16-782
Planning & Decision-making in Robotics

Search Algorithms: Multi-goal A*, IDA*

Maxim Likhachev
Robotics Institute
Carnegie Mellon University
Agenda

• A* with multiple goals

• Iterative Deepening A* (IDA*)
Support for Multiple Goal Candidates

• How to compute a least-cost path to any one of the possible goals?
  – Example 1: Computing a least-cost path to a parking spot given multiple parking spaces (some are better, some are worse, some are closer, some are further)

  – Example 2: Catching a moving target whose future trajectory is known (i.e., multiple potential intercept points)

  – Example 3: Mapping/exploration (next class)
**Main function**

\[ g(s_{\text{start}}) = 0; \text{ all other } g\text{-values are infinite}; \ OPEN = \{s_{\text{start}}\}; \]

ComputePath();
publish solution;

**ComputePath function**

while (s_{goal} is not expanded and OPEN ≠ \emptyset)

- remove \( s \) with the smallest \( f(s) = g(s) + h(s) \) from OPEN;
- insert \( s \) into CLOSED;
- for every successor \( s' \) of \( s \) such that \( s' \) not in CLOSED
  - if \( g(s') > g(s) + c(s, s') \)
    - \( g(s') = g(s) + c(s, s') \);
    - insert \( s' \) into OPEN;

How to find a least-cost path that is lowest across all possible goals?
Introducing “imaginary” goal

Main function

\[ g(s_{\text{start}}) = 0; \text{ all other } g\text{-values are infinite}; \quad OPEN = \{s_{\text{start}}\}; \]
ComputePath();
publish solution;

ComputePath function

while \( s_{\text{goal}} \) is not expanded and \( OPEN \neq 0 \)

remove \( s \) with the smallest \( f(s) = g(s) + h(s) \) from \( OPEN \);
insert \( s \) into \( CLOSED \);

for every successor \( s' \) of \( s \) such that \( s' \) not in \( CLOSED \)

if \( g(s') > g(s) + c(s, s') \)

\[ g(s') = g(s) + c(s, s'); \]
insert \( s' \) into \( OPEN \);
Introducing “imaginary” goal

Main function
\[ g(s_{\text{start}}) = 0; \] all other \( g \)-values are infinite; \( OPEN = \{s_{\text{start}}\} \);
ComputePath();
publish solution;

ComputePath function
while \( s_{\text{goal}} \) is not expanded and \( OPEN \neq 0 \)
  remove \( s \) with the smallest \( f(s) = g(s) + h(s) \) from \( OPEN \);
  insert \( s \) into \( CLOSED \);
  for every successor \( s' \) of \( s \) such that \( s' \) not in \( CLOSED \)
    if \( g(s') > g(s) + c(s,s') \)
      \[ g(s') = g(s) + c(s,s') \];
      insert \( s' \) into \( OPEN \);

Equivalent problem but with a single goal!

How to prove it?
Support for “unequal” goals

Main function
\[ g(s_{start}) = 0; \text{ all other } g\text{-values are infinite}; \ OPEN = \{s_{start}\}; \]
ComputePath();
publish solution;

ComputePath function
while (s_{goal} is not expanded and OPEN \neq \emptyset)
  remove \( s \) with the smallest \( f(s) = g(s) + h(s) \) from OPEN;
  insert \( s \) into CLOSED;
  for every successor \( s' \) of \( s \) such that \( s' \) not in CLOSED
    if \( g(s') > g(s) + c(s,s') \)
      \( g(s') = g(s) + c(s,s') \);
    insert \( s' \) into OPEN;

What if some goals are better than others?
Main function

\[ g(s_{\text{start}}) = 0; \text{ all other } g\text{-values are infinite}; \ OPEN = \{s_{\text{start}}\}; \]

ComputePath();
publish solution;

**ComputePath function**

while \( s_{\text{goal}} \) is not expanded and \( OPEN \neq \emptyset \)

remove \( s \) with the smallest \( [f(s) = g(s) + h(s)] \) from \( OPEN \);

insert \( s \) into \( CLOSED \);

for every successor \( s' \) of \( s \) such that \( s' \) not in \( CLOSED \)

if \( g(s') > g(s) + c(s, s') \)

\( g(s') = g(s) + c(s, s') \);

insert \( s' \) into \( OPEN \);

---

What if some goals are better than others?
Support for “unequal” goals

Main function
\[ g(s_{\text{start}}) = 0; \text{ all other } g\text{-values are infinite}; \text{ OPEN } = \{s_{\text{start}}\}; \]
ComputePath();
publish solution;

ComputePath function
while(s\_goal is not expanded and OPEN \(\neq\) 0)
remove s with the smallest \([f(s) = g(s)+h(s)]\) from OPEN;
insert s into CLOSED;
for every successor s’ of s such that s’ not in CLOSED
if \(g(s’) > g(s) + c(s,s’)\)
\(g(s’) = g(s) + c(s,s’);\)
insert s’ into OPEN;

Equivalent problem but with a single goal!
How to prove it?
Support for “unequal” goals

**Main function**

\(g(s_{\text{start}}) = 0;\) all other \(g\)-values are infinite; \(OPEN = \{s_{\text{start}}\}\);

`ComputePath();`

`publish solution;`

**ComputePath function**

while \((s_{\text{goal}}\) is not expanded and \(OPEN \neq 0\))

