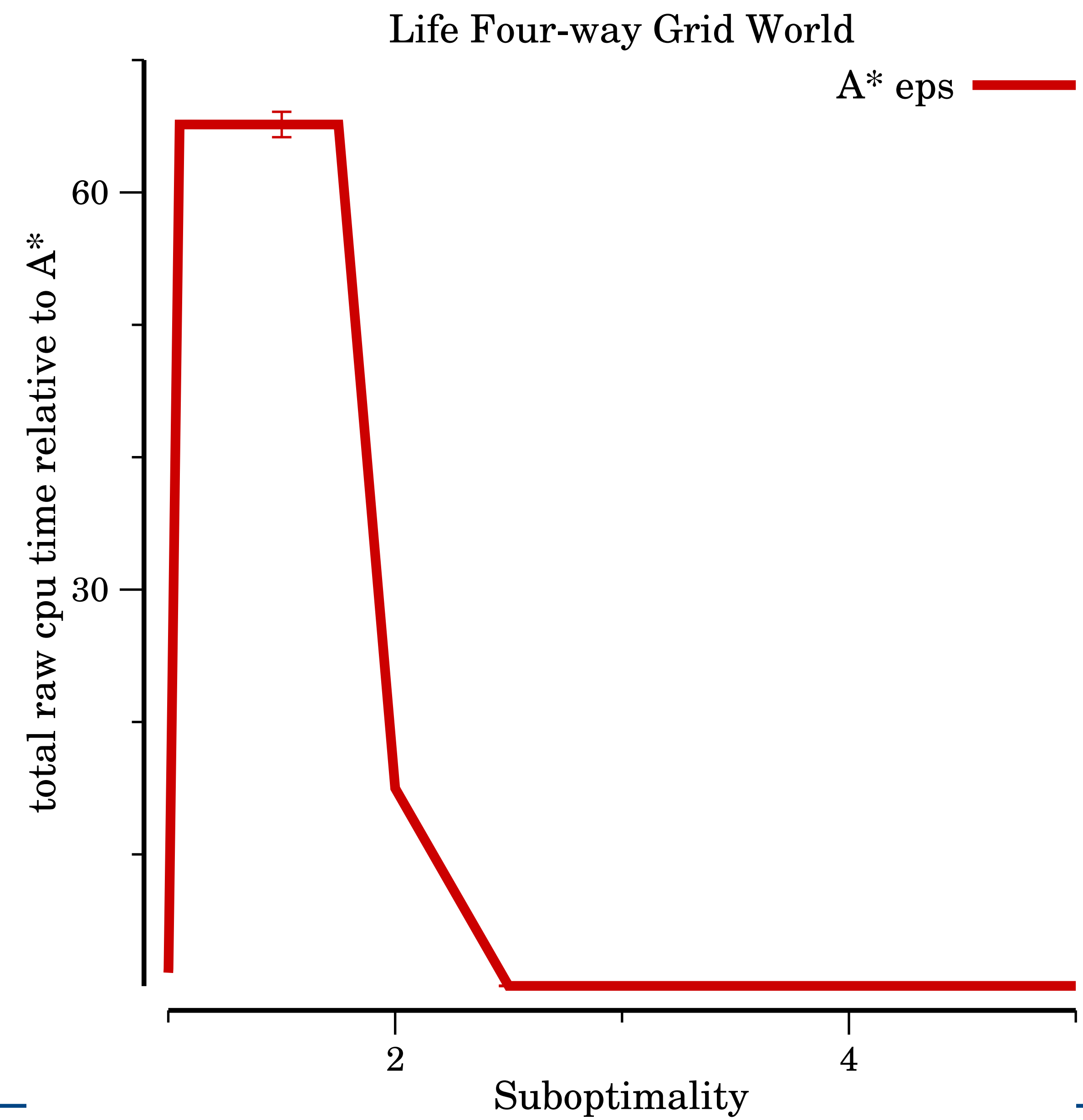
best-first search on two lists:

open: all generated but unexpanded nodes, sorted on $f(n)$.

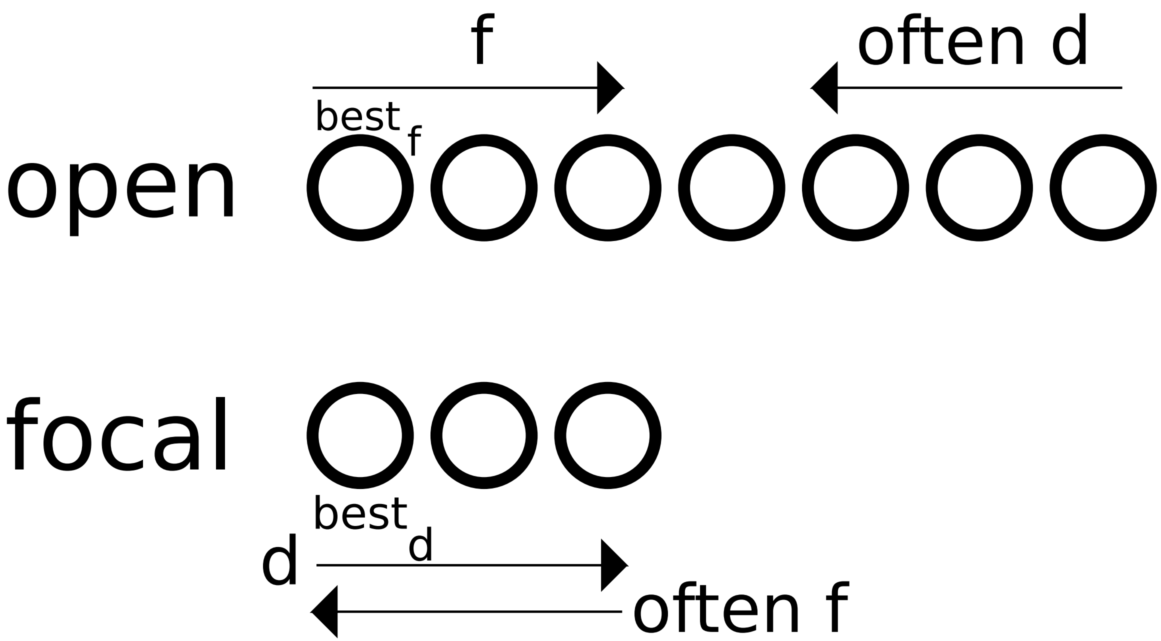
focal: all nodes where $f(n) \leq w \cdot f(best_f)$ sorted on $d(n)$

