Aalto University
School of Science

A! - A Cooperative Heuristic Search Algorithm

Antti Halme

antti.halme@aalto.fi

August 19, 2014

Outline

Cooperation and heuristic search

The A! algorithm

Solving *m*-puzzles with A!

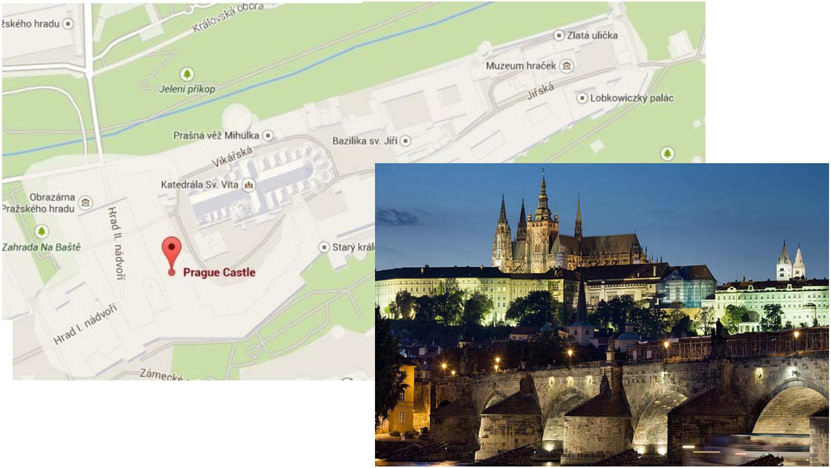
Outline

Cooperation and heuristic search

The A! algorithm

Solving *m*-puzzles with A!

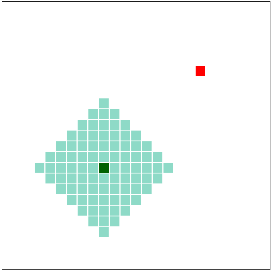
Intuition: Treasure Hunting



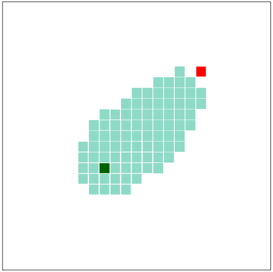
Google Maps; Jorge Royan / Wikimedia Commons

Heuristic Search

- Searching is a fundamental computing task
- A heuristic provides focus to searching efforts
- The most well-known heuristic search algorithm is the **A***



Uninformed search



Heuristic search

A^* search[1]

- Informed best-first graph search algorithm
- Single-source shortest path problem (SSSP)
- Fringe nodes ranked by a cost estimate

$$\underbrace{f(n)}_{\text{cost estimate}} = \underbrace{g(n)}_{\text{known distance}} + \underbrace{h(n)}_{\text{estimated remaining}}$$

- Admissible heuristics never overestimate, $h(n) \leq h^*(n)$
- Consistency gives optimality, $h(n) \leq d(n, n') + h(n')$.
- Optimally efficient on given heuristic
  - No algorithm can expand fewer node
  - **Except in tie-break situations**

[1] Peter Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100-107, 1968.

Constructing cooperative search

- **The goal**
  - Search faster as a collective (leverage parallel hardware)
- **The hypothesis**
  - Cooperating search agents outperform agents in isolation
- **The idea**
  - Cooperation is communication
  - Agents share and make good use of progress information
- **The method**
  - A secondary ranking heuristic, a dynamic tiebreaker
- **The mechanism**
  - Asynchronous messaging, natural concurrency, implicit randomness, nondeterministic exploration

Unus pro omnibus, omnes pro uno.

Outline

Cooperation and heuristic search

The A! algorithm

Solving *m*-puzzles with A!

Overview of A!

