The Eleventh International Symposium on Combinatorial Search
(SoCS 2018)

Position Paper: Using Early Goal Test in A*

Ariel Felner
Software and Information Systems Engineering
Ben-Gurion University
Be'er-Sheva, Israel 85104
felner@bgu.ac.il

Abstract

This paper focuses on the stopping condition of A*. Traditionally, A* is described such that the goal test is done once a node is chosen for expansion (A*-LATE). An alternative way is to perform the goal test when a node is generated (A*-EARLY). In this position paper we compare the two approaches from pedagogical and practical aspects and advocate for teaching and using A*-EARLY.

1 Introduction

The focus of this paper is A* (Hart, Nilsson, and Raphael 1968) which is a prominent instantiation of the general *best-first search* (denoted BFS) scheme. BFS seeds an *open list* (denoted OPEN) with the start state. At each *expansion cycle* the most promising node (*best*) from OPEN is popped and its children are generated and inserted to OPEN while *best* is moved to a *closed list* (denoted CLOSED). Instantiations of BFS differ in their cost function f. A* orders nodes according to $f(n) = g(n) + h(n)$, where $g(n)$ is the path cost from the start to n and $h(n)$ is an admissible heuristic estimation of the cost from n to a goal. A* with an admissible heuristic returns the optimal path and is optimally effective (expands the necessary and sufficient set of nodes) under some conditions (Dechter and Pearl 1985).

In order to halt, any search algorithm must apply a *goal test* to recognize that a goal node is found. A goal test is done either by checking whether a certain goal condition is satisfied or by comparing the node to the description of the goal. This paper focuses on two approaches for when to apply the goal test within the context of A*. The relatively common approach delays the goal test to the step where a node is chosen for expansion. Alternatively, the goal test can be performed earlier when a node is generated. We compare these two approaches pedagogically and practically. While there are pros and and cons for each approach, we advocate for teaching and using the early goal test approach which performs the goal test when a node is generated. Even readers who will still prefer the late goal test approach should hopefully benefit from reading this paper.

Algorithm 1: A*-LATE

```
Input: (Start state $s$)
1  $g(s) \leftarrow 0$; OPEN $\leftarrow \emptyset$; CLOSED $\leftarrow \emptyset$
2  Add $s$ to OPEN with $f(s) = h(s)$
3  while (OPEN $\neq \emptyset$) do
4      $best \leftarrow$ ExtractMin(OPEN)
5      if GoalTest($best$)==TRUE then
6          return the lowest-cost path found to $best$
7      Move $best$ from OPEN to CLOSED
8      for every action $A$ applicable on state $best$ do
9          $c \leftarrow$ generate a state by applying $A$ to $best$
10         $g_{new} \leftarrow g(best) + cost(best, c)$
11         if $c$ in OPEN $\cup$ CLOSED then
12             if $g(c) \leq g_{new}$ then
13                 continue (duplicate node, goto line 9)
14             Remove $c$ from OPEN and CLOSED
15         $g(c) \leftarrow g_{new}$
16         Insert $c$ to OPEN with key $f(c) = g(c) + h(c)$
17 return No solution exists
```

2 The Two Variants of A*

2.1 A* with Late Goal Test

In general, the "textbook" definition of A* performs the goal test when a node is chosen for expansion. We call this *late goal test* and denote it by A*-LATE. A*-LATE halts if the node chosen for expansion is the goal. At this stage we have reached the goal via an optimal solution (whose cost is denoted by C^*) due to the following reasoning (the formal proof is much longer): $f(goal) = g(goal)$ is the minimal f-value in OPEN. Thus, the cost of the current path to the goal, $g(goal)$, is *smaller than or equal to* all other possible paths to the goal as they all have lower bounds in OPEN which are $\geq g(goal)$. Pseudo code for A*-LATE is provided in Algorithm 1. The goal test is performed at Line 5, after choosing to expand a node. It is important to note that A*-LATE does not need to perform a goal test for nodes with $h > 0$, as such nodes are certainly not the goal.