Expand the best node from *focal*

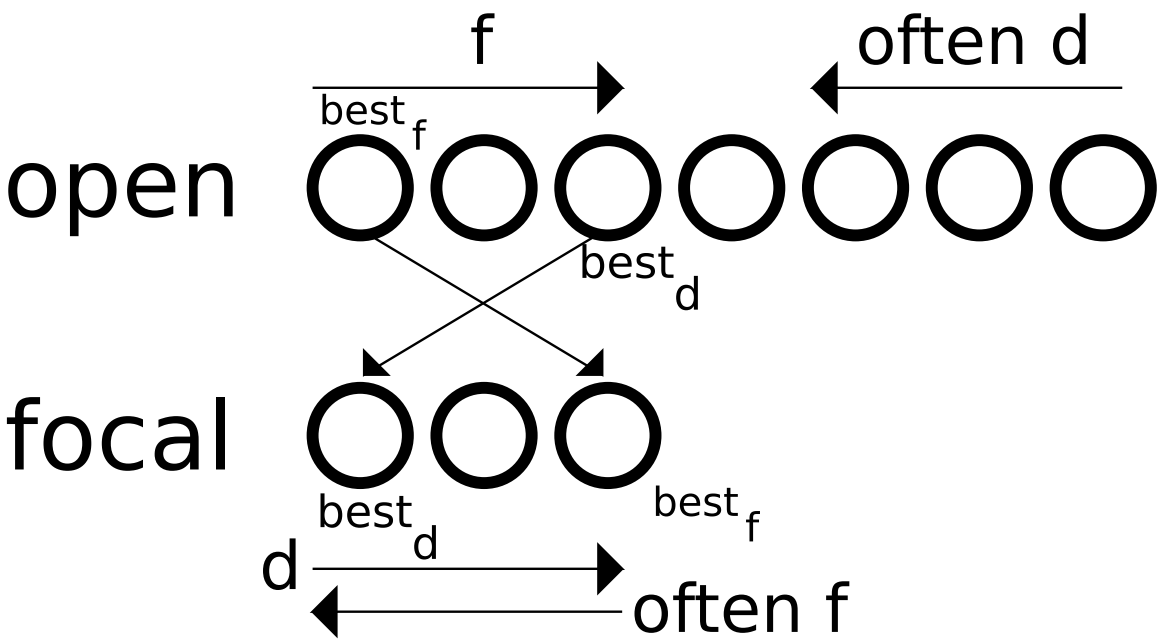
A^*_ϵ Doesn't Work Very Well



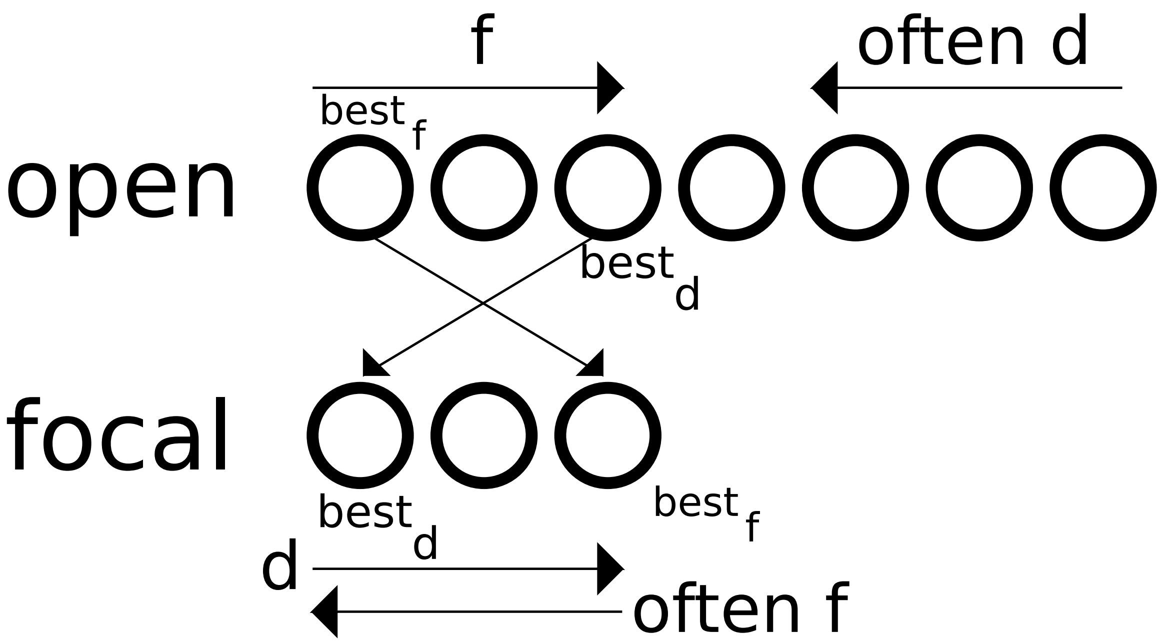
open all generated but unexpanded nodes, sorted on $f(n)$.
focal all nodes where $f(n) \leq w \cdot f(best_f)$ sorted on $d(n)$



open all generated but unexpanded nodes, sorted on $f(n)$.
focal all nodes where $f(n) \leq w \cdot f(best_f)$ sorted on $d(n)$



open all generated but unexpanded nodes, sorted on $f(n)$.
focal all nodes where $f(n) \leq w \cdot f(best_f)$ sorted on $d(n)$



f rises as search progresses (h is admissible)
$best_d$'s children won't remain on focal

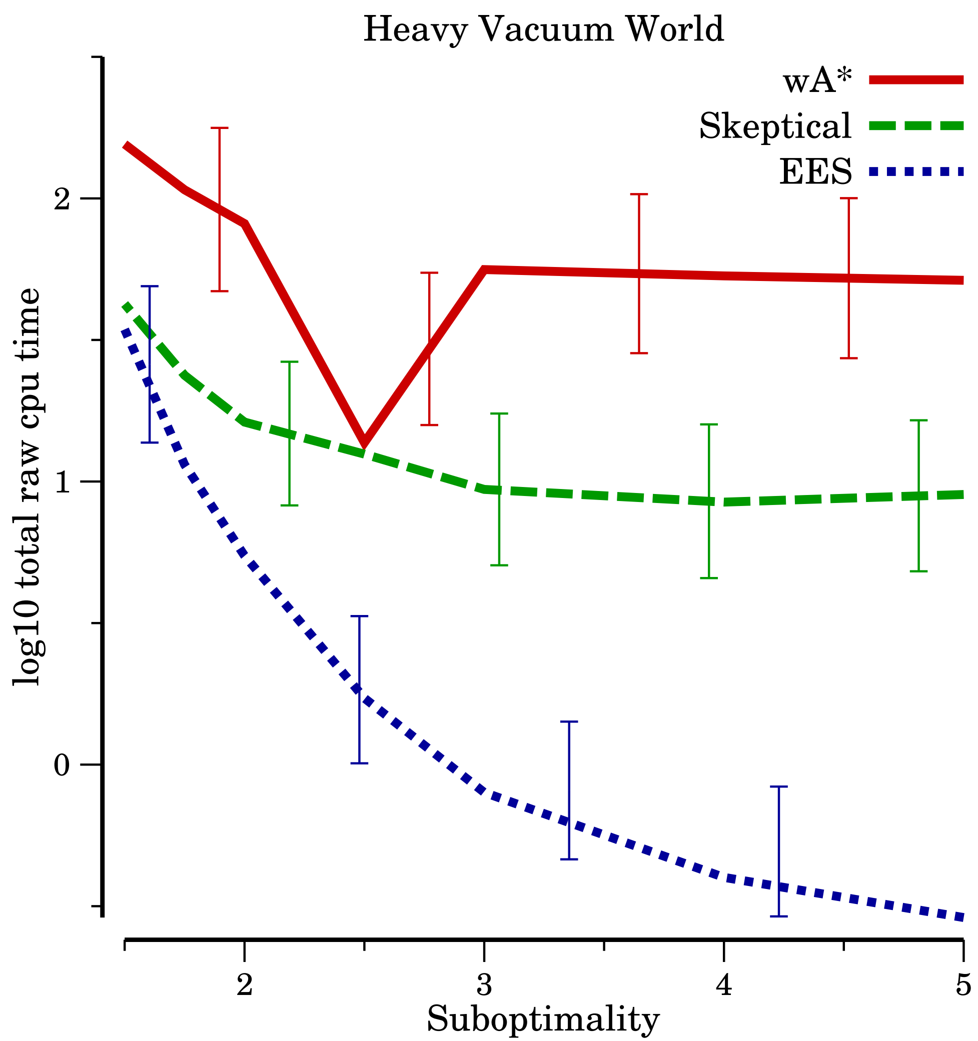
intuition: pursuing the shortest solution within the bound should be fast

intuition′: using unbiased estimates of cost should prevent $\widehat{f}$ from rising

