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 $A^*$
A* search

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

\[ f(n) = g(n) + h(n) \]

- Cost estimate
- Known distance
- 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

---

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 → 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

![Graph showing cooperation benefit and scalability](image)

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! - A Cooperative Heuristic Search Algorithm

Aalto University
School of Science

Antti Halme
13/15

Visited vertices for winning agent, means by group, normalized to 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  
  - Asynchrony, 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
- 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 $f(n) > f(m)$.

---

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, ...
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

\[
mb \leftarrow \text{MsgBroker}()
\]

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 \( path \leftarrow \text{getPath}(workers) \)
Algorithm 2: A!Solver

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

1: \hspace{1em} openHeap ← FibonacciHeap\langle\text{INTEGER, NODE}\rangle
2: \hspace{1em} closedSet ← Set\langle\text{NODE}\rangle
3: \hspace{1em} pathMap ← Map\langle\text{NODE, NODE}\rangle
4: \hspace{1em} current ← start
5: \hspace{1em} repeat
6: \hspace{2em} if isGoal(current) then terminate(current, start, pathMap) end if
7: \hspace{2em} closedSet.add(current)
8: \hspace{2em} for each n in current.getNeighbors() do
9: \hspace{3em} if closedSet.contains(n) then continue end if
10: \hspace{3em} g ← current.g + dist(current, n)
11: \hspace{3em} f ← g + h(n)
12: \hspace{3em} improved ← openHeap.update(n, f)
13: \hspace{3em} if improved then pathMap.update(n, current) end if
14: \hspace{2em} end for
15: \hspace{1em} peekList ← openHeap.getPeekList()
16: \hspace{1em} if isEmpty(peekList) then terminate() end if
17: \hspace{1em} current ← A!Select(peekList, portIn, portOut, h, \( \hat{h} \))
18: \hspace{1em} openHeap.remove(current)
19: \hspace{1em} 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
1: update, updateH ← asyncRecv(portIn)
2: if updateH < bestH then
3:   best, bestH ← update, updateH
4: end if
5: select ← peekList.pop()
6: selectD ← $\hat{h}(select, best)$
7: for each node in peekList do
8:   d ← $\hat{h}(node, best)$
9:   if d < selectD then select, selectD ← node, d end if
10: end for
11: if h(select) < bestH then
12:   best, bestH ← select, selectH
13: asyncSend(portOut, {best, bestH})
14: end if
15: 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 $1.3 \times 10^{12}$, 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 \(\text{(Rokicki et al., 2010)}\)

---


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

15-puzzle runs, length groups 40–59, 1–8 agents, PDB heuristic.

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

15-puzzle runs, length groups 40–59, 1–8 agents, PDB heuristic.

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

15–puzzle runs, length groups 40–59, 1–8 agents, PDB heuristic.
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

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.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>A*</th>
<th>A?</th>
<th>A!</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREQ.</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>MEAN</td>
<td>648</td>
<td>414</td>
<td>269</td>
</tr>
<tr>
<td>RATIO</td>
<td>0.11</td>
<td>0.07</td>
<td>0.04</td>
</tr>
<tr>
<td>STD</td>
<td>476</td>
<td>363</td>
<td>235</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>MEAN</td>
<td>3290</td>
<td>1147</td>
<td>903</td>
</tr>
<tr>
<td>RATIO</td>
<td>0.31</td>
<td>0.11</td>
<td>0.09</td>
</tr>
<tr>
<td>STD</td>
<td>1345</td>
<td>559</td>
<td>327</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>MEAN</td>
<td>562</td>
<td>600</td>
<td>1006</td>
</tr>
<tr>
<td>RATIO</td>
<td>0.08</td>
<td>0.09</td>
<td>0.15</td>
</tr>
<tr>
<td>STD</td>
<td>233</td>
<td>254</td>
<td>372</td>
</tr>
</tbody>
</table>

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

16 × 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].

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.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>A*</th>
<th>A?</th>
<th>A!</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>FREQ.</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>MEAN</td>
<td>661</td>
<td>862</td>
<td>346</td>
</tr>
<tr>
<td>RATIO</td>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>STD</td>
<td>352</td>
<td>1573</td>
<td>379</td>
</tr>
</tbody>
</table>
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