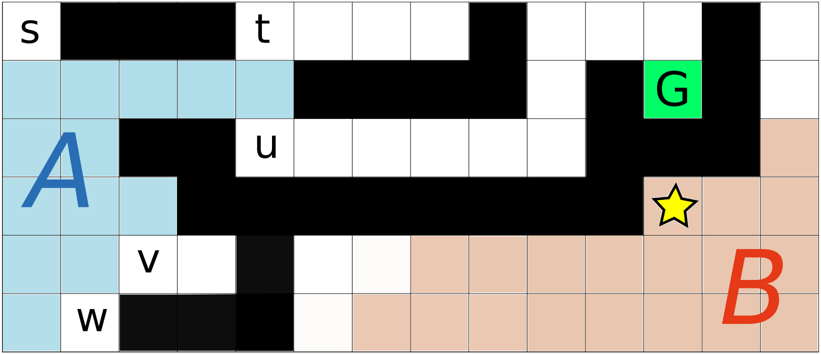
- A^* + cooperation + concurrency = A! *(a-bang)*
- A cooperative heuristic search algorithm for the SSSP
- Search agents run an upgraded version of A^*
- Shared information included as a secondary heuristic, $\hat{h}$
- Simplest case: share best encountered $(n, h(n))$-pair

- Vanilla A^* maintains a node priority queue sorted by

$$f(n) = g(n) + h(n),$$

A! takes k **equal-valued** nodes from the top, ranks by $\hat{h}$

Overview of A!



A! search on a grid graph with two agents, *A* and *B*. Between nodes *t* and *u*, of equal distance to *G (1st heuristic)*, agent *A* chooses *u*, because its closer *(2nd heuristic)* to best node marked with star, discovered by *B*.

A! in a nutshell

A!Search

- Launch N workers in parallel, wait for one to terminate

A!Solver

- At each agent, maintain a fringe node priority queue
  - Sorted by $f(n) = g(n) + h(n)$
- Take k **equal-valued** nodes from the top as list `candidates`

A!Select

- Get updates about global state
- Rank `candidates` by $\hat{h}$, select best as next to be explored
- Send an update back if found an improving node

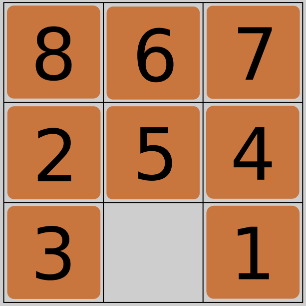
Outline

Cooperation and heuristic search

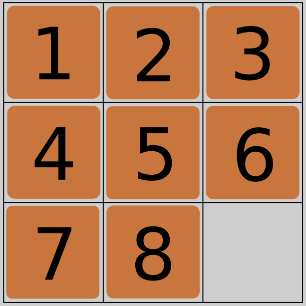
The A! algorithm

Solving *m*-puzzles with A!

The *m*-puzzle



A start state



A standard goal state

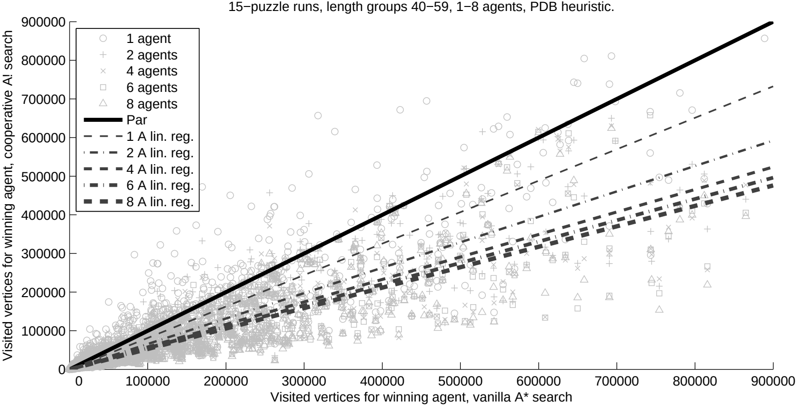
Computational setting

- Randomly generated 8- and 15-puzzle instances
- Grouped by optimal path length
  - Substantial variance within each group