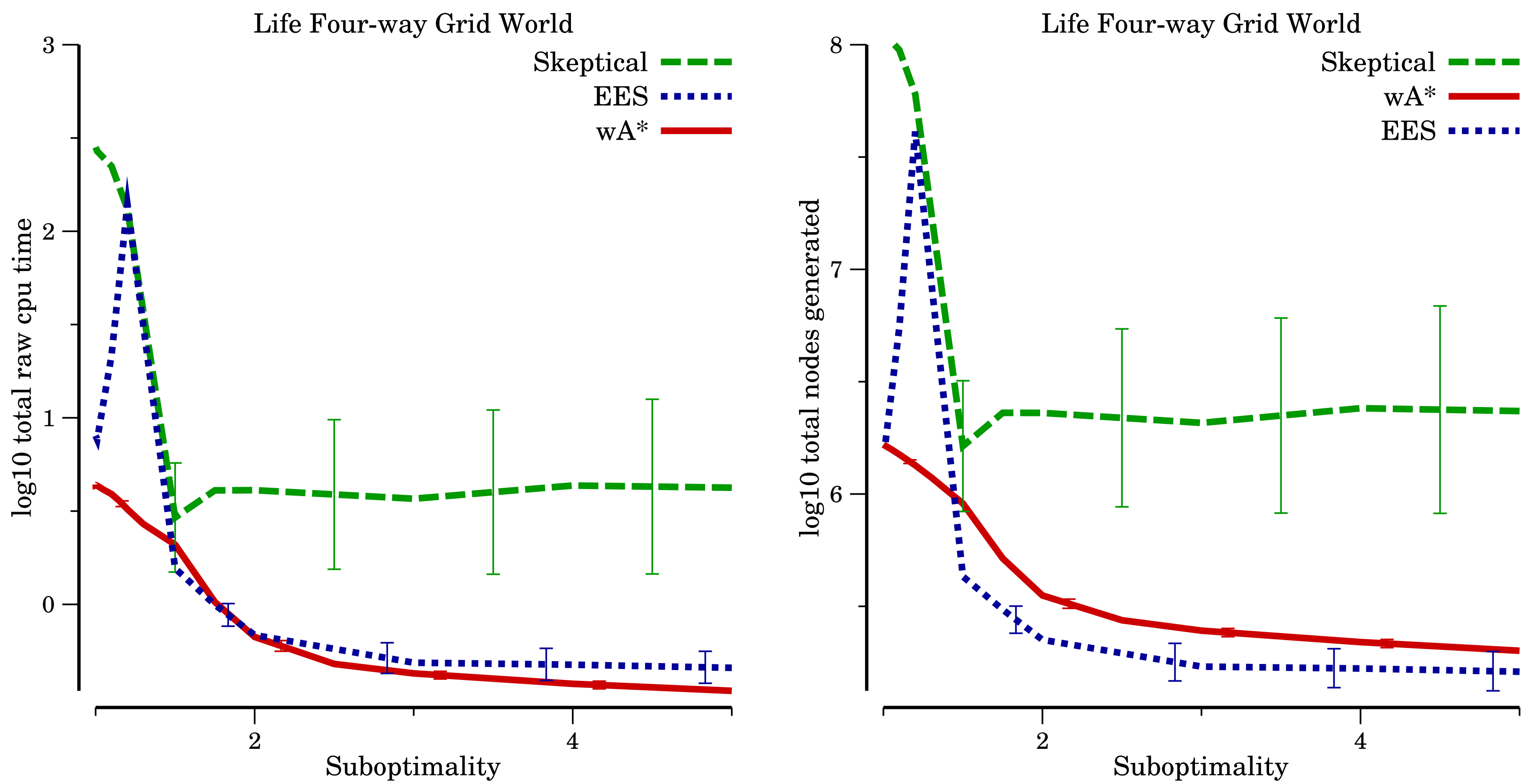
open all generated but unexpanded nodes, sorted on $\widehat{f}(n)$.
focal all nodes where $\widehat{f}(n) \leq w \cdot f(best_{\widehat{f}})$, sorted on $\widehat{d}(n)$
cleanup all generated but unexpanded nodes, sorted on $f(n)$

$selectNode$

1. **if** $\widehat{f}(best_{\widehat{d}}) \leq w \cdot f(best_f)$ **then** $best_{\widehat{d}}$
2. **else if** $\widehat{f}(best_{\widehat{f}}) \leq w \cdot f(best_f)$ **then** $best_{\widehat{f}}$
3. **else** $best_f$



searching on d better than augmenting cost estimates



EES not best if expansion essentially free
thank you planning community!

Summary: d in Bounded Suboptimal Search

- d needn't be admissible
  - even though an admissible h is required
  - inadmissible guidance with quality bounds
- d can dramatically improve performance
  - provides estimate of quantity being optimized
- d can be computed alongside h
  - so it's essentially free
- can be used to add introspection to search
  - Wednesday 10:30 "Frontiers of Planning"

Anytime Search

Distance Estimates In Anytime Search

- d-fenestration

  explores a subset of search space based on d values

- size-cost search

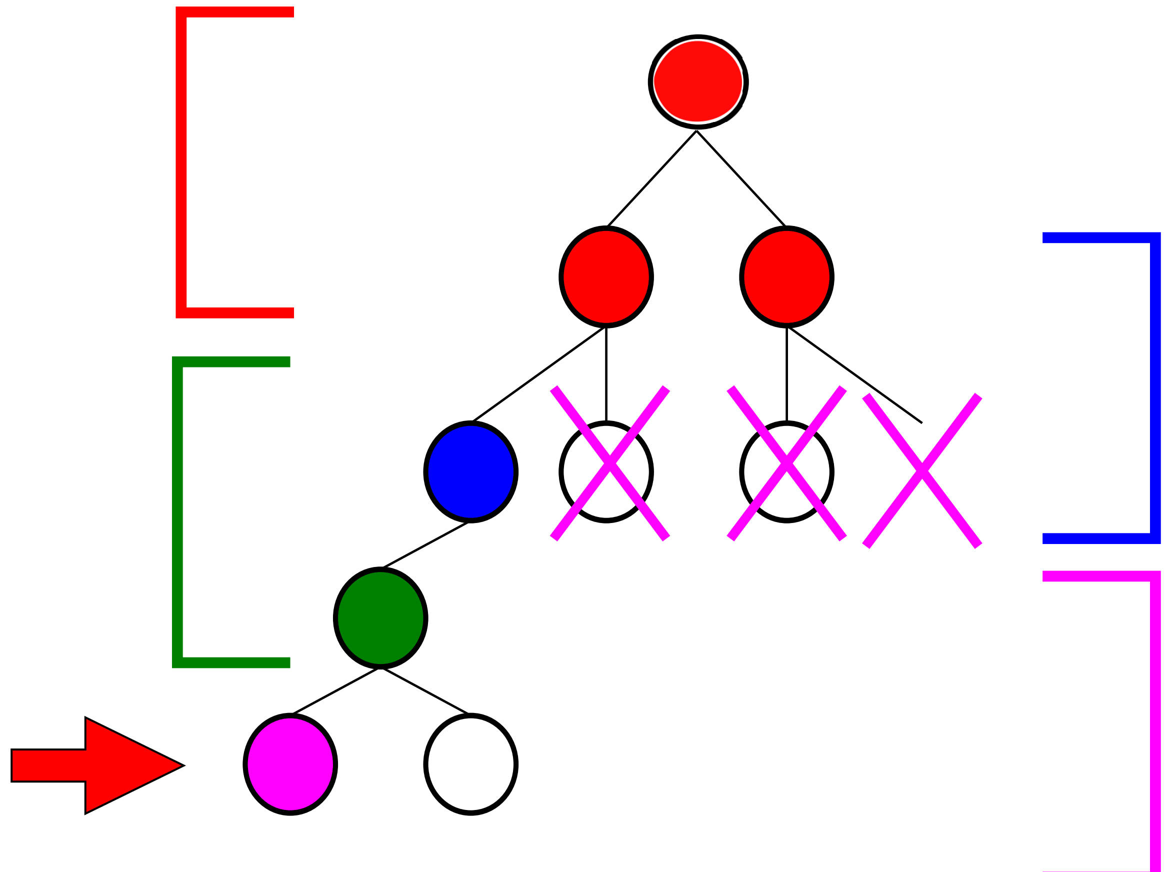
  searches on length, but prunes on cost