2.2 A* with Early Goal Test

Another, less common implementation of A* is to apply the goal test when a node is generated. We call this *early goal*

Algorithm 2: A*-EARLY

```
Input: (Start state $s$)
1  $U \leftarrow \infty$
2  $g(s) \leftarrow 0$;
3  OPEN $\leftarrow \emptyset$;
4  CLOSED $\leftarrow \emptyset$
5  Add $s$ to OPEN with $f(s) = h(s)$
6  while (OPEN $\neq \emptyset$ and $fmin < U$) do
7      $best \leftarrow$ ExtractMin(OPEN)
8      Move $best$ from OPEN to CLOSED
9      for every action $A$ applicable on state $best$ do
10         $c \leftarrow$ generate a state by applying $A$ to $best$
11         $g_{new} \leftarrow g(best) + cost(best, c)$
12         if $c$ in OPEN $\cup$ CLOSED then
13             if $g(c) \leq g_{new}$ then
14                 continue (duplicate node, goto line 9)
15             Remove $c$ from OPEN and CLOSED
16         if GoalTest(c)==TRUE then
17             if $g_{new} < U$ then
18                 $U \leftarrow g_{new}$
19                 empty-open($U$) // optional.
20         $g(c) \leftarrow g_{new}$
21         $f(c) = g(c) + h(c)$
22         if $f(c) < U$ then
23             Insert $c$ to OPEN with key $f(c)$
24 return $U$ and the associated path
```

test and denote it by A*-EARLY. In this variant, we maintain a variable U which stores the cost of the best solution found so far — the *incumbent solution*. U is initially set to ∞ and is decreased every time a better solution is found for a newly generated goal. A*-EARLY halts once there are no nodes in OPEN with $f < U$, or equivalently when $f_{min} >= U$, where f_{min} is the minimal f-value in OPEN. In this case, $U = C^*$ is returned as the cost of the optimal solution.

Pseudo code for A*-EARLY is provided in Algorithm 2. The main loop ensures that there are nodes in OPEN with $f < U$ (Line 6). The goal test is performed when a node is generated (Line 16) and U is updated if necessary (Line 18). It is easy to reason that A*-EARLY also returns the cost of the optimal solution. When OPEN (which is always a perimeter around the start state) does not contain any node with $f < U$ it means that U is optimal.

An enhancement for A*-EARLY is that newly generated nodes with $f \geq U$ can be discarded and not be inserted to OPEN (Line 22). These nodes will never be expanded and we designate them here as *surplus*[1]. This enhancement may save a significant amount of CPU time compared to A*-LATE for not applying the *insert* operation on OPEN (Line 23) for surplus nodes. This will also save a significant amount of memory as such nodes will not be stored in OPEN. Furthermore, when U is decreased when a better path to the goal has been found, OPEN can optionally be cleared from nodes with $f \geq U$ (Line 19). This might further save a considerable amount of memory but might cost time.

[1]This is a slightly different usage of this term which was coined by (Goldenberg et al. 2014) to designate all nodes with $f > C^*$.

A*-EARLY only needs to perform a goal test for nodes with $h = 0$. Equivalently, a heuristic calculation should only be done for non-goal nodes. Thus, A*-EARLY has the flexibility to choose which of these operations to perform first.

2.3 Tie Breaking

A common tie breaking rule used by A* among nodes with the same f-value is to prefer nodes with smaller h-values. This rule is denoted by TB_h. Another tie breaking rule used by A* is to break ties in favor of the goal (denoted by TB_{Goal}). TB_{Goal} is included by TB_h if only the goal may have $h = 0$. This might be a direct attribute of the heuristic especially if zero edges are not allowed. An example is the 15-Puzzle with Manhattan Distance. In other settings, where non-goal nodes may have $h = 0$, or if TB_h is not used, TB_{Goal} should be implemented by performing the goal test on two or more nodes with the same g-values and with $h = 0$. TB_{Goal} is specifically needed for A*-LATE. Assume that the goal was generated via the optimal path (i.e., with $f(goal) = C^*$). Once $f_{min} = C^*$, TB_{Goal} assures that the goal will immediately be moved to the front of OPEN and expanded right away. Nevertheless, omitting TB_{Goal} will only increase the running time but will not affect the optimality of the solution returned. We note that more advanced tie breaking rules exist (Asai and Fukunaga 2017), especially if 0-edges are allowed.

