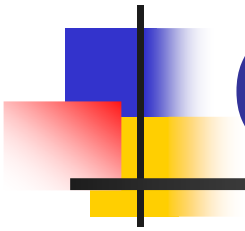
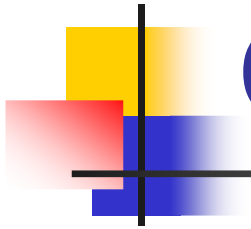


On Transposition Tables for Single-Agent Search and Planning: Summary of Results (Akagi, Kishimoto, Fukunaga) [1]

Seminar Search and Optimization

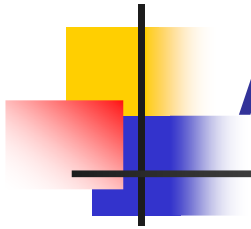
Basel, 23.10.2012

Christian Mächler



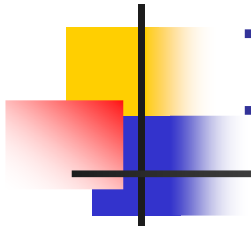
Overview

- Review of A* and IDA*
- Transposition Tables
- IDA* + TT
  - Case studies
  - Experimental results
- Questions



A* search algorithm

- Best-first search
- Finds the least cost path with an admissible heuristic
- Path-cost function g(x)
- Heuristic estimate h(x)
- Next node: lowest f(x) = g(x) + h(x)
- Open set: Priority queue of nodes to be traversed
- Closed set: Nodes already visited



Iterative Deepening A* (IDA*)

- Iterative deepening depth-first search
- Uses the f-costs as next limit/bound
- Costs exceed limit → path cut off
- The limit for the next iteration is set to the lowest-cost node that was pruned during the previous iteration
- Finds an optimal solution if an admissible heuristic is used

IDA* Pseudocode

function DFS($n, bound, path$): **real**

1: **if** n is a goal state **then**
2: $solved := true$; $answer := path$; **return** (0);
3: **end if**
4: **if** successors$(n) = \emptyset$ **then**
5: $new\_bound := \infty$;
6: **else**
7: $new\_bound :=$ min$\{$BD$(m) | m \in$ successors$(n)\}$;
8: **end if**
9: **return** (new_bound);

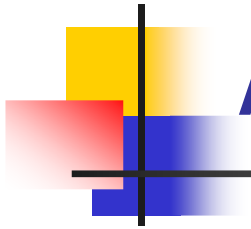
where BD(m) :=

Case 1: ∞, if $path + m$ forms a cycle

Case 2: $c(n, m) +$ DFS$(m, bound - c(n, m), path + m)$,
if $c(n, m) + h(m) \leq bound$

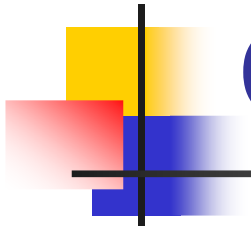
Case 3: $c(n, m) + h(m)$, if $c(n, m) + h(m) > bound$

[1]



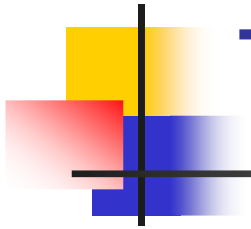
A* versus IDA*

	A*	**IDA***
Memory requirements	large	small
Re-expansion of duplicate states reached via different paths	no (with a consistent heuristic)	yes



Consistent heuristic

- h(G) = 0
- h(N) <= c(N,P) + h(P)
- P: successor of N
- c(N,P): costs from N to P



Transposition Table (TT)

- Hash table of positions/nodes analyzed so far up to a certain depth
- Used to avoid re-expanding the same node (or re-evaluating the same position)
- Usually not enough memory → a replacement strategy/policy has to be used
- For IDA* the TT is a cache where the keys are states and the entries contain the estimated cost to a solution state

DFSTT1 a straightforward extension of DFS

function DFSTT1($n, bound, path$): **real**

1: **if** n is a goal state **then**
2: $solved := true$; $answer := path$; **return** (0);
3: **end if**
4: **if** successors$(n) = \emptyset$ **then**
5: $new\_bound := \infty$;
6: **else**
7: $new\_bound := \min\{\text{BD}(m) | m \in \text{successors}(n)\}$;
8: **end if**
9: store (n, new_bound) in TT;
10: **return** (new_bound);

where BD(m) :=

Case 1: ∞, if $path + m$ forms a cycle

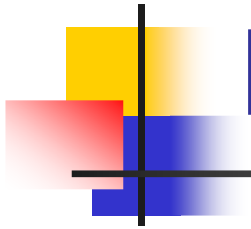
Case 2: $c(n, m) + \text{DFSTT1}(m, bound - c(n, m), path + m)$,
if $c(n, m) + \text{Lookup}(m) \leq bound$