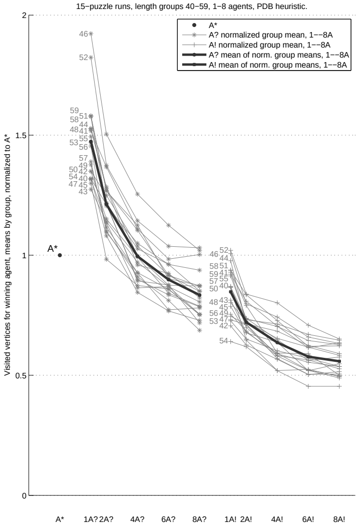
- Simple A! implementation based on A^* by Brian Borowski
- Focus on **nodes opened by winning agent**
  - Correlates with runtime, total opened count
- Comparison with A^* and non-coop randomized A^*, A?
  - `A!Select` $\rightarrow$ `A*Select`, `A?Select`

- Test execution on Aalto SCI Science-IT project resources
  - *Triton* cluster of mixed multi-core blade servers
  - 2.6GHz Opteron 2435, 2.67GHz Xeon X5650, and 2.8GHz Xeon E5 2680 v2

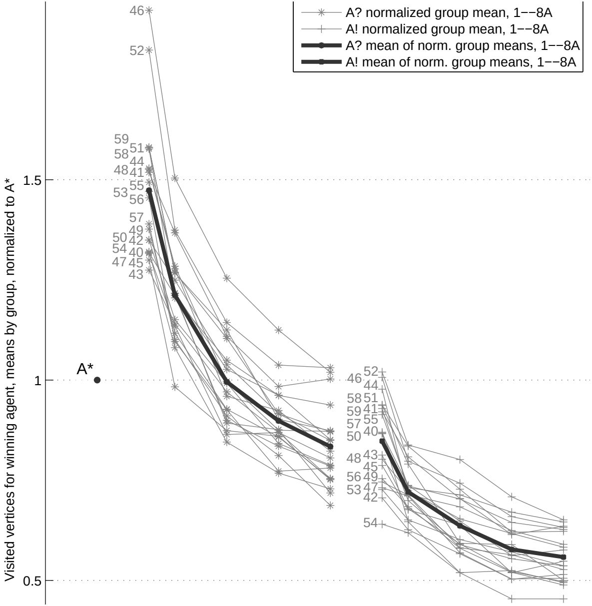
Results: Cooperation benefit and scalability



Relative performance of A* (x-axis) and A! (y-axis). The black line is par, so data points below it represent instances for which A! performs better than A*. Agent count is evaluated in five batches – 1, 2, 4, 6, and 8 agents – with the respective trend lines showing how the methods compare.



- A*-normalized length groups
- Scaling benefit from more agents
- Overlapping trend lines
    - Rudimentary asymptotics



A? normalized group mean, 1−−8A
A! normalized group mean, 1−−8A
A? mean of norm. group means, 1−−8A
A! mean of norm. group means, 1−−8A
Visited vertices for winning agent, means by group, normalized to A*
1.5
1
0.5
A*

Results: Discussion

- A! outperforms both vanilla A^* and the non-cooperative random parallel search A^*-variant A?
- Nondeterministic cooperation emerging from async message exchange was shown to be beneficial
- With more agents, A! was shown to work better
  - Rapidly diminishing returns
  - Search overhead appeared to be an issue

Conclusion

- Cooperation approach to parallel computing
- Heuristic search context, parallel A*
  - Cooperation as a secondary tiebreaking heuristic

- Cooperation is communication
  - Asynchronity, concurrency and nondeterminism
  - Ripple effects, implicit randomness

- **The A! algorithm**

- Empirical study, computational experiments
  - Performance on *m*-puzzles, scalability

Extra slides

Philosophy of Cooperation

co·op·er·ate

- to work together
- to work with another person or group to do something
- to be helpful by doing what someone asks or tells you to do
- to act in a way that makes something possible or likely
- to produce the right conditions for something to happen

(Merriam-Webster)

Optimality of A!

