ARA*+: Improved path planning algorithm based on ARA*

Bo Li, Jianwei Gong, and Yan Jiang, Hany Nasry, Guangming Xiong
Intelligent Vehicle Research Center
Beijing Institute of Technology
Beijing, China
gongjianwei@bit.edu.cn

***Abstract*—A* path planning algorithm cannot always guarantee the continuity of a robot's movements when the allocated time is limited, however Anytime Repairing A*(ARA*) can get a sub-optimal solution quickly, and then work on improving the solution until the allocated time expires. This paper proposes a variation of ARA* algorithm (ARA*+) which executes multiple Weighted A* to search the solution. During the first search of ARA*+, Weighted A* with a larger inflation factor is applied and no state is expanded more than once, in this way, the time needed for finding a sub-optimal solution can be remarkably shortened. Then, Weighted A* will be executed again for better path, by decreasing the inflation factor and reusing the previous planning efforts. Here, with the same inflation factor the expanded states can be used again, and this is different from ARA*, which forbids the expanded states to be expanded again. If the allocated time does not expire, this process will not stop until the optimal solution is found, or the current sub-optimal solution will be regarded as the output. According to our robot path planning experiments, in most cases the number of expanded states in ARA*+ is smaller than that in ARA*, as a result, the time required to get the optimal solution will be shorter.**

Keywords-Path planning; Robot; ARA*; ARA*+;A*

I. INTRODUCTION

Path planning of a robot is to calculate a collision-free path from the start state to the goal state, being satisfied with some given environmental constraints and the robot's own mobility constraints, and a certain performance requirements as well [1]. The performance requirements are used to evaluate the optimality of the solution, for instance, the shortest path will be considered as the optimal solution if the cost of the path is the evaluation indicator.

Two widely used path planning methods are sampling-based algorithm [2] and heuristic search algorithm. The sampling-based algorithm shows a very high efficiency in high dimensional space, but it does not have any optimality guarantees. A recent variation known as improved randomly exploring randomized tree (RRT*) was developed that adds an asymptotic optimality guarantee without changing the computational overhead [3]. On the other hand, A* [4], a typical heuristic search algorithm, can significantly reduce the number of expanding states and ensure the optimality of solutions as well. However, these advantages will be weakened when the search area is too large, or the robot moves at a higher speed.

A* can search the optimal solution if the value of heuristic function from current state to the goal state is less than the cost of the optimal path. Based on this, we can come to a conclusion: the larger the value of heuristic function is, the less the number of expanding states will be. As a result, the searching efficiency will become higher. If the value of heuristic function is larger than the cost of the optimal solution, a feasible path can be searched out quickly, though the optimality of the solution might not be ensured.

As an improved A*-based algorithm, Weighted A* [5], which heuristic function, $h(s)$, is multiplied by an inflation factor ε ($\varepsilon \geq 1$), can reduce the searching time and expand less states. Although the optimality of the path has been sacrificed, a sub-optimal path can be searched out quickly, whose cost is no larger than ε times that of the optimal path [6] [7], i.e., Weighted A* can search out a sub-optimal path quickly. In this way, Weighted A* can solve the problem of planning failure in limited time. If there is more time left, Weighted A* will decrease ε and search again for a better solution until the allocated time expires. However, this method does not reuse the previous planning efforts and its efficiency is not very high.

Anytime Weighted A* (AWA*) [8] executes Weighted A* at the beginning of each iteration and quickly comes to a sub-optimal solution. If there is more time left, it will keep improving the result until the optimal solution is obtained or take the current path as the output when the allocated time expires. Although previous planning efforts have been reused, it will expand too many states. Moreover, it will keep expanding useless states even after the optimal solution has been searched out.

Anytime Repairing A* (ARA*) [9] executes multiple Weighted A*, and can effectively reduce the number of states to be expanded by reusing the previous planning efforts. ARA* restricts the re-expansion of states when executing Weighted A* with the current ε, as a result, the current Weighted A* can search for a feasible solution a bit quickly. But more states will be expanded during another search with a lower ε. Hansen [10] has queried the limitation on expanding times of ARA*. He pointed out that this method was flawed because it might expand more states during the next search. Our experimental tests also support Hansen's argument.

