We solve a UAV small set partitioning problem using the *A*\* algorithm (with a Path Based Evaluation Function – gs-bfs search graph – OR graph) .

Suppose we have four targets that each are required to be covered by exactly one UAV. There is a deployment cost associated with sending a set of UAVs from a unit to cover a set of targets.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Targets | 1 | 2 | 3 | 4 | 12 | 13 | 14 | 23 | 24 | 123 | 124 |
| Cost | 64 | 74 | 72 | 69 | 125 | 110 | 113 | 132 | 138 | 162 | 164 |

Notice that no unit can cover all four targets. There are eleven UAVs to consider. Also, cost/UAV decreases as the number of UAVs increases. We form a modified Tableau from that found in Christofides’ text [2].

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Target 1 block | | | | | | Target 2 block | | | 3 | 4 |
|  | A | B | C | D | E | F | G | H | I | J | K |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
|  | 64 | 125 | 110 | 113 | 162 | 164 | 74 | 132 | 138 | 72 | 69 |

Each set is given a label from A through K, the cost of each set is at the bottom of the tableau, the targets are listed along the left-hand side, and the sets are organized into blocks where block *i* has its first 1 at target *i*. Note that the sets are NOT sorted per block as in the original Tableau.

The entire search graph that corresponds to this problem is shown. The initial state contains no sets. Since a target 1 has to be covered, the sets A-F form the children of the root node. If a node does not correspond to a set partition, then there must be a first target *i* not covered. The children of such a node consist of nodes formed by adding sets from block *i* that do not intersect with already chosen sets. All paths down this search graph terminate in a set partition because all singleton sets are allowed.

We take as *g*(*n*) the sum of the costs of the sets contained in *n*. We need an admissible *h* heuristic. So we need to make sure that *h* is monotonic and optimistic. To realize this, we let *h*(*n*) be the minimum cost set from the next block to be considered. For example, the left-most child of the starting node contains the set A. Since A costs 64, *g*(*n*) = 64. Set A does not cover targets 2, 3, or 4, so the next block to be considered is block 2. The cheapest set in block 2 is G with a cost of 74. Thus, *h*(*n*) = 74. Clearly, *h* is optimistic.

Since the search graph is a tree because of the Tableau, nodes are never reopened once closed, so *h* is monotonic by default. Following the full search graph is a series of figures that shows the progress of the *A\** algorithm. Red nodes are CLOSED and green nodes are OPEN.

Operations, F

* *I*(*n*): *n* is the start node, {} which is initially placed on the OPEN list
* *state:* set of nodes explored (OPEN) and set of nodes expanded (CLOSED)
* *set of candidates*: nodes on the frontier grouped by inclusion in the OPEN list
* *selection function*: choose node n’ with smallest *f*(*n*’), place it on OPEN list and its descendants on the CLOSED list
* *feasibility function*: all nodes are feasible by construction (using the tableau); no PD constraints
* *solution function:* a node is a solution (terminal) if the sets it contains cover the set of targets. Terminal nodes are placed on the CLOSED list.

|  |
| --- |
| OPEN  {} f({}) = 0 |
| CLOSED |

|  |
| --- |
| OPEN  A: f(A) = 64+74 = 138 🡨Best choice  B: f(B) = 125+72 = 197  C: f(C) = 110+74 = 184  D: f(D) = 113+74 = 187  E: f(E) = 162+69 = 231  F: f(F) = 164+72 = 236 |
| CLOSED  {} |

In the table above, *f*(*A*) is calculated by taking the cost of set *A*, 64, for *g*(*A*) and the cost of the cheapest set from block 2, 74, for *h*(*A*). Similar calculations are done for the other OPEN nodes. According to the *f* values, the best choice for expansion is *A*. I’m not including the back pointers for this example because of its simplicity. It is obvious which node is the predecessor.

|  |
| --- |
| OPEN  B: f(B) = 125+72 = 197  C: f(C) = 110+74 = 184 🡨Best choice  D: f(D) = 113+74 = 187  E: f(E) = 162+69 = 231  F: f(F) = 164+72 = 236  AG: f(AG) = 138+72 = 210  AH: f(AH) = 196+69 = 265  AJ: f(AJ) = 202+72 = 274 |
| CLOSED  {}  A |

h Example:

h(A) < c(A,AG) + h(AG)