3 Comparison of the Two Variants

In this section we compare A*-LATE and A*-EARLY pedagogically and practically. There are pros and cons for both but we will advocate for teaching and using A*-EARLY as we believe that its advantages outweigh its disadvantages.

3.1 Order of Node Expansion

First, we show that A*-LATE coupled with TB_{Goal} and A*-EARLY are identical in their node expansions.

Claim 1: Both A*-LATE (with TB_{Goal}) and A*-EARLY expand the same set of nodes and in the same order.

Proof: Their node expansion behavior is identical when $f_{min} < C^*$. Similarly, when f_{min} becomes C^*, then their node expansion behavior is identical if the goal node was not yet generated with $f = C^*$. When the optimal goal was generated with $f = C^*$ all that is needed is to clear OPEN from all nodes with $f < C^*$. Indeed, both A*-LATE and A*-EARLY next expand all nodes with $f < C^*$. At this stage A*-EARLY immediately halts. By contrast, A*-LATE inserted this goal node to OPEN. However, using TB_{Goal} this goal node will immediately move to the front of OPEN and be chosen for expansion right away.[2] □

[2]There are two options to count node expansions. If we increment the counter when a node is chosen for expansion, A*-LATE will count one more node than A*-EARLY — the goal node. If incrementing the counter is delayed to when the children of the node are being generated then A*-LATE and A*-EARLY are identical in their node expansions. This was done in our experiments below.

This means that both A*-EARLY and A*-LATE are implementation variants of the same algorithm and are not fundamentally different. However, in the rest of this paper we highlight the advantages of A*-EARLY over A*-LATE.

3.2 Using TB_h and TB_{Goal}

Both A*-LATE and A*-EARLY may optionally use TB_h and the are both equally affected positively when they use it. However, A*-LATE has a disadvantage because it *must* use TB_{Goal}. Without TB_{Goal}, A*-LATE might expand a significant number of nodes with $f = C^*$, even if the optimal goal was generated with $f = C^*$, as it was not necessarily chosen for expansion. By contrast, A*-EARLY does not need to employ TB_{Goal} because it will never expand nodes with $f = C^*$ after the goal node is generated with $g = C^*$. This is a great advantage of A*-EARLY in the pedagogical sense — no need to teach and implement TB_{Goal}. Furthermore, as discussed above, in some settings non-goal nodes may also have $h = 0$. In such cases, A*-LATE must perform some kind of early goal test when it needs to employ TB_{Goal} on a generated node with $h = 0$. In these cases, A*-LATE does not employ a pure *late goal test* strategy. Adding TB_{Goal} complicates the implementation of A*-LATE.

3.3 Pedagogical Aspects

We believe that A*-EARLY has pedagogical advantages over A*-LATE. There are many optimization problems in computer science where a solution with optimal cost/utility is needed. Typical algorithms that find optimal solutions go through the following four phases:

Phase 1: No solution yet. The algorithm did not yet find any solution. The time was used in building partial solutions or checking possibilities that did not yield a solution.

Phase 2: Suboptimal solution found. This phase starts after the first, possibly suboptimal, solution was found. The algorithm keeps looking for better solutions. But, it has the advantage over Phase 1 in the sense that there exists an incumbent solution. Thus, it can prune/discard options that certainly cannot improve the incumbent solution. It can also return a solution if it is halted.

Phase 3: An optimal solution was found. Now the algorithm has found the optimal solution but it has not yet proven it. In this phase the algorithm needs to prove that the solution it found is optimal. In practice, the algorithm may not know whether it is in Phase 2 or in Phase 3.

Phase 4: The optimal solution was proved. In this phase the algorithm can halt and return the optimal solution.

Algorithms that have this structure have benefits because they have an anytime behavior. They can be halted and return a solution in any of the phases 2 – 4. Naturally, these solutions improve over time. In Phase 2, a suboptimal solution will be returned. In Phase 3, practically an optimal solution will be returned but without a guarantee on its optimality. In Phase 4, an optimal solution is returned and is guaranteed.