Considering the problems above, we proposed a variation of ARA* algorithm (ARA*+). The first search of this algorithm is the same as ARA* that executes Weighted A* with given ε_0 to obtain a sub-optimal path quickly, here no state is expanded more than once. Obviously the cost of

this sub-optimal path dose not exceed ε_0 times of that of the optimal path. If there is more time left, Weighted A* will be executed again with a lower ε and reuse the previous planning efforts. Unlike ARA*, states re-expansion can be allowed under a certain ε in ARA*+.

II. ARA* (ANYTIME REPAIRING A*)

In heuristic search algorithm, A* has defined an evaluation function $f(s)$:

$$f(s) = g(s) + h(s) \tag{1}$$

where s refers to a certain state and $f(s)$ is the estimated cost from the start state to the goal state; $g(s)$ is the cost from the start state to the current state; $h(s)$ is the estimated cost from the current state to the goal state. A* can finally search out the optimal solution under the following condition:

$$h(s) \le cost^*(s, s_{goal}) \tag{2}$$

where $cost^*(s, s_{goal})$ is the cost of the optimal solution from the current state to the goal state. Once A* meets (2), no state will be expanded more than once. And the larger the $h(s)$ is, the less states will be expanded when the optimal solution is searched out. In a word, $h(s)$ can directly influence the planning efficiency. Fig. 1 shows the operation of A* in a cell map, in which the white stands for free cells, the black for obstacles and the grey for expanded cells.

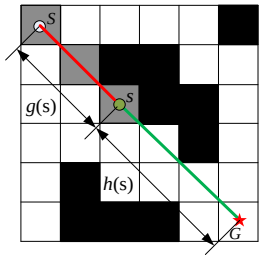

Figure 1. Sketch map of A* Search Algorithm

A* manages all states with two sets: *OPEN* and *CLOSED*. Expanded states belong to *CLOSED*. Except for obstacles, the adjacent states around an expanded state are defined as its sub-states. And sub-states of expanded states, which might be expanded in the future, belong to *OPEN*. The state with the lowest $f(s)$ in *OPEN* will be chosen for expanding. The algorithm does not stop until the goal state is expanded. Since every time A* chooses the state with the lowest $f(s)$ in *OPEN* for expanding, there will not be any state in *OPEN* which $f(s)$ is less than $f(s_{goal})$. So A* can always search out the optimal path.

Weighted A* multiplies $h(s)$ by inflation factor ε ($\varepsilon \ge 1$), so heuristic function turns into $h'(s) = \varepsilon * h(s)(\varepsilon \ge 1)$, while evaluation function $f(s)$ into $key(s)$:

$$key(s) = g(s) + h'(s) = g(s) + \varepsilon * h(s) \tag{3}$$

Weighted A* also chooses the state with the lowest $key(s)$ for expanding. This expanded state will be put into *CLOSED*. The algorithm will not stop until the goal state gets expanded, and when it stops, a feasible path will be searched out. Weighted A* with an inflated $h'(s)$ can effectively reduce the number of expanded states and quickly obtain a feasible solution. But if $h'(s)$ doesn't meet (2), the algorithm cannot ensure the optimality of the result, and one certain state might be expanded several times.

Since one state might be expanded several times if Weighted A* with $h'(s)$ cannot meet (2), ARA* will restrict state expansion more than once for a quicker search when executing A* under a certain ε. ARA* also has another set named *INCONS* for those may be re-expanded states. A state will be put into *INCONS* as a sub-state if it was in *CLOSED*. In this way, state re-expansion gets effectively limited. Though ARA* restricts state re-expansion, but its cost of the searching path under a given ε will not be larger than ε times of the cost of optimal path. Firstly ARA* gets an initial path under given in*fl*ation factor ε_0. Then it decreases ε and moves all states from *INCONS* into *OPEN*. With this new lower ε, ARA* will update and sort $key(s)$ of all states in *OPEN* by (3). Then ComputePath function will be executed again for better path. The pseudocode of ARA* is shown in Fig. 2.

The pseudocode of ARA*

```
1 ComputePath ()
2 while(key(s_goal) > min_{s ∈ OPEN} key(s) and OPEN ≠ Ø)
3   remove s with the smallest key(s) from OPEN;
4   CLOSED ← CLOSED ∪ {s}
5   for each successor s' of s
6    if g(s') > g(s) + c(s, s')
7      g(s') ← g(s) + c(s, s'); key(s')←g(s') + ε * h(s');
8     if(s' ∉ CLOSED)
9       insert or update s' in OPEN
10    else
11      insert s' into INCONS

12 procedure Main ()
13 g(s_start)←0 ; g(s) of the rest states are set to ∞ ;
14 OPEN←{s_start};CLOSED←Ø;INCONS←Ø;
15 ComputePath();
16 publish current ε0-suboptimal solution;
17 while(ε > 1)
18   decrease ε;
19   Move states from INCONS into OPEN;
20   Update priorities for all s∈OPEN according to key(s);
21   CLOSED←Ø;INCONS←Ø;
22   ComputePath();
23   publish current ε-suboptimal solution;
```