- Proof sketch based on an argument for A* itself[2]
- Cost estimates, f-values, are nondecreasing for all paths
  - Consistent heuristic
- Optimal path to a node is found before the node is opened
  - Expansion from the fringe, edge connection
- Let S be a subset of all equally good candidate nodes, E_S all explored nodes with a successor in S

1. $f(n')$ is the same for all $n' \in S$, so for $n \in E_S$

$$f(n') = g(n') + h(n') = g(n) + d(n, n') + h(n') \geq g(n) + h(n) = f(n)$$

2. For a node to be opened before an optimal path there is found, another fringe node m with a better f-value is implied, but then $f(n) < f(m)$ and $f(n) > f(m)$.

[2] Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach, 3rd ed. Prentice Hall Press, 2009.

A! cooperation architecture

- Agents process async messages at their own pace
  - Concurrency, nondeterminism; implicit randomness
- Best-effort notion of a globally superior reference node
- A *message broker* entity can facilitate communication
  - Instantaneous diffusion is *not* the goal
  - Publish/subscribe-topology *(cf. web chat)*
- Short-circuiting best-update leads to a **momentum effect**
  - Prefer nodes near the previously explored ones
  - Manifest already with a single agent
- Actor abstraction, fast termination, diversification, . . .

A! details – A!Search, A!Solver, A!Select

Algorithm 1 : A!Search

Require: $N > 0$, NODE *start*, PREDICATE *isGoal*, HEURISTIC h, HEURISTIC $\hat{h}$
Ensure: *path* from *start* to nearest node satisfying *isGoal* is shortest possible

$mb \leftarrow MsgBroker()$
for $i = 0$ to N **do**
  $workers[i] \leftarrow A!Solver(mb.portOut, mb.portIn, s, isGoal, h, \hat{h})$
end for
for each *worker* in *workers* **in parallel do**
  $worker.launch()$
end for

wait for termination
return $path \leftarrow getPath(workers)$

Algorithm 2 : A!Solver

Require: PORT *portIn*, *portOut*, NODE *start*, PRED *isGoal*, HEUR h, $\hat{h}$

```
openHeap ← FibonacciHeap⟨INTEGER, NODE⟩
closedSet ← Set⟨NODE⟩
pathMap ← Map⟨NODE, NODE⟩
current ← start
repeat
    if isGoal(current) then terminate(current, start, pathMap) end if
    closedSet.add(current)
    for each n in current.getNeighbors() do
        if closedSet.contains(n) then continue end if
        g ← current.g + dist(current, n)
        f ← g + h(n)
        improved ← openHeap.update(n, f)
        if improved then pathMap.update(n, current) end if
    end for
    peekList ← openHeap.getPeekList()
    if isEmpty(peekList) then terminate() end if
    current ← A!Select(peekList, portIn, portOut, h, ĥ)
    openHeap.remove(current)
until termination
```

Algorithm 3 : A!Select

Require: LIST *peekList*, PORT *portIn*, *portOut*, HEURISTIC h, $\hat{h}$
Ensure: *select* is the most promising node in *peekList* according to $\hat{h}$ on *best*

```
update, updateH ← asyncRecv(portIn)
if updateH < bestH then
    best, bestH ← update, updateH
end if
select ← peekList.pop()
selectD ← ĥ(select, best)
for each node in peekList do
    d ← ĥ(node, best)
    if d < selectD then select, selectD ← node, d end if
end for
if h(select) < bestH then
    best, bestH ← select, selectH
    asyncSend(portOut, {best, bestH})
end if
return select
```

A! tradeoffs and challenges

- Shared memory vs. distributed
- Cooperation details: what to share and how, usage
- Maintaining diversity: partitioning, hashing, ...
- IDA^* and maintaining memory-efficiency
- Clean, fast termination
- **Priority queue bottleneck**
- Agent abstraction: actors, coroutines, ...