A*-EARLY falls into this general structure. When the first solution was found, A*-EARLY enters Phase 2. From now on, it can return a solution and (optionally) prune surplus nodes (with $f \geq U$). During Phase 3, A*-EARLY will practically return the optimal solution. Indeed, the user will not get a guarantee on optimally but will have it in his hands. This is a great advantage because phase 3 might be very long.

By contrast, A*-LATE only has two phases. In the first phase, no solution was recognized. In the second phase, the optimal solution was found and returned. A*-LATE will not be able to return a solution if it is halted before finishing the search. On the other hand, the pseudo code of A*-LATE is shorter and might be seen as more elegant, exactly because it only has two phases. Thus, some people might find it easier to teach or understand. Such people probably omit, or are not aware of the TB_{Goal} issue raised above.

3.4 Memory Usage

A*-EARLY has a straightforward advantage in its memory usage over A*-LATE. That is, once the goal has been found with $f = U$, all new surplus nodes with $f \geq U$ can be discarded instead of being inserted to OPEN. One might also iterate over OPEN and throw away such nodes too, freeing memory. By contrast, A*-LATE will add all these surplus nodes to OPEN which will increase the size of OPEN.

3.5 Time Overhead

A*-EARLY has two advantages over A*-LATE with regards to the time overhead per node. First, A*-EARLY performs fewer *insert* operations than A*-LATE as it does not insert the surplus nodes. If OPEN is implemented as a priority queue then *insert* might be relatively time consuming as it is logarithmic in the size of OPEN. Second, in many cases smaller OPEN has direct impact on the CPU overhead both because *insert* is logarithmic in the size of OPEN but also because of the memory access time which might be faster if OPEN is smaller, e.g., if it can be stored in cache etc.

By contrast, the tradeoff is that A*-LATE may perform fewer goal tests than A*-EARLY. If the goal-test operation is very expensive and inserting nodes to OPEN is cheap (both in time and memory) then A*-LATE has benefits over A*-EARLY. Nevertheless, the goal test is usually easy to implement. Also, as described above, it is not needed for nodes with $h > 0$ (for both A*-EARLY and A*-LATE). Thus, its time overhead is usually not a dominating factor.

3.6 Breadth-First Search

Breadth-first search (denoted here by BRFS) is a degenerate form of best-first search (BFS). On unit edge-cost domains, at all times BRFS has at most two distinct f-values in OPEN: $f = k$ and $f = k + 1$ for some integer k. In addition, when a node with $f = k$ is expanded, all its children have $f = k + 1$. Thus, BRFS might use a FIFO queue for OPEN instead of a priority queue. Furthermore, it is very easy to recognize, even by novice implementers, that BRFS can halt when a goal node is generated. That is, an efficient implementation of BRFS (probably recognized and employed by most implementers) performs early goal test and halts right away. But, if A*-LATE is taught to students then the instructor needs to teach the students that performing early goal test is a nice enhancement over A* that exists for BRFS. Some instructors might not even mention it and leave it to the good implementation capabilities of their students. By contrast, if A*-EARLY is taught, then there is absolutely no difference between BRFS and BFS in that regards.

In fact, the common AI textbook : "Artificial Intelligence, a Modern Approach", 3rd Edition (Russell and Norvig 2010) has this pitfall. It introduces two pseudo codes for general BFS: *Tree-Search* for trees (no cycles) and Graph-Search for graphs (cycles exist). In these pseudo codes, late goal test is implemented. Then, a separate pseudo code with early goal test is given for BRFS. If A*-EARLY is taught then no pseudo code will be even needed for BRFS.

3.7 Other Algorithms with Early Goal Test

Where to perform the goal test might be seen as an implementation detail. However, many search algorithms can greatly benefit from using early goal test. Indeed, some papers on search algorithms specificality mention that they used early goal test either as an enhancement trick or as an implementation detail. We cover such algorithms now.