Case 3: $c(n, m) + \text{Lookup}(m)$,
if $c(n, m) + \text{Lookup}(m) > bound$

function Lookup(m, TT): **real**

1: **if** m is is in TT **then**
2: **return** $esti(m)$;
3: **else**
4: store $(m, h(m))$ in TT
5: **return** $h(m)$
6: **end if**

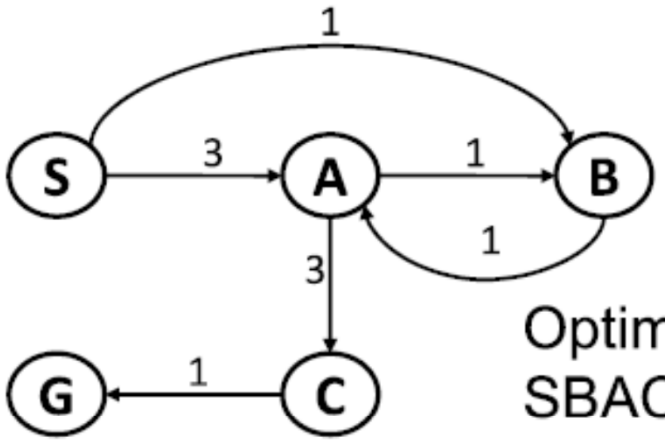
[1]



Properties of DFSTT1

- Given a consistent heuristic, IDA* using DFSTT1 with an infinite capacity TT is admissible
- Given a consistent heuristic, IDA* using DFSTT1 with a finite-capacity TT is not admissible (for some replacement policies)
- Given an admissible inconsistent heuristic, IDA* using DFSTT1 is not admissible

Counterexample



Optimal path:
SBACG (cost 6)

$$c(S,A) = c(A,C) = 3$$

$$c(S,B) = c(A,B) = c(B,A) = c(C,G) = 1$$

$$h(S) = 2, h(A) = 2, h(B) = h(C) = 1$$

Successor ordering for S: A, B
Successor ordering for A: B, C

[1]

DFSTT2

function DFSTT2($n, bound, path$): (**real**,**real**)

1: **if** n is a goal state **then**
2: $solved := true$; $answer := path$; **return** $(0, 0)$;
3: **end if**
4: **if** successors$(n) = \emptyset$ **then**
5: $new\_esti := \infty$; $new\_bound := \infty$;
6: **else**
7: $new\_esti := \min\{\text{ET}(m) | m \in \text{successors}(n)\}$;
8: $new\_bound := \min\{\text{BD}(m) | m \in \text{successors}(n)\}$;
9: **end if**
10: store (n, new_esti)in TT;
11: **return** (new_esti, new_bound);

Where ET(m) :=

Case 1: $c(n, m) + \text{Lookup}(m)$, if $path + m$ forms a cycle

Case 2: $c(n, m) + \text{DFSTT2}(m, bound - c(n, m), path + m)[0]$,
if $c(n, m) + \text{Lookup}(m) \leq bound$

Case 3: $c(n, m) + \text{Lookup}(m)$,
if $c(n, m) + \text{Lookup}(m) > bound$

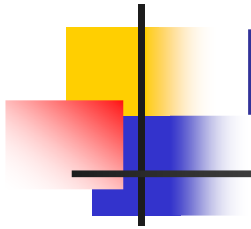
BD(m) :=

Case 1: ∞, if $path + m$ forms a cycle

Case 2: $c(n, m) + \text{DFSTT2}(m, bound - c(n, m), path + m)[1]$,
if $c(n, m) + \text{Lookup}(m) \leq bound$

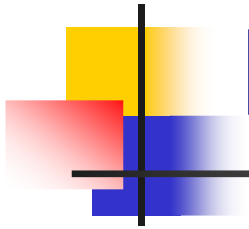
Case 3: $c(n, m) + \text{Lookup}(m)$,
if $c(n, m) + \text{Lookup}(m) > bound$

[1]