Figure 2. The pseudocode of ARA*

ARA* can get a sub-optimal solution quickly and reuse the previous planning efforts. ARA* puts the expanded states into *INCONS* and does not allow them to be re-expanded. However, the re-expansion of a certain state may make contribution to improving the optimality of the path. The key point is to make the cost of path not larger than ε times of that of the optimal path. Besides, under a

given ε, limitation on state re-expansion might cause more states expansion in another Weighted A* execution with a lower ε. As a result, we proposed an improved ARA* algorithm, ARA*+, which allows states re-expansion under a certain ε. And we will expand each state at most once during the first search, considering searching the feasible path quickly.

III. ARA*+ (ANYTIME REPAIRING A*+)

The pseudocode of ARA*+ is shown in Fig. 3. We set the g-values of all states except for s_{start} to infinity, and we set $g(s_{start})$ to 0 (Line 19). *OPEN* only includes the state s_{start}, *CLOSED* and *INCONS* present $\emptyset$ and set $\varepsilon = \varepsilon_0$. Then the ComputePath function will be executed. During the first execution of ComputePath, states will be expanded no more than once in order to lessen expanded states (Line 8-12). In this way, the search speed of a feasible path can be improved. When $key(s_{goal})$ is less than the minimum *key* in *OPEN*, the ComputePath function stops, and the algorithm produces a feasible path. On the other side, *OPEN* will be $\emptyset$ and the ComputePath function stops as a failed execution.

The pseudocode of ARA*+

```
 1 ComputePath ()
 2 while(key(s_goal) > min_{s ∈ OPEN} key(s) and OPEN ≠ Ø)
 3   remove s with the smallest key(s) from OPEN;
 4   CLOSED ← CLOSED ∪ {s}
 5   for each successor s' of s
 6    if g(s') > g(s) + c(s, s')
 7      g(s') ← g(s) + c(s, s'); key(s')←g(s') + ε * h(s');
 8      if(ε = ε0)
 9        if(s' ∉ CLOSED)
10          insert or update s' in OPEN
11        else
12          insert s' into INCONS
13      else
14        if(s' ∉ CLOSED )
15          insert or update s' in OPEN
16        else
17          removes s' from CLOSED to OPEN

18 procedure Main ()
19 g(s_start)←0 ; g(s) of the rest states are set to ∞ ;
20 OPEN←{s_start};CLOSED←Ø;INCONS←Ø;
21 ComputePath();
22 publish current ε0-suboptimal solution;
23 while(ε > 1)
24   decrease ε;
25   if(INCONS ≠ Ø)
26     Move states from INCONS into OPEN;
27   Update priorities for all s∈OPEN according to key(s);
28   CLOSED←Ø;
29   ComputePath();
30   publish current ε-suboptimal solution;
```

Figure 3. The pseudocode of ARA*+

When the algorithm comes to a feasible path, this path will be considered as the current sub-optimal path. If there is more time left, ε will be decreased, the states in *INCONS* will be removed to *OPEN* (Line 26), and $key(s)$ in *OPEN* will be updated (Line 26-27) with the lower ε by (3), along with the second execution of ComputePath function. During the second execution, the states in *OPEN* will be allowed to re-expand (Line 14-17). When ε differs from ε_0, a certain state may be re-expanded many times. If a certain state s gets re-expanding means its g value will be decreased again (Line 6), which may indicate a shorter path.

If there is more time left, ε of ARA*+ will be finally decreased to 1, and the optimal path will be produced. If not, the current sub-optimal solution will be considered as the output.

IV. SIMULATION EXPERIMENTS

Simulation experiments are conducted in Visual Studio 2008/C++. We are trying to show the differences between ARA* and ARA*+ in the number of states expanded, time and path optimality.

The simulation search map is generated randomly. Free cells and obstacle cells are determined by system random number, ranging from [0，1]. Then we set an obstacle boundary *OB*, and let the value of the state larger than *OB* will be defined as the obstacle cell, and the rest as the free cell. Fig.4 shows a 60×60 sized map and its optimal search result. The obstacle boundary number in Fig. 4 is 0.75, and the green line begins from the start cell *S* to the goal cell *G* is the optimal result. The black areas represent obstacle cells and white areas are free cells. In the following simulation maps, for convenience we set the start cell *S* and the goal cell *G* to be at left up corner and right bottom corner respectively.