First, as explained above, BRFS should exclusively use early goal test. In addition, IDA* can also employ early goal test. Once a goal node is generated with $f = U$, IDA* can halt as soon as all IDA* iterations with thresholds smaller than U are finished.

Potential Search (PTS) (Stern et al. 2014) which is an algorithm for *bounded cost search* inherently uses early goal test. Similarly, adapting A* to maximization problems inherently uses early goal test in its structure (Stern et al. 2015).

Early goal test in a bidirectional search checks whether a generated node on one side of the search also appears in OPEN of the other side. Using early goal test in bidirectional search algorithms is specifically beneficial because it allows them to prune faster pairs of nodes that will certainly not improve the incumbent solution. Indeed, many of the early bidirectional search algorithms (see (Holte et al. 2017) for a survey) and certainly all the recent ones (Barker and Korf 2015; Holte et al. 2017; Chen et al. 2017; Shaham et al. 2017) use early goal test.

Focal Search is a general scheme for search algorithms. Focal searches maintain a list of nodes FOCAL $\subseteq$ OPEN meeting special properties. Most Focal Search algorithms aim to find a goal node within FOCAL.[3] Certainly, employing early goal test in Focal Search has a great advantage over late goal test. Focal Search can halt as soon a a goal node is generated and is within FOCAL. While some papers in the Focal Search family mention early goal test in their descriptions, e.g., (Thayer and Ruml 2011; Hansen and Zhou 2007), others leave it vague (Likhachev et al. 2008) or describe late goal test (Gilon, Felner, and Stern 2016; Thayer and Ruml 2008; Ebendt and Drechsler 2009), even though they might have implemented early goal test in the experiments without specifically writing it.

[3]A* can be described as a Focal Search where FOCAL includes all node with $f = f_{min}$.

	Expanded	Generated	Surplus	Inserted
15 puzzle	1,910,009	4,108,662	2,198,653	1,275,218
10 x 10	43	58	15	2
50 x 50	1215	1299	84	3
100 x 100	4785	4956	171	6

Table 1: Experiments on weighted graphs

3.8 Cases Where They are Identical

In some cases there will be no practical difference between A*-LATE and A*-EARLY. In such cases, when the goal is generated for the first time it is through the optimal path (with $f(goal) = C^*$) directly from a parent which also has $f = C^*$ (cases with this last attribute are called *pathological* (Dechter and Pearl 1985; Goldenberg et al. 2013)). In such cases, both A*-LATE and A*-EARLY will halt right after the generation of the first goal because at this time $f_{min}= C^*$ and surplus nodes will never be generated.

Many of the unit edge-cost domains have this behavior. For example, consider the 15-puzzle with the Manhattan Distance heuristic where the goal of the blank is at the top left corner. The last move before reaching the goal is to either move tile 1 from the top left corner to the right or move tile 4 from that corner down. In both cases, the Manhattan Distance of the parent is 1 and therefore the parent will also have $f = C^*$. Nevertheless, there are no circumstances where A*-LATE will have practical advantages over A*-EARLY.

4 Experiments

To show the practical benefits of A*-EARLY we performed a small number of experiments with A*-LATE and A*-EARLY on two domains: (1) The *heavy 15-puzzle* (Thayer and Ruml 2011; Gilon, Felner, and Stern 2016) which is a weighted graph where moving tile $\#X$ costs X. (2) Square grids where the cost of an edge was uniformly randomized to be an integer between 1 and 10. The unweighed Manhattan Distance heuristic was used for both domains. Table 1 shows the results averaged over 50 random instances for each domain. We present the number of expanded nodes, the number of generated nodes and the number of surplus nodes which is calculated by $Surplus = Generated - Expanded$. A*-LATE inserts all the generated nodes including *all* surplus nodes into OPEN. A*-EARLY only inserts some of the surplus nodes and skips others. We also report the number of surplus nodes that A*-EARLY inserted into OPEN in the *Inserted* column. The number of states in the 15 puzzle grow exponentially with the depth and the number of surplus nodes was relatively large. Of them, only 58% were inserted by A*-EARLY and the other were skipped. The grids grow linearly and the number of surplus nodes is relatively smaller. However, the saving of A*-EARLY was more dramatic here as it never inserted more than 10% of the surplus nodes into OPEN. The decrease in CPU time was of up to 10% for some instances which is naturally more modest.