- continued, repairing, restarting search

  use existing frameworks with d-aware search

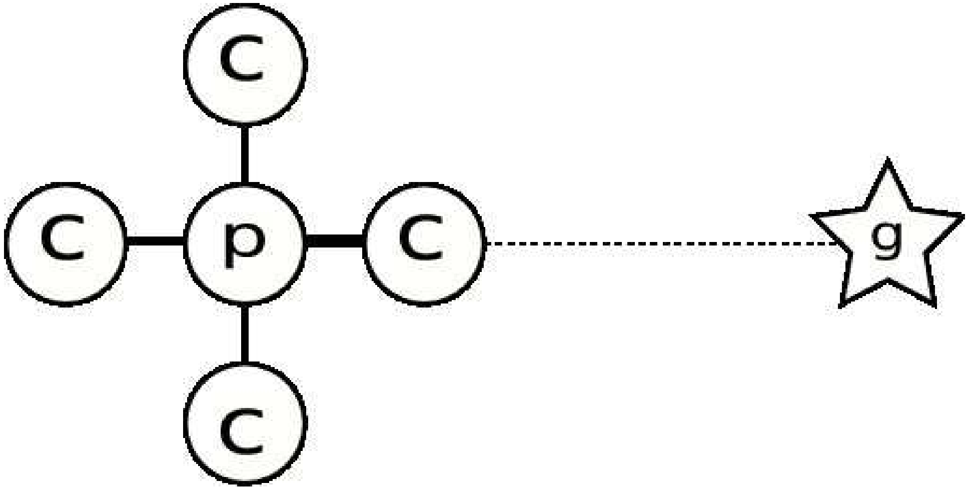
force nodes being compared to be similarly informed

assume similar depth implies similarly accurate heuristics



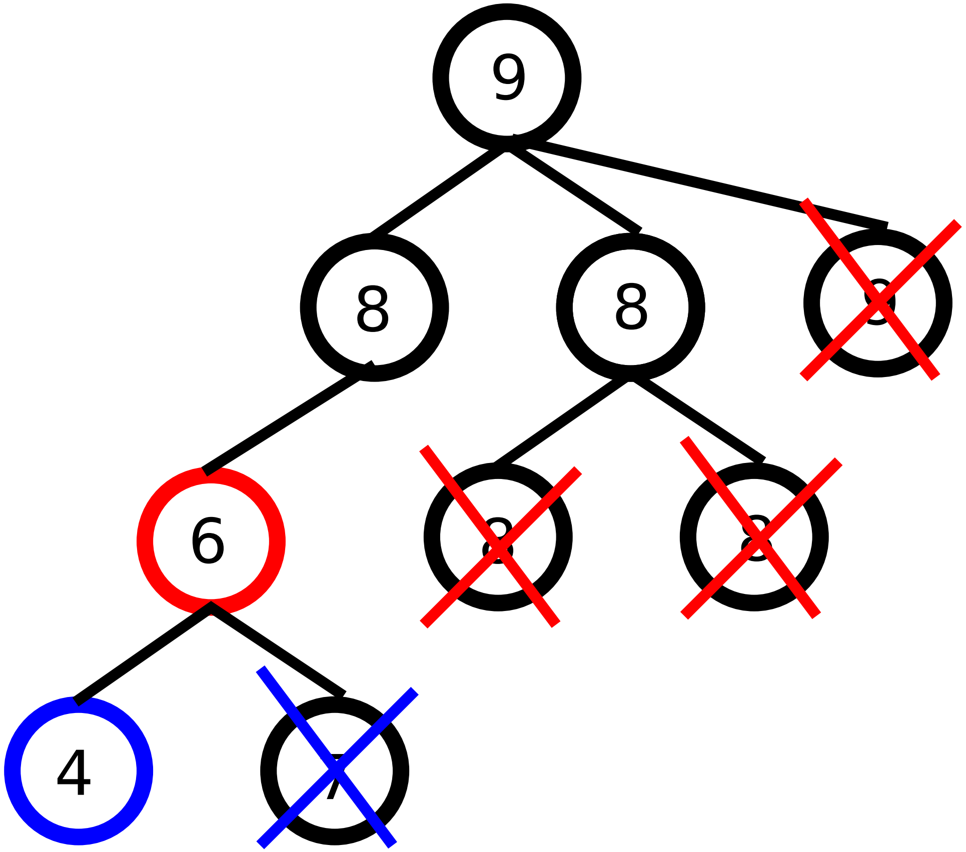
force nodes being compared to be similarly informed

- assume similar depth implies similarly accurate heuristics
- assumes depth $\propto d$, conflates effort with progress

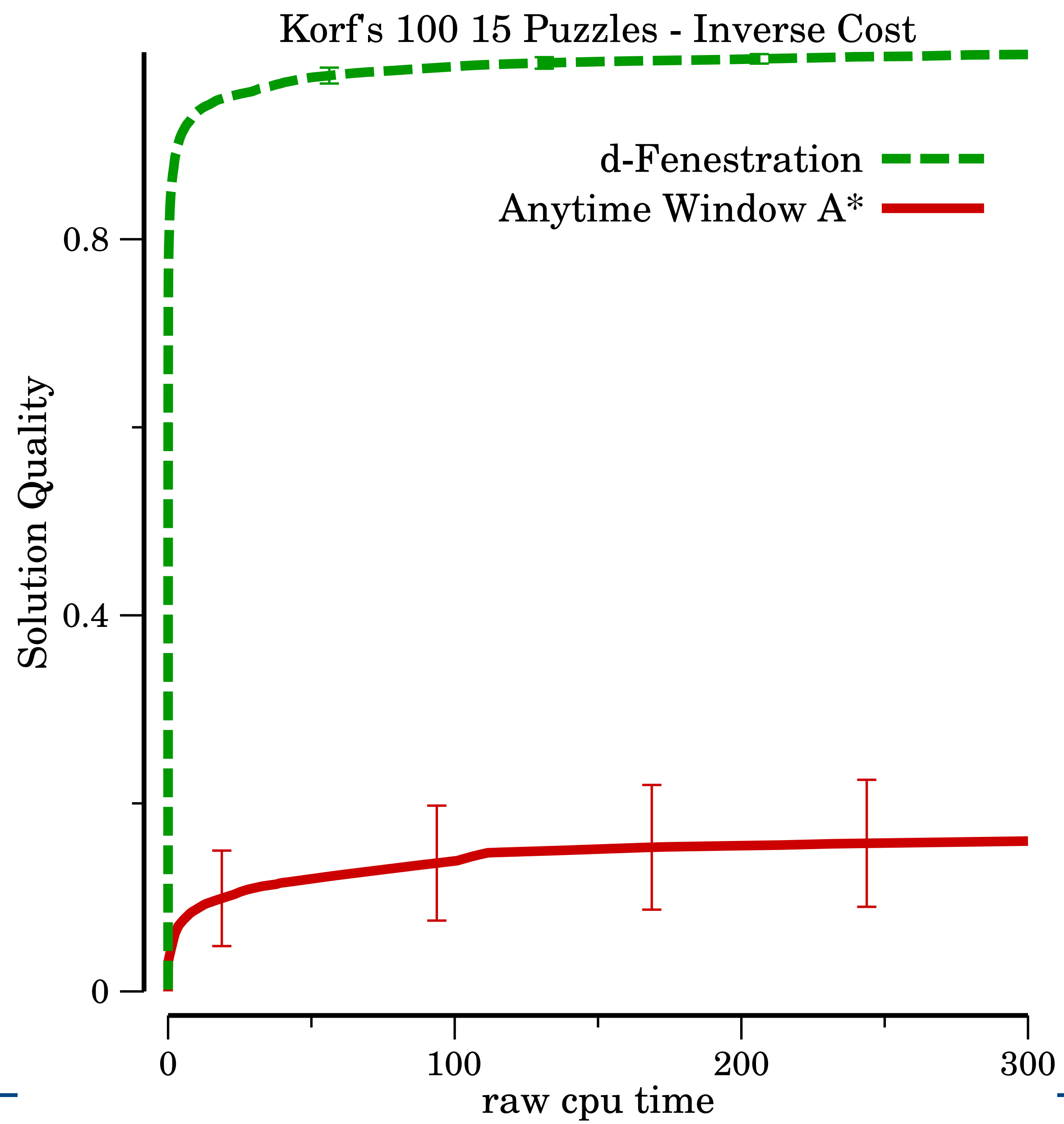


d and depth are not always related

force nodes being compared to be similarly informed
assume similar d implies similarly accurate heuristics