The *m*-puzzle

- Classic sliding tile puzzle with a long history[3]
- Turn the start state into the target state by sliding tiles
- Literally a toy problem
  - Finding k-bound sequence for general $p \times p$-1 is NP-C[4]
- 8-puzzle avg branching factor $\sim$3, avg solution 22 steps
  - 3^{22} tree states, $\frac{9!}{2} \approx 180k$ graph
  - 15-puzzle 1.3×10^{12}, 24-puzzle $\sim 10^{25}$
- Hardest 8-p on 31 steps, 15-p 80, 24-p 152–208
  - cf. God's Number for the Rubik's Cube is 20 (Rokicki et al., **2010**)

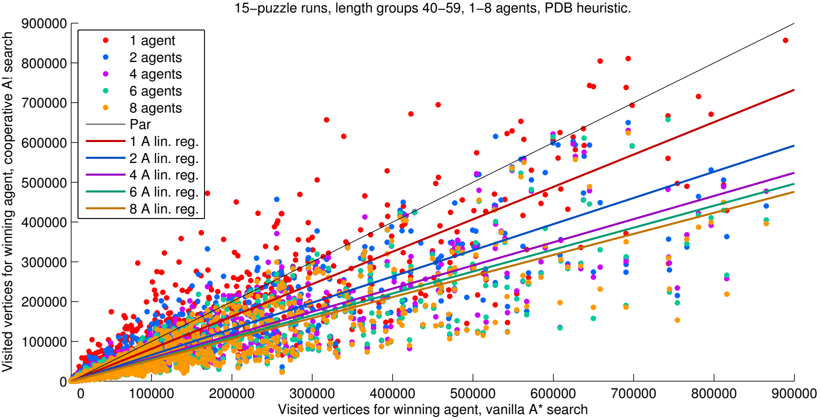
[3] Jerry Slocum and Dic Sonneveld. The 15 Puzzle Book. The Slocum Puzzle Foundation, 2006.

[4] Daniel Ratner and Manfred Warmuth. Finding a Shortest Solution for the NxN Extension of the 15-PUZZLE Is Intractable. AAAI '86, pages 168-172, 1986.

Heuristics for *m*-puzzle

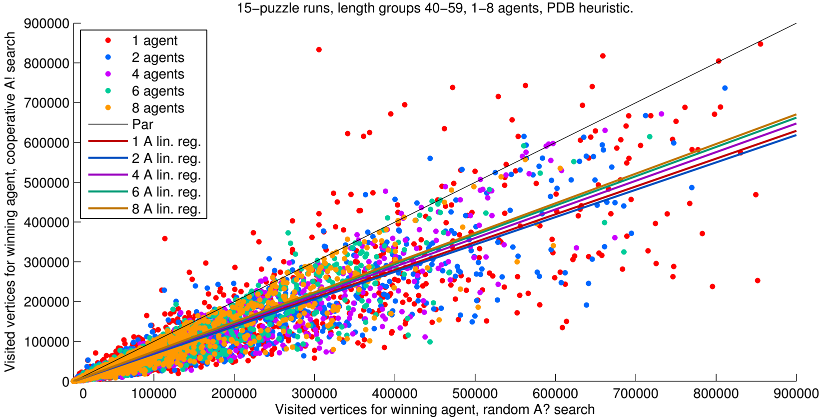
- Misplaced tile count
- **Manhattan distance**
  - The sum of distances the tiles are from their target positions, counted as moves along the grid
- **Manhattan distance with linear collisions**
  - Two tiles on the right row, but in the wrong order must pass each other to reach their targets
- Walking distance
- Pattern databases
- **Additive/disjoint pattern databases**
  - Store precomputed solutions to sub-problems; disjoint sets ensure combination heuristic remains admissible
- Learned heuristics