Properties of DFSTT2

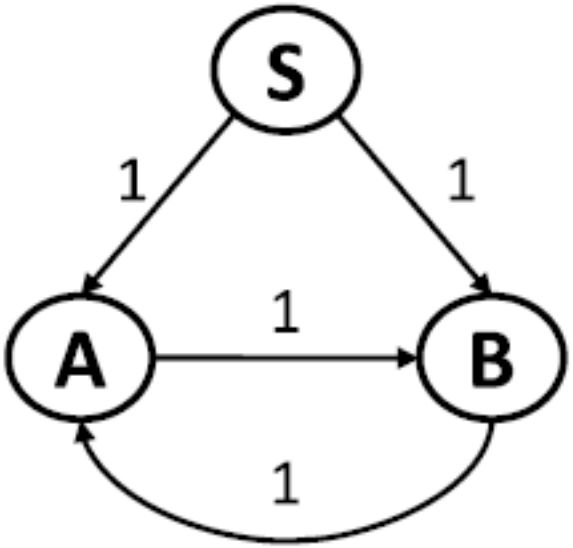
- Given an admissible heuristic function IDA* + DFSTT2 is admissible
- But IDA* + DFSTT2 is not complete



RollingStone strategy (RS)

- Stores *bound – g(n) + 1* before searching the subtree
- g(n): costs to reach node n
- Cycling back into this state will cause a cutoff because g(s) will be higher than its previous value
  → no cycle detection needed
- Admissible

Counterexample

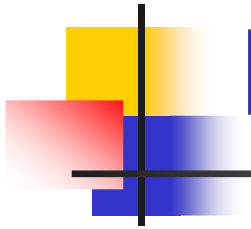


No solution

$$c(S,A) = c(S,B) = 1$$

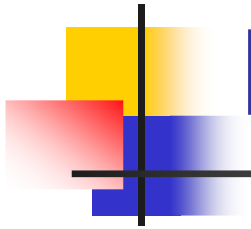
$$h(S) = h(B) = 2, h(A) = 1$$

[1]



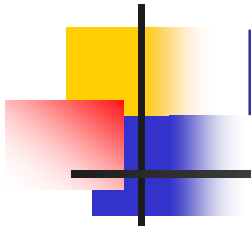
DFSTT3

- Not only store esti, but also the associated g-cost
- Allows to determine if a revisited node was already reached via a shorter path
- This allows to label dead ends
- Complete
- But large performance degradation



Replacement Policies

- No replacement
- Stochastic Node Caching
- Collision-based replacement
- Batch replacement
  Sort criterias:
  - subtree size
  - backed up cost estimate
  - # accesses