74 < 138 + 72 “monotonic”

|  |
| --- |
| OPEN  B: f(B) = 125+72 = 197  D: f(D) = 113+74 = 187 🡨Best choice  E: f(E) = 162+69 = 231  F: f(F) = 164+72 = 236  AG: f(AG) = 138+72 = 210  AH: f(AH) = 196+69 = 265  AJ: f(AJ) = 202+72 = 274  CG: f(CG) = 184+69 = 253  CI: f(CI) = 248+0 = 248 🡨Terminal |
| CLOSED  {}  A  C |

We have a situation now where a node is terminal. CI covers all the targets. Its true cost is 248 and it will be moved to the CLOSED list. The next best choice for expansion is D.

|  |
| --- |
| OPEN  B: f(B) = 125+72 = 197 🡨Best choice  E: f(E) = 162+69 = 231  F: f(F) = 164+72 = 236  AG: f(AG) = 138+72 = 210  AH: f(AH) = 196+69 = 265  AJ: f(AJ) = 202+72 = 274  CG: f(CG) = 184+69 = 253  DG: f(DG) = 187+72 = 259  DH: f(DH) = 245+0 = 249 🡨Terminal |
| CLOSED  {}  A  C  CI (248)  D |

h Example:  
 h(C) < c(C,CG) + h(CG)

74 < 184 + 69 “montonic”

“Need to show monotonic for all nodes n and children”

|  |
| --- |
| OPEN  E: f(E) = 162+69 = 231  F: f(F) = 164+72 = 236  AG: f(AG) = 138+72 = 210 🡨Best choice  AH: f(AH) = 196+69 = 265  AJ: f(AJ) = 202+72 = 274  CG: f(CG) = 184+69 = 253  DG: f(DG) = 187+72 = 259  BJ: f(BJ) = 197+69 = 266 |
| CLOSED  {}  A  C  CI (248)  D  DH (249)  B |

|  |
| --- |
| OPEN  E: f(E) = 162+69 = 231 🡨Best choice  F: f(F) = 164+72 = 236  AH: f(AH) = 196+69 = 265  AJ: f(AJ) = 202+72 = 274  CG: f(CG) = 184+69 = 253  DG: f(DG) = 187+72 = 259  BJ: f(BJ) = 197+69 = 266  AGJ: f(AGJ) = 210+69 = 279 |
| CLOSED  {}  A  C  CI (248)  D  DH (249)  B  AG |

|  |
| --- |
| OPEN  F: f(F) = 164+72 = 236 🡨Best choice  AH: f(AH) = 196+69 = 265  AJ: f(AJ) = 202+72 = 274  CG: f(CG) = 184+69 = 253  DG: f(DG) = 187+72 = 259  BJ: f(BJ) = 197+69 = 266  AGJ: f(AGJ) = 210+69 = 279  EK: f(EK) = 231+0 = 231 🡨Terminal |
| CLOSED  {}  A  C  CI (248)  D  DH (249)  B  AG  E |

The value of 231 is the optimal solution because no partial solution has a value equal or below 231 at this point in the search process. Thus, the A\* Algorithm should stop with just enough “delayed termination”. However, we continue to generate all solutions for pedagogical presentation.

|  |
| --- |
| OPEN  AH: f(AH) = 196+69 = 265  AJ: f(AJ) = 202+72 = 274  CG: f(CG) = 184+69 = 253 🡨Best choice  DG: f(DG) = 187+72 = 259  BJ: f(BJ) = 197+69 = 266  AGJ: f(AGJ) = 210+69 = 279  FJ: f(FJ) = 236+0 = 236 🡨Terminal |
| CLOSED  {}  A  C  CI (248)  D  DH (249)  B  AG  E  EK (231)  F |

|  |
| --- |
| OPEN  AH: f(AH) = 196+69 = 265  AJ: f(AJ) = 202+72 = 274  DG: f(DG) = 187+72 = 259 🡨Best choice  BJ: f(BJ) = 197+69 = 266  AGJ: f(AGJ) = 210+69 = 279  CGK: f(CGK) = 253+0 = 253 🡨Terminal |
| CLOSED  {}  A  C  CI (248)  D  DH (249)  B  AG  E  EK (231)  F  FJ (236)  CG |

|  |
| --- |
| OPEN  AH: f(AH) = 196+69 = 265 🡨Best choice  AJ: f(AJ) = 202+72 = 274  BJ: f(BJ) = 197+69 = 266  AGJ: f(AGJ) = 210+69 = 279  DGJ: f(DGJ) = 259+0 = 259 🡨Terminal |
| CLOSED  {}  A  C  CI (248)  D  DH (249)  B  AG  E  EK (231)  F  FJ (236)  CG  CGK (253)  DG |