Results: Cooperation benefit and scalability



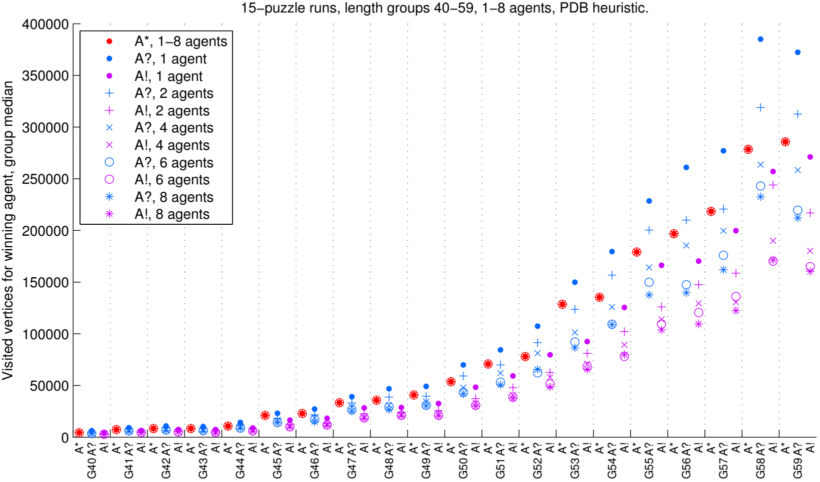
Relative performance of A* (x-axis) and A! (y-axis). The black line is par, so data points below it represent instances for which A! performs better than A*. Agent count is evaluated in five batches – 1, 2, 4, 6, and 8 agents – with the respective trend lines showing how the methods compare.

Results: Cooperation benefit and scalability

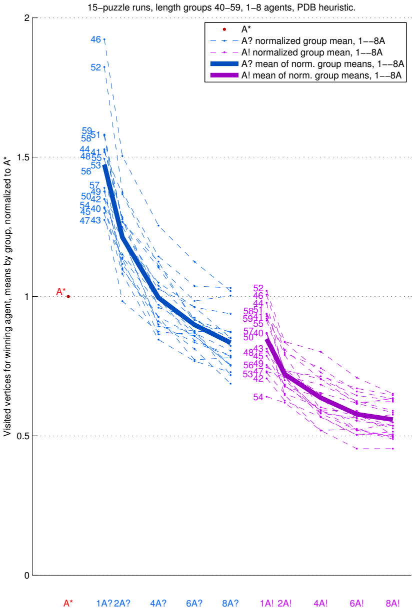


Relative performance of A? and A!. As before, the data points and trend lines below par-line reflect the benefit from cooperation. The slopes vary from around $\frac{7}{10}$ to $\frac{8}{10}$, reflecting a $25 - 40\%$ performance difference in favor of A!.

Results: Cooperation benefit and scalability

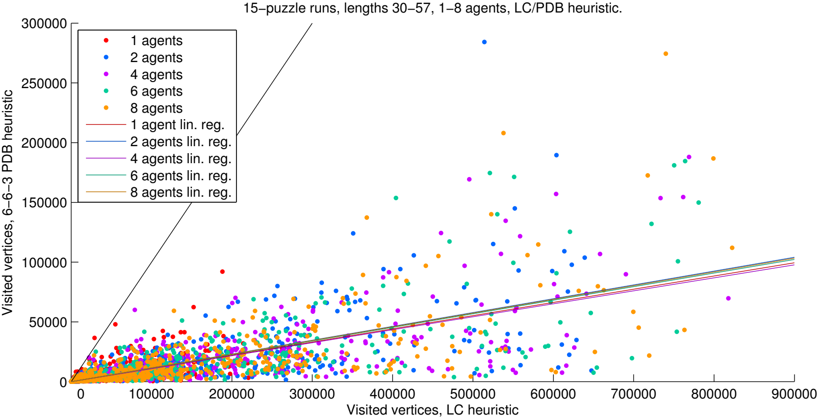


General A^*, A? and A! performance trends. 100 inst./group. 1, 2, 4, 6, and 8 agents. Cooperation appears to be more beneficial with harder instances.