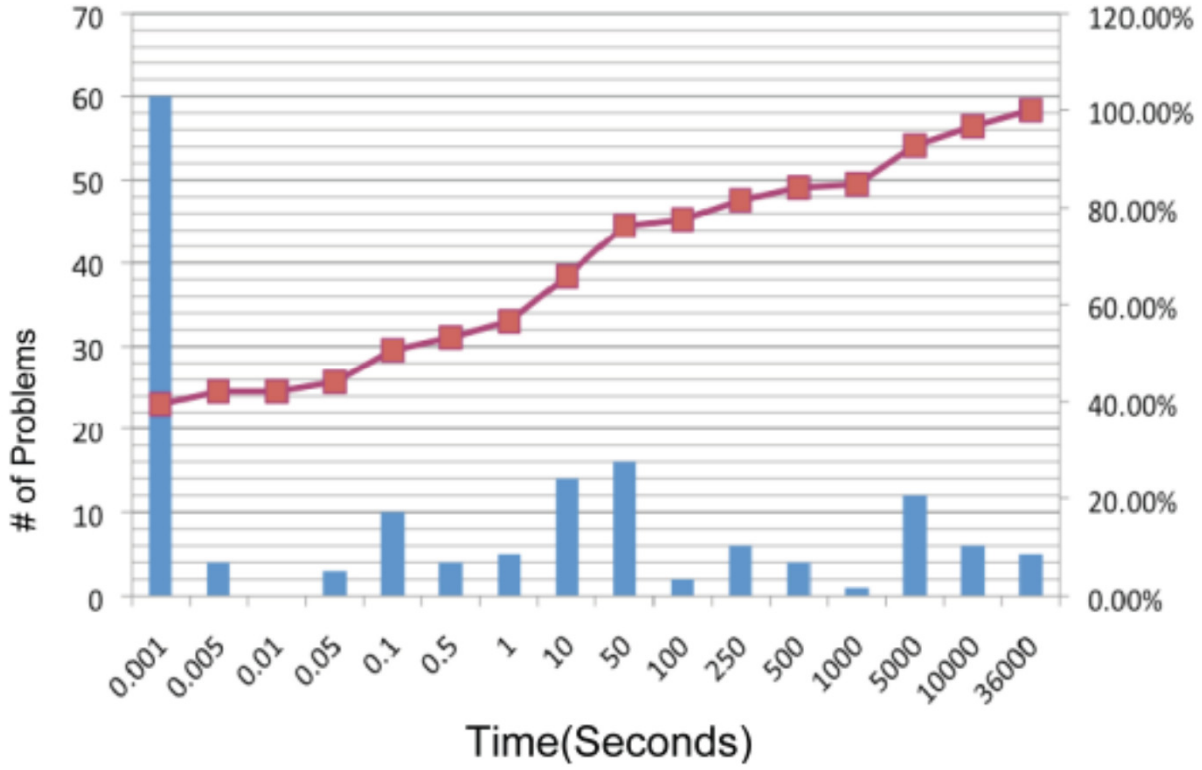


Experimental results

Algorithm	TT Replacement Policy	Num Solved	Tot. Runtime (seconds)
A*		173	539
DFS	No TT	128	178098
DFSTT2	TT, No Replace	183	73477
DFSTT2	Replace 0.3, subtree size	194	52256
RS	Replace 0.3, subtree size	195	68319
DFSTT2+RS	TT, No Replace	189	106147
DFSTT2+RS	Stochastic Caching, p=0.001	187	213765
DFSTT2+RS	Replace 0.3, est	194	40290
DFSTT2+RS	**Replace 0.3, subtree size**	**195**	**66960**
DFSTT2+RS	Replace 0.3, access freq.	194	39187
DFSTT2+RS	Collision, est	189	141249
DFSTT2+RS	Collision, subtree size	192	114057
DFSTT3+RS	Replace 0.3, subtree size	152	170296

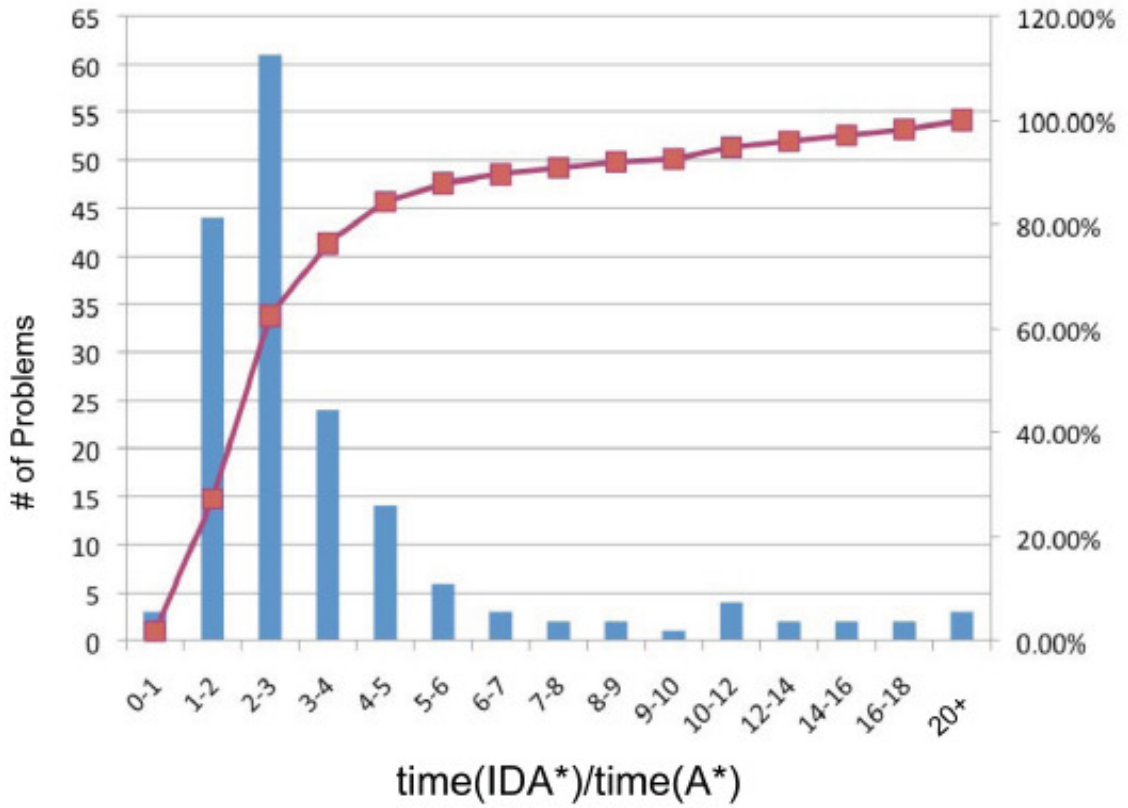
[1]

Experimental results

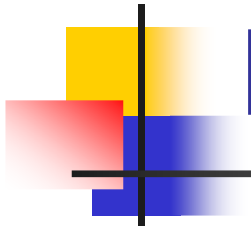


Runtime distribution of IDA* + DFSTT2 [1]

Experimental results



IDA* + DFSTT2 + RS vs. A* performance [1]



End of presentation

- Questions?