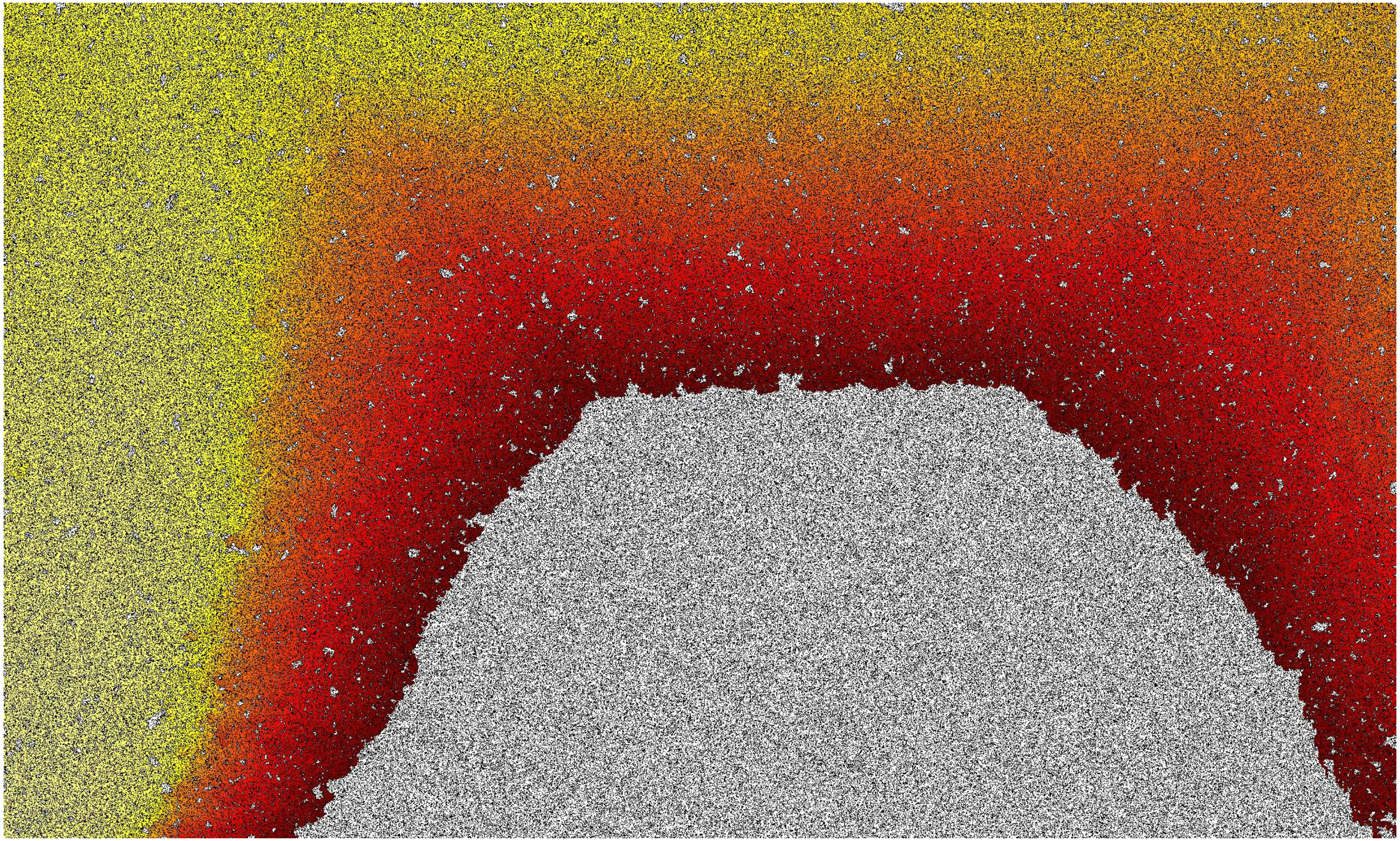
```
1.  while open is not empty
2.      select n ∈ open with minimum f(n)
3.      if n is a goal
4.          update incumbent solution
5.          empty delay into open
6.          increment window
7.          set min_d to inf
8.      if d(n) − min_d > window add n to delay
9.      otherwise for each child c of n
10.         if d(c) < min_d then min_d := d(c)
11.         if d(c) − min_d ≤ window add c to open
12.         otherwise add c to delay
13. if delay is not empty
14.     empty delay into open, set min_d to inf,
15.     increment window, and goto 1
16. otherwise return incumbent
```

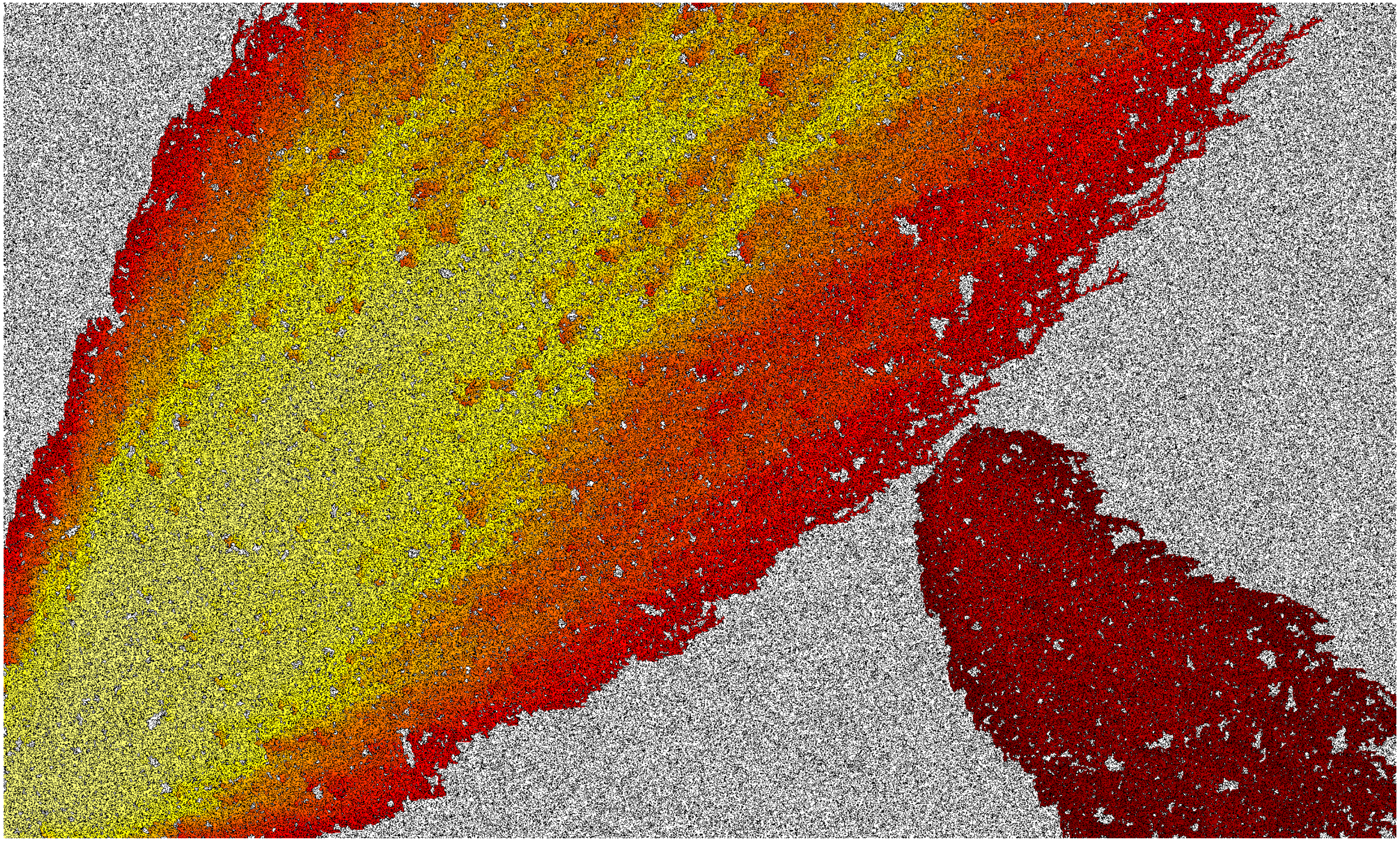

Korf's 100 15 Puzzles - Inverse Cost
d-Fenestration
Anytime Window A*
Solution Quality
0.8
0.4
0
0
100
200
300
raw cpu time

in domains with a large range of action costs,
cost-based search performs poorly.