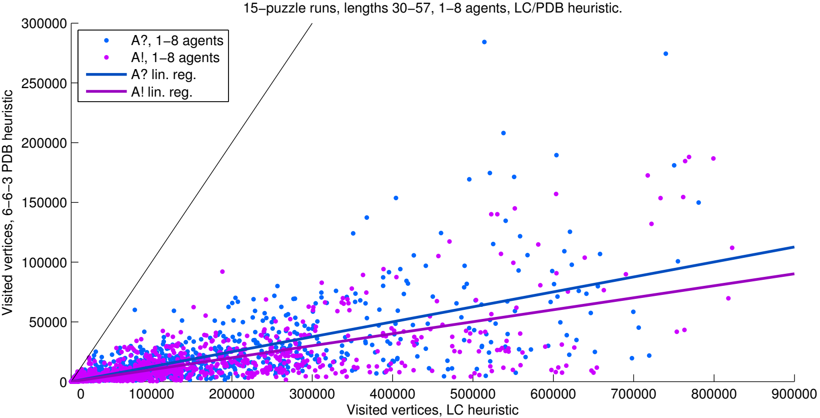
- A^*-normalized length groups
- Scaling benefit from more agents
- Overlapping trend lines
  - Rudimentary asymptotics

Results: Heuristic impact



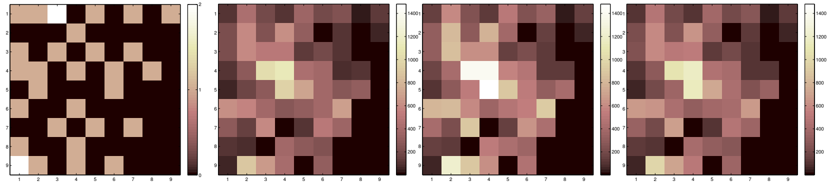
Heuristic comparison with data grouped by agent configuration. The trend lines being essentially the same indicates that the number of agents is not strongly correlated with heuristic impact: regardless of method, two agents benefit from a better heuristic as much (or little) as eight agents.

Results: Heuristic impact



Heuristic comparison with data grouped by method. The now more visible difference in trend lines suggests that A! benefits more from the improved heuristic than A?. The slope is about $\frac{1}{8}$ for A?, and around $\frac{1}{10}$ for A!.

Results: Path diversity



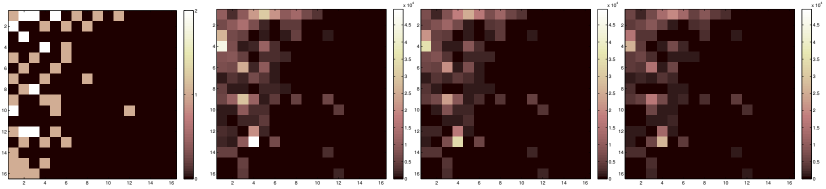
Solution path　　Vanilla A*　　Random A?　　Cooperative A!

Visualization of A*, A? and A! on 8-puzzle [`8 6 7 2 5 4 3 0 1`]. The heat-maps are derived from a 9×9 self-organizing map trained on an optimal solution path of 31 steps, shown top left in the codomain.

METHOD	A*				A?				A!			
FREQ.	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
MEAN	648	414	269	4780	3290	1147	903	5245	562	600	1006	4544
RATIO	0.11	0.07	0.04	0.78	0.31	0.11	0.09	0.50	0.08	0.09	0.15	0.68
STD	476	363	235	525	1345	559	327	716	233	254	372	656

State visits for A*, A? and A!. The table gives the mean, ratio and standard deviation of state visit frequencies from ten iterations of A* on four agents with Manhattan heuristic.

Results: Path diversity



Solution path　　Vanilla A*　　Random A?　　Cooperative A!

16 $\times$ 16 SOM visualization of A*, A? and A! on 54-optimal 15-puzzle [12 8 6 3 13 4 2 7 0 9 15 5 14 10 11 1].