- remove \(s\) with the smallest \([f(s) = g(s)+h(s)]\) from \(OPEN\);
- insert \(s\) into \(CLOSED\);
- for every successor \(s'\) of \(s\) such that \(s'\) not in \(CLOSED\)
  - if \(g(s') > g(s) + c(s,s')\)
    - \(g(s') = g(s) + c(s,s')\);
  - insert \(s'\) into \(OPEN\);

Once the graph transformation is done, you can run either forward or backwards search.
Main function
\[ g(s_{\text{start}}) = 0; \text{ all other } g\text{-values are infinite}; \ OPEN = \{s_{\text{start}}\}; \]
ComputePath();
publish solution;

\textbf{ComputePath function}
while(s_{\text{goal}} \text{ is not expanded and } OPEN \neq 0)
  remove \( s \) with the smallest \( [f(s) = g(s)+h(s)] \) from OPEN:
  insert \( s \) into CLOSED;
for every successor \( s' \) of \( s \) such that \( s' \) not in CLOSED
  if \( g(s') > g(s) + c(s,s') \)
    \( g(s') = g(s) + c(s,s') \);
  insert \( s' \) into OPEN;

Once the graph transformation is done, you can run either forward or backwards search

Any impact on how heuristics is computed?
Agenda

• A* with multiple goals

• Iterative Deepening A* (IDA*)
Memory Issues

- A* does provably minimum number of expansions (O(n)) for finding a provably optimal solution

- Memory requirements of weighted A* are often but not always better
Search with Linear Memory Requirement

- Depth-First Search (w/o coloring all expanded states):
  - explore each every possible path one at a time avoiding looping and keeping in the memory only the best path discovered so far

**DFS function**

LIFO list = \{s_{\text{start}}\}; //stack

bestpathsofar = NONE;

While (list != 0)
  s = list.pop();
  if (s = s_{\text{goal}})
    if (cost of the found path from s_{\text{start}} to s < cost of bestpathsofar)
      set bestpathsofar to the current path from s_{\text{start}} to s
  else
    for every successor s’ of s that is not on in list already
      list.push(s’);

return bestpathsofar;
Search with Linear Memory Requirement

- Depth-First Search (w/o coloring all expanded states):
  - explore each every possible path one at a time avoiding looping and keeping in the memory only the best path discovered so far

**DFS function**

LIFO list = \{s_{start}\}; //stack
bestpathsofar = NONE;

While (list != 0)
  s = list.pop();
  if (s = s_{goal})
    if (cost of the found path from s_{start} to s < cost of bestpathsofar)
      set bestpathsofar to the current path from s_{start} to s
    else
      for every successor s’ of s that is not on in list already
        list.push(s’);
  return bestpathsofar;

What is memory complexity?

What are its disadvantages?
Search with Linear Memory Requirement

• Depth-First Search (w/o coloring all expanded states):
  • explore each every possible path one at a time avoiding looping and keeping in the memory only the best path discovered so far
  • Complete and optimal (assuming finite state-spaces)
  • Memory: $O(bm)$, where $b$ – max. branching factor, $m$ – max. pathlength in graph
  • Complexity: $O(b^n)$, since it will repeatedly re-expand states
Search with Linear Memory Requirement

• Depth-First Search (w/o coloring all expanded states):
  • explore each every possible path one at a time avoiding looping and keeping in the memory only the best path discovered so far
  • Complete and optimal (assuming finite state-spaces)
  • Memory: $O(bm)$, where $b$ – max. branching factor, $m$ – max. pathlength in graph
  • Complexity: $O(b^m)$, since it will repeatedly re-expand states
  • Example:
    – graph: a 4-connected grid of 40 by 40 cells, start: center of the grid
    – A* expands up to 800 states, DFS may expand way over $4^{20} > 10^{12}$ states
Search with Linear Memory Requirement

• **Depth-First Search (w/o coloring all expanded states):**
  • explore each every possible path one at a time avoiding looping and keeping in the memory only the best path discovered so far
  
  • Complete and optimal (assuming finite state-spaces)

  • Memory: $O(bm)$, where $b$ – max. branching factor, $m$ – max. pathlength in graph

  • Complexity: $O(b^m)$, since it will repeatedly re-expand states

• Example:
  • graph: a 4-connected grid of 40 by 40 cells, start: center of the grid
  • A* expands up to 800 states, DFS may expand way over $4^{20} > 10^{12}$ states

*What if goal is few steps away in a huge state-space?*
Search with Linear Memory Requirement

- IDA* (Iterative Deepening A*) [Korf, ‘85]:
  1. set $f_{\text{max}} = 1$ (or some other small value)
  2. execute (previously explained) DFS that does not expand states with $f > f_{\text{max}}$
  3. If DFS returns a path to the goal, return it
  4. Otherwise $f_{\text{max}} = f_{\text{max}} + 1$ (or larger increment) and go to step 2
Search with Linear Memory Requirement

• **IDA* (Iterative Deepening A*) [Korf, ‘85]:**
  1. set $f_{max} = 1$ (or some other small value)
  2. execute (previously explained) DFS that does not expand states with $f > f_{max}$
  3. If DFS returns a path to the goal, return it
  4. Otherwise $f_{max} = f_{max} + 1$ (or larger increment) and go to step 2

• Complete and optimal in any state-space (with positive costs)

• Memory: $O(bl)$, where $b$ – max. branching factor, $l$ – length of optimal path

• Complexity: $O(kbl)$, where $k$ is the number of times DFS is called
What You Should Know…

• How to search for a path that is cost-minimal given multiple potential goals with different goal costs (e.g., know how the graph transformation using “imaginary” goal)

• The operation of Iterative Deepening A* (IDA*)

• Pros and cons of IDA* as compared with A*