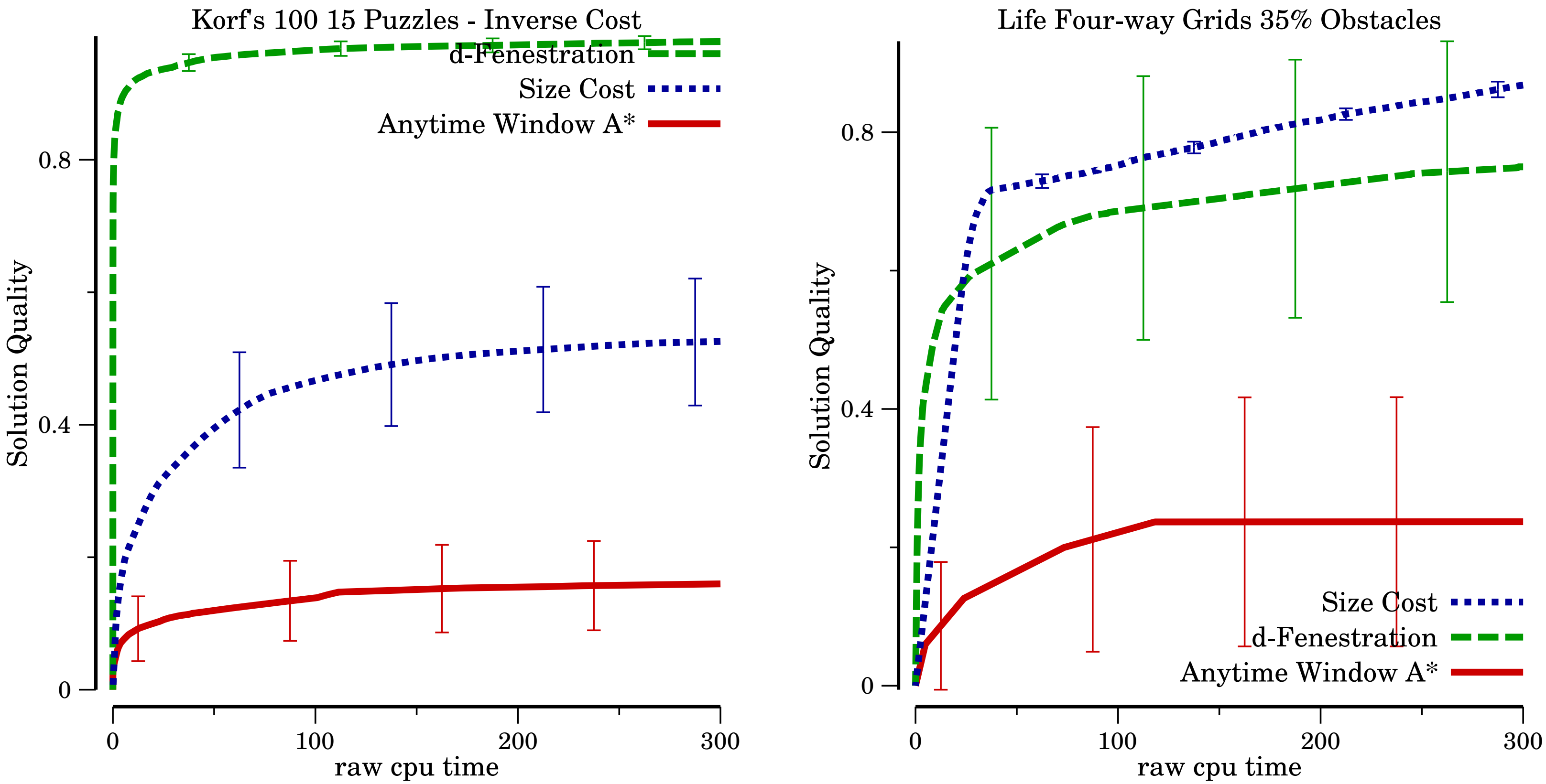
search on length instead, use pruning to converge on optimal.

1. run A* on length
2. keep going, pruning on $f(n)$.





1. while *open* has nodes
2. remove n from *open* with minimum $L(n)$
3. if n is a goal then set n as inc
4. otherwise for each child c of n
5. if $f(c) \leq f(inc)$ insert c into *open*
6. return inc



size-cost search handles duplicates better than d-Fenestration

for technical reasons, d-fenestration can't easily delay duplicates

Anytime Algorithms Based on Weighted A*

anytime weighted A*, Hansen et al 1997

1. run weighted A*
2. if you find a goal, keep going

anytime repairing A*, Likhachev et al, 2003

1. run weighted A*
2. if you find a duplicate, don't look at it just yet.
3. if you find a goal
4. dump duplicates into *open*, reduce w, keep going.

restarting weighted A*, Richter et al 2010

1. run weighted A*
2. if you find a goal, start over with a lower weight.

Anytime Algorithms Based on Weighted A*

continued search, Hansen and Zhou 1997

1. EES, A^*_ϵ, rdwA*, ...
2. if you find a goal, keep going.

repairing search, Likhachev et al, 2003

1. EES, A^*_ϵ, rdwA*, ...
2. if you find a duplicate, don't look at it just yet.
3. if you find a goal
4. dump duplicates into *open*, reduce w, keep going.

restarting search, Richter et al 2010

1. EES, A^*_ϵ, rdwA*, ...
2. if you find a goal, start over with a lower weight.

Anytime Algorithms Based on Weighted A*

specific results / guidelines not available

continued search, Hansen and Zhou 1997

- tight lower bound
- few cycles

repairing search, Likhachev et al, 2003

- many duplicates
- difficult to tune algorithm parameters

restarting search, Richter et al 2010

- cheap expansion
- heuristic bias in underlying search

- use d to force progress: d-fenestration
  - and make for fairer comparisons
- use d to guide search directly: size-cost search
  - rely on pruning to get high quality solutions
- use d-based algorithms in anytime frameworks
  - EES and A^*_ϵ both rely on d and work well
  - frameworks can make up for algorithm weaknesses
  - different problems have different best algorithms