|  |
| --- |
| OPEN  AJ: f(AJ) = 202+72 = 274  BJ: f(BJ) = 197+69 = 266 🡨Best choice  AGJ: f(AGJ) = 210+69 = 279  AHK: f(AHK) = 265+0 = 265 🡨Terminal |
| CLOSED  {}  A  C  CI (248)  D  DH (249)  B  AG  E  EK (231)  F  FJ (236)  CG  CGK (253)  DG  DGJ (259)  AH |

|  |
| --- |
| OPEN  AJ: f(AJ) = 202+72 = 274 🡨Best choice  AGJ: f(AGJ) = 210+69 = 279  BJK: f(BJK) = 266+0 = 266 🡨Terminal |
| CLOSED  {}  A  C  CI (248)  D  DH (249)  B  AG  E  EK (231)  F  FJ (236)  CG  CGK (253)  DG  DGJ (259)  AH  AHK (265)  BJ |

|  |
| --- |
| OPEN  AGJ: f(AGJ) = 210+69 = 279 🡨Best choice  AIJ: f(AIJ) = 274+0 = 274 🡨Terminal |
| CLOSED  {}  A  C  CI (248)  D  DH (249)  B  AG  E  EK (231)  F  FJ (236)  CG  CGK (253)  DG  DGJ (259)  AH  AHK (265)  BJ  BJK (266)  AI |

|  |
| --- |
| OPEN  AGJK: f(AGJK) = 279+0 = 279 🡨Terminal |
| CLOSED  {}  A  C  CI (248)  D  DH (249)  B  AG  E  EK (231)  F  FJ (236)  CG  CGK (253)  DG  DGJ (259)  AH  AHK (265)  BJ  BJK (266)  AI  AIJ (274)  AGJ |

At this point, we no longer have any OPEN nodes to explore. So our version of the A\* Algorithm terminates with everything on the CLOSED list. Observing the terminal nodes on the CLOSED list, choosing E and K minimizes cost at 231. Of course, we have “delayed termination” until all the possibilities were generated (enumerated). This is not required in A\* since when we find the cost at 231, no other nodes need be expanded on OPEN as discussed previously!

Questions:

1. Thoughts if this were a Set Covering Problem?
2. What if we use the format of Christofides Tableau exactly?
3. What is f1 in this A\* algorithm?

The specific evolution of the OPEN and CLOSED lists and the A\* structure are indicated explicitly as related to the six basic search elements:

**The A\* Algorithm (Pearl)**

// Initialize variables and sets– OPEN, CLOSED, *g(n), h(n), f(n,),* s

1. Put the start node ***s*** on OPEN
2. If OPEN is empty, exit with failure.

// Selection Function

1. Remove from OPEN and place on CLOSED, a node ***n*** for which ***f*** is minimum (or max for merit). One would add feasibility functions from problem domain (PD) constraints here (creative heuristics).

// Solution Function

1. If ***n*** is a goal node, exit successfully with the optimal solution obtained by tracing back the pointers from ***n*** to ***s***.

// Next State Generator

1. Otherwise expand ***n***, generating **all** its successors, and attach to them pointers back to ***n***. For every successor ***n’*** of ***n***:

// Evaluation Functions *g(n)* and *f(n)* and associated path update in this Graph Search -> tree search

* 1. If ***n’*** is not already on OPEN or CLOSED, estimate ***h(n’)***(an estimate of the cost of the best path from ***n’*** to some goal node), and calculate ***f(n’)=g(n’)+h(n’)*** where ***g(n’)=g(n)+c(n, n’)*** and ***g(s)=0***.
  2. If ***n’*** is already on OPEN or CLOSED, direct its pointers along the path yielding the lowest ***g(n’)***.
  3. If ***n’*** required pointer adjustment and was found on CLOSED, reopen it (put ***n’*** back on OPEN).