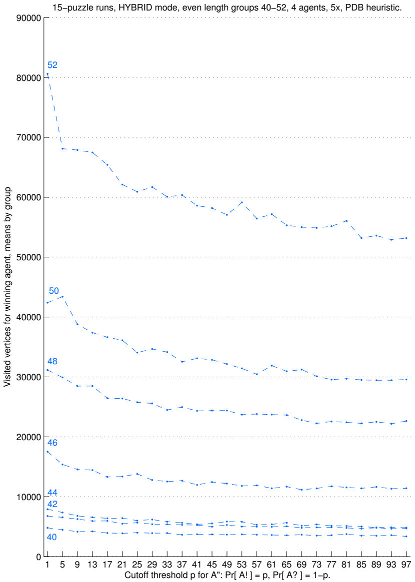
METHOD	A*				A?				A!			
FREQ.	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
MEAN	661	862	346	158124	91459	32347	13256	76659	40168	8406	4253	75356
RATIO	0.00	0.01	0.00	0.99	0.43	0.15	0.06	0.36	0.31	0.07	0.03	0.59
STD	352	1573	379	1420	27768	27092	14508	4702	29408	8212	1800	1040

State visits for A*, A? and A! on 15-puzzle [12 8 6 3 13 4 2 7 0 9 15 5 14 10 11 1]. The table gives the mean, ratio and standard deviation of state visit frequencies from ten iterations of A* on four agents with a 6-6-3 disjoint pattern database heuristic.

Results: Hybrid performance

- Diminishing returns from simply adding more agents
  - Search overhead, new agents mostly tread on old paths
- Path diversity is essential
  - Secondary heuristic performance depends on it
- Ideally, one would like to have the focused search performance of A! *and* the diversity apparent in A?
  - A! + A? = A″

- Simple threshold combination does not appear to work
  - A! seems to skip past the very states that A? wastes time on



- Hybrid A$''$ performance over multiple p-thresholds
- Three iterations per instance per group in the 45–54 range
- A! likelihood over A? grows with p to the right
    - The downward trending slopes suggest that adding some A? elements into A! does not improve overall performance

Related work

- Alba, E. (Ed.). (2005). Parallel Metaheuristics. John Wiley & Sons. doi:10.1002/0471739383
- Alba, E., Luque, G., & Nesmachnow, S. (2013). Parallel Metaheuristics: Recent Advances and New Trends. Intl. Trans. in Operational Research, 20(1), 1âĂŞ48. doi:10.1111/j.1475-3995.2012.00862.x
- Barbucha, D. (2012). Search Modes for the Cooperative Multi-agent System Solving the Vehicle Routing Problem. Neurocomputing, 88, 13âĂŞ23. doi:10.1016/j.neucom.2011.07.032
- Burns, E., Lemons, S., Ruml, W., & Zhou, R. (2010). Best-First Heuristic Search for Multicore Machines. Journal of Artificial Intelligence Research, 39, 689âĂŞ743. doi:10.1613/jair.3094
- Clearwater, S. H., Huberman, B. A., & Hogg, T. (1991). Cooperative Solution of Constraint Satisfaction Problems. Science, 254(5035), 1181âĂŞ3. doi:10.1126/science.254.5035.1181
- Crainic, T. G., & Toulouse, M. (2008). Explicit and Emergent Cooperation Schemes for Search Algorithms. In Proc. of the 2nd Intl. Conf. on Learning and Intelligent Optimization (LION '07) (pp. 95âĂŞ109). doi:10.1007/978-3-540-92695-5_8
- Hogg, T., & Huberman, B. A. (1993). Better Than the Best: The Power of Cooperation. In 1992 Lectures in Complex Systems (Vol. V, pp. 165âĂŞ184). Addison-Wesley.
- Hogg, T., & Williams, C. P. (1993). Solving the Really Hard Problems with Cooperative Search. In Proc. of the 11th National Conference on Artificial Intelligence (AAAI '93) (pp. 231âĂŞ236).
- Kishimoto, A., Fukunaga, A., & Botea, A. (2013). Evaluation of a Simple, Scalable, Parallel Best-First Search Strategy. Artificial Intelligence, 195(0), 222âĂŞ248. doi:10.1016/j.artint.2012.10.007
- Nitschke, G. (2005). Emergence of Cooperation: State of the Art. Artificial Life, 11(3), 367âĂŞ96. doi:10.1162/1064546054407194

- Halme, A. (2014). Cooperative Heuristic Search with Software Agents. M.Sc. thesis, Aalto University.