Summary

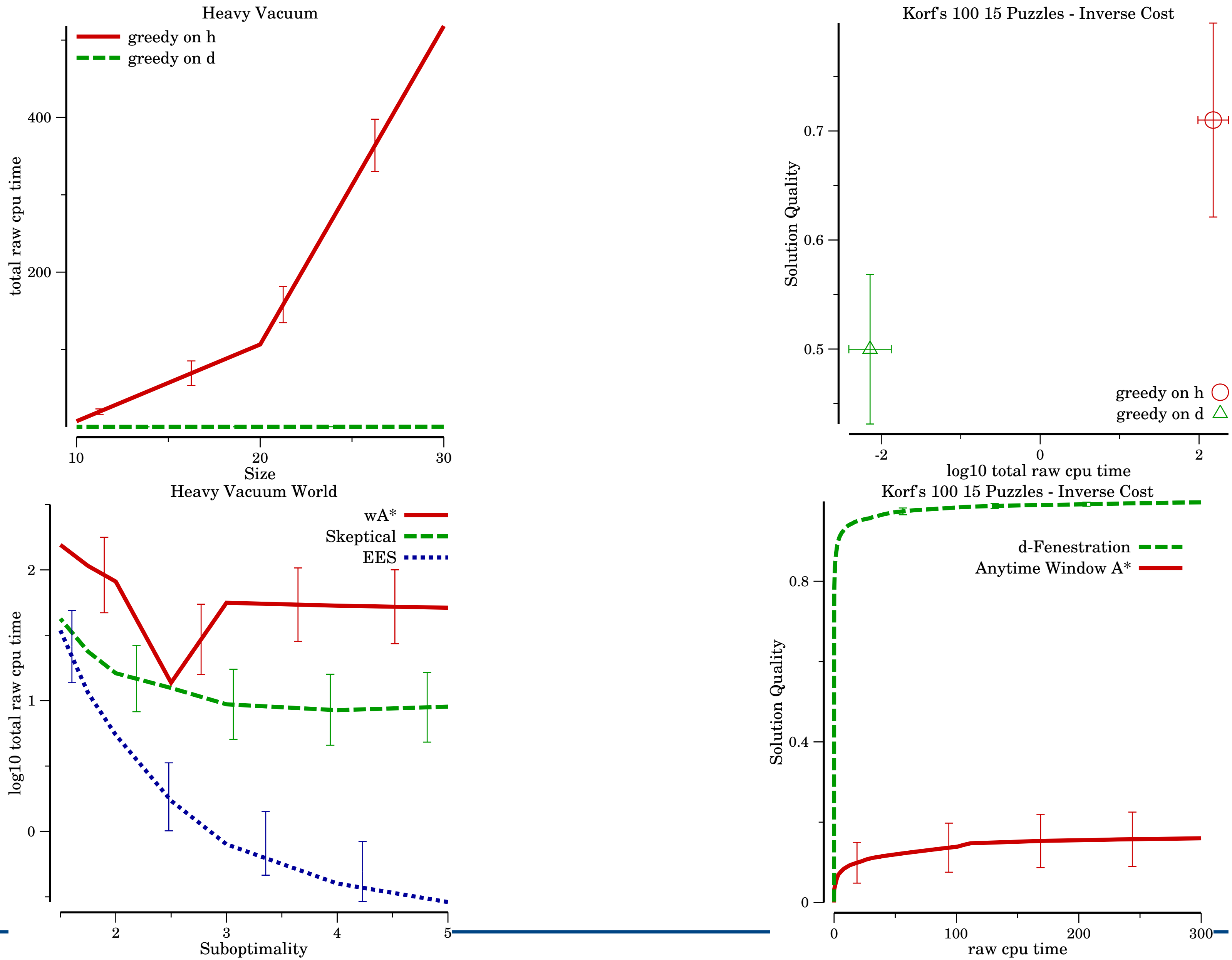
d estimates length

- d estimates solution length
  - $d_{nearest}$ estimates nearest solution
  - $d_{cheapest}$ estimates cheapest solution
- $d_{cheapest}$ and h are related
  - they can be computed at the same time
- compute $d_{nearest}$ by ignoring all cost information
- d estimates don't have to be admissible
  - admissible h is required for pruning, bounding

d is super effective!

- $d(n)$ in suboptimal search
  - d represents the quantity you're optimizing
  - $d_{cheapest}$ retains some cost information
  - useful in correcting h
- $d(n)$ in bounded suboptimal search
  - d represents the quantity you're optimizing
  - inadmissible d needn't compromise bounds
- $d(n)$ in anytime search
  - useful in finding solutions quickly
    - a key component with unknown deadlines

d is super effective!



Bibliography

- J. E. Dorand and D. Michie,
  "Experiments with the Graph Traverser Program",
  Proceedings of the Royal Society of London. Series A,
  Mathematical and Physical Sciences, 1966.
- Jordan T. Thayer and Wheeler Ruml,
  "Faster Than Weigthed A*:
  An Optimistic Approach to Bounded Suboptimal Search",
  ICAPS-2008.
- Jordan T. Thayer and Wheeler Ruml,
  "Bounded Suboptimal Search; A Direct Approach Using
  Inadmissible Estimates", IJCAI-2011.
- Jordan T. Thayer, Wheeler Ruml, and Jeff Kreis, "Using Distance Estimates in Heuristic Search: A Re-evaluation", SoCS-2009.
- Jordan T. Thayer and Wheeler Ruml, "Using Distance Estimates in Heuristic Search", ICAPS-2009.

Bibliography

- Jordan T. Thayer and Wheeler Ruml, "Learning Inadmissible Heuristics During Search", ICAPS-2011.
- Judea Pearl and Jin H. Kim, "Studies in Semi-Admissible Heuristics", IEEE PAMI volume 4, 1982.
- Gabrielle Röger and Malte Helmert, "The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning", ICAPS-2010.
- Chris Wilt and Wheel, "Cost-Based Heuristic Search is Sensetive to the Ratio of Operator Costs", SoCS-2011
- William Cushing, J Benton, and Subbarao Kambhampati, "Cost Based Search Considered Harmful", SoCS-2010.
- Ira Pohl, "The Avoidance of (Relative) Catastrophe, Heuristic Competence, Geuine Dynamic Weighting, and Computation Issues in Heuristic Problem Solving, IJCAI-1973.

Bibliography

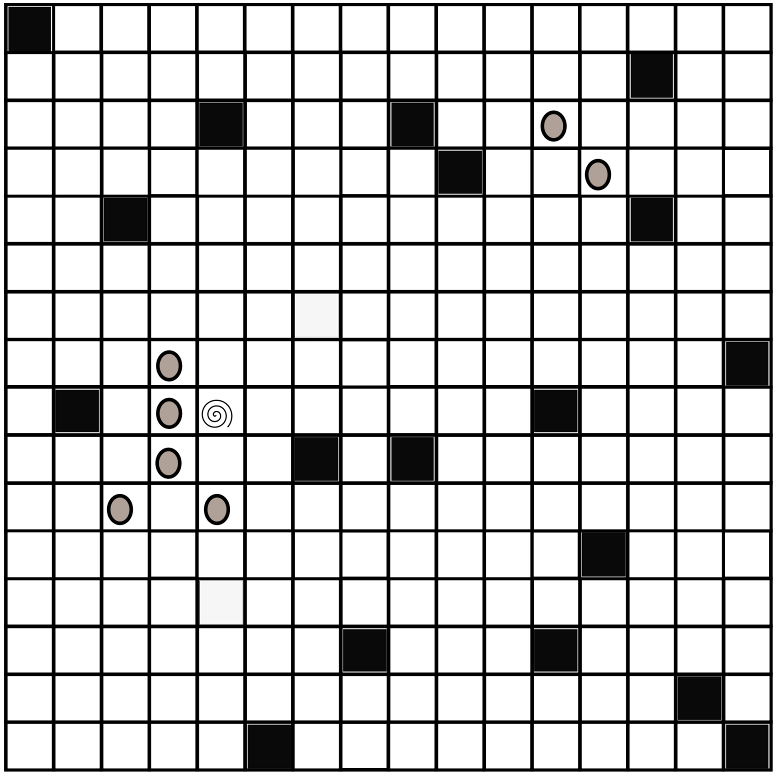
- Eric A. Hansen, Shlomo Zilberstein and Victor A. Danilchenko, "Anytime Heuristic Search: First Results", University of Massachusetts, Amherst Technical Repart 97-50, 1997.
- Maxim Likhachev, Geoff Gordon and Sebastian Thrun "ARA*: Anytime A* with Provable Bounds on Sub-Optimality", NIPS-2003.
- Eric A. Hansen and Rong Zhou, "Anytime Heuristic Search", Journal of Artificial Intelligence Research volume 28, 2007.