1. Go to step 2.

NOTE: This is the A\* algorithm from Judea Pearl [1]

**Questions:** (relate additional operations as part of its underlying generic search element)

1. How would you modify Pearl’s A\* algorithm to indicate an iterative deepening (**IDA**) approach?
2. How would you modify Pearl’s A\* algorithm to indicate a **beam search** approach?
3. In both of these cases how would explicitly define the associated heuristic parameter values for a given NPC problem domain?
4. How would you evolve A\* pseudo code for a given NPC problem using our standard design technique? (of course, we have done this for dfs (Greedy), gs\_dfs, Tabu, and SA already!)
5. Do explicit pointers need to be shown in the algorithm? Can one maintain a different data structure represent all the characteristics of DP at each node?
6. How can we make A\* a generic breath-first search strategy? Un-informed vs. informed A\* Search? How about a uniform cost strategy? (see page 65 in Pearl[1])
7. What about an A\* depth-first strategy? (see page 65 in Pearl[1])
8. What about utility of other A\* variants? (see Wikipedia – A\* search algorithm)
9. How does step 5 change if one requires a Z\* algorithm?
10. How is “delayed termination” incorporated in Pearl’s A\* algorithm?
11. Can the CLOSED set be omitted (yielding a tree search algorithm) if a solution is guaranteed to exist? Computational Impact?
12. Can the A\* algorithm be adapted so that new nodes are added to OPEN only if they have a lower f value than at any previous iteration? Computational impact?

**\*\*\*\*\*\*\*\*\***

“One can map the generic A\* Algorithm to a more definitive pseudocode closer to a selected programming language implementation”

**Pseudocode (Wikipedia[3])** “A number of Java and C implementations also are available online”

The following more detailed pseudocode describes the A\* Algorithm:  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
**function A\***(start,goal)

// Initialize variables and sets – OPEN, CLOSED, *g(n), h(n), f(n)*

closedset := the empty set // The set of nodes already evaluated.

openset := set containing the initial node // The set of tentative nodes to be evaluated.

came\_from := the empty map // The map of navigated nodes.

g\_score[start] := 0 // Cost from start along best known path.

h\_score[start] := heuristic\_cost\_estimate(start, goal)

f\_score[start] := h\_score[start] // Estimated total cost from start to goal through y.

while openset is not empty **//Section Function**

x := the node in openset having the lowest f\_score[] value

if x = goal \\Solution Function

return reconstruct\_path(came\_from, came\_from[goal])\\[Delayed](file:///\\delay) Termination

remove x from openset

add x to closedset

foreach y in neighbor\_nodes(x) [\\Set](file:///\\Set) of Candidates

if y in closedset

continue

tentative\_g\_score := g\_score[x] + dist\_between(x,y) [\\Evaluation](file:///\\Evaluation) g(n)

if y not in openset

add y to openset

tentative\_is\_better := true

else if tentative\_g\_score < g\_score[y]

tentative\_is\_better := true

else

tentative\_is\_better := false

if tentative\_is\_better = true

came\_from[y] := x

g\_score[y] := tentative\_g\_score

h\_score[y] := heuristic\_cost\_estimate(y, goal)

f\_score[y] := g\_score[y] + h\_score[y] [\\Evaluation](file:///\\Evaluation) f(n)

return failure

**function reconstruct\_path**(came\_from, current\_node)

if came\_from[current\_node] is set

p = reconstruct\_path(came\_from, came\_from[current\_node])

return (p + current\_node)

else

return current\_node

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\*\*\*\*\*\*\***

**A\* Code: (**Note the different data structures and A\* Algorithm variations)

1. <http://www.cokeandcode.com/info/showsrc/showsrc.php?src=../../pathfinder/PathFindingTutorial/target/src/org/newdawn/slick/util/pathfinding/AStarPathFinder.java>
2. <http://memoization.com/2008/11/30/a-star-algorithm-in-java/>
3. [http://code.activestate.com/recipes/577457-a-star-shortest-path-algorithm/](http://code.activestate.com/recipes/577457-a-star-shortest-path-algorithm/http://code.activestate.com/recipes/577457-a-star-shortest-path-algorithm/)

**References**: (incomplete)

[1] Pearl, J. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison Wesley (1984).

[2] Christofides, N. *Graph Theory: An Algorithmic Approach*. Academic Press (1975).

[3] A\* Search Algorithm, Wikipedia

**Demos:** (incomplete)

1. <http://www.policyalmanac.org/games/aStarTutorial.htm>
2. [http://thoughtsfrommylife.com/article-720-A\*\_Pathfinding\_Algorithm\_Demo\_Applet](http://thoughtsfrommylife.com/article-720-A*_Pathfinding_Algorithm_Demo_Applet)
3. <http://www.vision.ee.ethz.ch/~cvcourse/astar/AStar.html> “may not work”
4. <http://www.briangrinstead.com/blog/astar-search-algorithm-in-javascript>
5. <http://www.hdinteraction.nl/blog/2009/02/astar-pathfinding-demo/>