5 Conclusions

In this paper we highlighted the advantages of A*-EARLY over A*-LATE. Its pedagogical structure is more general and it may have practical advantages in both memory and time for many cases. Under no circumstances A*-EARLY will be worse than A*-LATE. By contrast, A*-LATE might be seen as simpler and more elegant by some people (although it must use the TB_{Goal} rule). In addition, there are many cases (e.g., in many unit edge cost domains) where A*-LATE and A*-EARLY are identical. Which variant to teach and to implement is left for each one to choose. But, we encourage instructors and implementers to only use A*-EARLY as the benefits outweigh the disadvantages in our opinion.

6 Acknowledgements

The work was supported by Israel Science Foundation (ISF) under grant #844/17. Thanks to Yoni Cohen, Noah Shogan, Etai Gal and Yuval Goldeman for performing the experiments.

References

Asai, M., and Fukunaga, A. 2017. Tie-breaking strategies for cost-optimal best first search. *J. Artif. Intell. Res.* 58:67–121.

Barker, J. K., and Korf, R. E. 2015. Limitations of front-to-end bidirectional heuristic search. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, 1086–1092.

Chen, J.; Holte, R. C.; Zilles, S.; and Sturtevant, N. R. 2017. Front-to-end bidirectional heuristic search with near-optimal node expansions. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 489–495.

Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM* 32(3):505–536.

Ebendt, R., and Drechsler, R. 2009. Weighted A* search – unifying view and application. *Artif. Intell.* 173(14):1310–1342.

Gilon, D.; Felner, A.; and Stern, R. 2016. Dynamic potential search - A new bounded suboptimal search. In *Proceedings of the Ninth Annual Symposium on Combinatorial Search, SOCS 2016, Tarrytown, NY, USA, July 6-8, 2016.*, 36–44.

Goldenberg, M.; Felner, A.; Sturtevant, N. R.; Holte, R. C.; and Schaeffer, J. 2013. Optimal-generation variants of EPEA. In *Proceedings of the Sixth Annual Symposium on Combinatorial Search, SOCS 2013, Leavenworth, Washington, USA, July 11-13, 2013.*

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N. R.; Holte, R. C.; and Schaeffer, J. 2014. Enhanced partial expansion A. *J. Artif. Intell. Res. (JAIR)* 50:141–187.

Hansen, E. A., and Zhou, R. 2007. Anytime heuristic search. *J. Artif. Intell. Res.* 28:267–297.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* SCC-4(2):100–107.

Holte, R. C.; Felner, A.; Sharon, G.; Sturtevant, N. R.; and Chen, J. 2017. MM: A bidirectional search algorithm that is guaranteed to meet in the middle. *Artif. Intell.* 252:232–266.

Likhachev, M.; Ferguson, D.; Gordon, G. J.; Stentz, A.; and Thrun, S. 2008. Anytime search in dynamic graphs. *Artif. Intell.* 172(14):1613–1643.

Russell, S., and Norvig, P. 2010. *Artificial Intelligence, A Modern Approach, Third Edition*. Pearson.

Shaham, E.; Felner, A.; Chen, J.; and Sturtevant, N. R. 2017. The minimal set of states that must be expanded in a front-to-end bidirectional search. In *SoCS*, 82–90.

Stern, R.; Felner, A.; van den Berg, J.; Puzis, R.; Shah, R.; and Goldberg, K. 2014. Potential-based bounded-cost search and anytime non-parametric A*. *Artif. Intell.* 214:1–25.

Stern, R.; Kiesel, S.; Puzis, R.; Felner, A.; and Ruml, W. 2015. Max is more than min: Solving maximization problems with heuristic search. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 4324–4330.

Thayer, J. T., and Ruml, W. 2008. Faster than weighted A*: An optimistic approach to bounded suboptimal search. In *ICAPS*.

Thayer, J. T., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*.