Bibliography

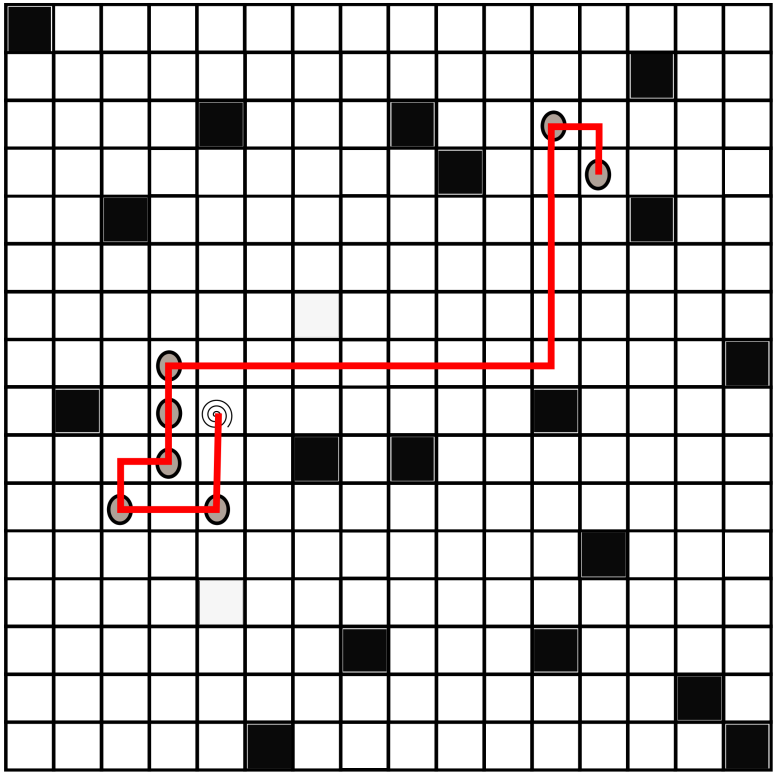
- Silvia Richter, Jordan Thayer and Wheeler Ruml, "The Joy of Forgetting: Faster Anytime Search via Restarting", ICAPS-2010.
- Jordan T. Thayer and Wheeler Ruml, "Anytime HEuristic Search: Frameworks and Algorithms", SoCS 2010.
- Sandip Aine, P.P. Chakrabarti, and Rajeev Kumal, "AWA* - A Window Constrained Anytime Heuristic Search Algorithm", IJCAI-2007.

Backup Slides

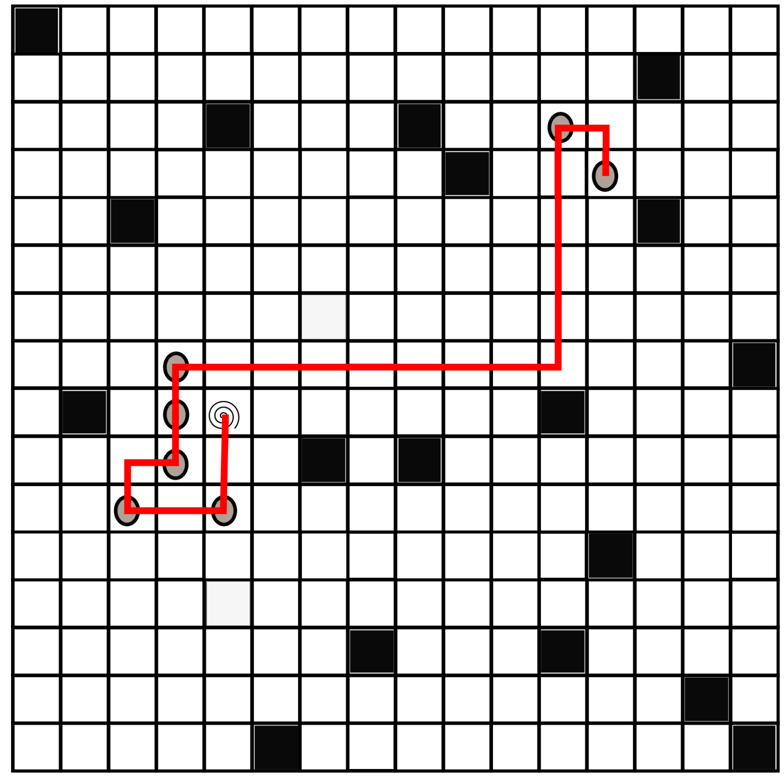
Heavy Vacuums Domain



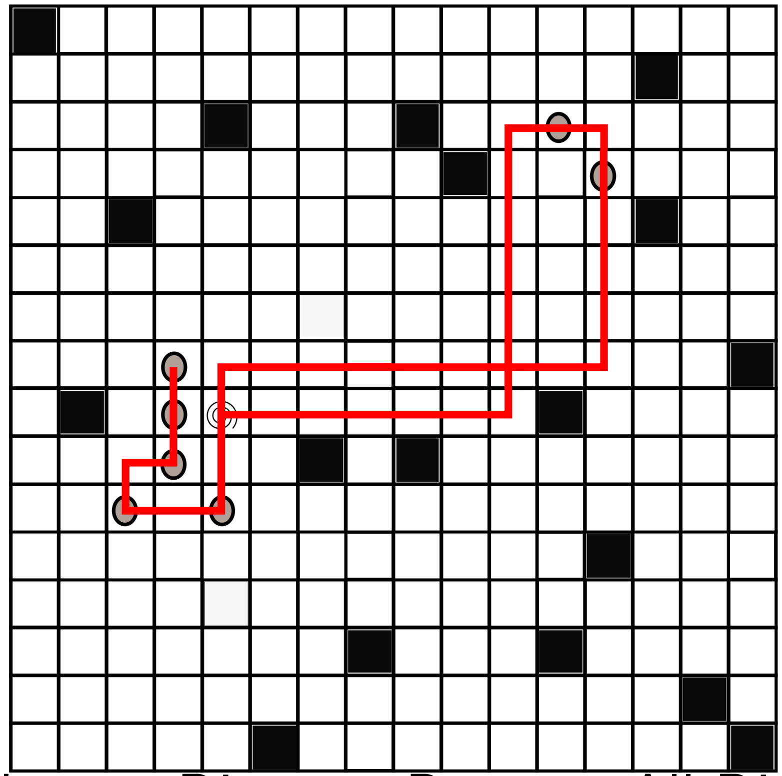
- Goal: Navigate To And Vacuum All Dirt
- Action Cost: $1 + \#$ Vacuumed Dirt



Estimate The Length Of The Shortest Possible Solution



1. Compute Manhattan Distance Between All Dirt Pairs
2. Build Minimum Spanning Tree Of Dirt Piles
3. Find Manhattan Distance From Vacuum To Dirt
4. Sum Edges In 2 And Minimum Of 3



1. Compute Manhattan Distance Between All Dirt Pairs
2. Build Minimum Spanning Tree Of Dirt Piles
3. Find Manhattan Distance From Vacuum To Dirt
4. Sort Edges In 2, Longest First
5. Iterate Over 4, Summing $i + weight \cdot edge$
6. Sum 5 And Minimum Of 3

1. Compute Manhattan Distance Between All Dirt Pairs
2. Build Minimum Spanning Tree Of Dirt Piles
3. Find Manhattan Distance From Vacuum To Dirt
4. Sort Edges In 2, Longest First
5. Iterate Over 4, Summing $i + weight \cdot edge$
6. Sum 5 And Minimum Of 3

---

1. Compute Manhattan Distance Between All Dirt Pairs
2. Build Minimum Spanning Tree Of Dirt Piles
3. Find Manhattan Distance From Vacuum To Dirt
4. Sum Edges In 2 And Minimum Of 3

Length Of Cheapest Solution

1. Compute Manhattan Distance Between All Dirt Pairs
2. Build Minimum Spanning Tree Of Dirt Piles
3. Find Manhattan Distance From Vacuum To Dirt
4. Sort Edges In 2, Longest First
5. Iterate Over 4, Summing $i + weight \cdot edge$
6. Sum 5 And Minimum Of 3

1. Assume No Obstacles
2. Greedily Solve The Problem
3. Report Solution Length