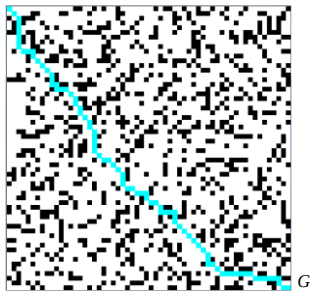

Figure 4. Simulation search result

In order to compare the number of expanded states and the time to get the optimal result between ARA* and ARA*+, we set two sizes of map: 800×800 and 600×600.

Fig. 5 is the comparison of the number of the expanded states in 20 800×800 sized simulation maps. Note that 20 maps are selected from 90 randomly built maps, the other 70 maps do not contain feasible paths from *S* to *G*. The Y-coordinate in Fig.5 stands for the number of expanded states when the optimal path is searched out, while X-coordinate represents the simulation map sequence number. A complete set of search comparison data in the third simulation is shown in Table I, in which ε stands for inflation factor, *EXPAND* for the number of expanded states, *COST* for the cost of path, and the total number of expanded states and total cost of the optimal path in this simulation are listed at the bottom of the table. The initial value of ε is set at 3.0, and then subtracts 0.2 every time for a lower ε after

executing the ComputePath function. The algorithm finally comes to the optimal path when $\varepsilon = 1.0$.

As shown in Fig. 5, the number of expanded states of ARA* exceeds that of ARA*+ for 19 times in the 20 simulation experiments. We can see in Table I, initially ε is set to 3.0, all states in both ARA* and ARA*+ are expanded no more than once, so these two algorithms will result in the same path when the number of expanded states are equal. When ε decreases to 1.6, though the re-expansion rule in ARA*+ makes it have more expanded states than ARA*, its cost (1699) is a bit lower than that of ARA*(1718); and when ε decreases to 1.4, ARA*+ still has more expanded states than ARA*, but ARA*+ has searched out the optimal path already; and when ε equals 1.2 or 1.0, ARA*+ has less expanded states than ARA*. The reason is ARA* has limitation on state *s* re-expansion under a relatively larger ε, which will influence state expansion when ε decreases and causes more states to be expanded later.

TABLE I. COMPARSION DATA IN 800×800 MAP

	ARA*		ARA*+	
ε	*EXPAND*	*COST*	*EXPAND*	*COST*
3	5891	1910	5891	1910
2.8	1	1910	1	1910
2.6	0	1910	0	1910
2.4	0	1910	0	1910
2.2	0	1910	0	1910
2.0	0	1910	0	1910
1.8	296	1910	455	1910
1.6	19481	1718	28058	1699
1.4	78389	1629	83590	1617
1.2	193802	1617	151776	1617
1.0	242908	1617	83580	1617
TOTAL	540768	1617	353351	1617

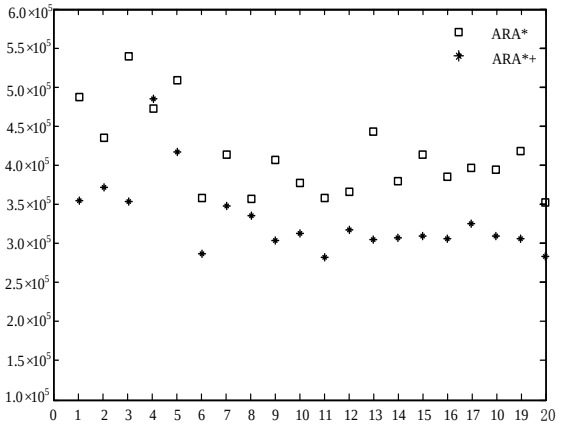

Figure 5. Comparison of the number of expanded states in Map800×800

In some cases, this state re-expansion limitation rule in ARA* is useful, for example, when the planning time is very short (in Table I, before when ε decreases to 1.4), ARA* can produce a better path. But in most cases, if there is enough time for planning, ARA*+ will search out a better path and be faster to obtain the optimal path.

We conduct another comparison in 600×600 sized maps. The results of 37 600×600 sized maps are listed in Fig. 6 and Table II. Also, the 37 maps with feasible path are randomly built. As a result, in 36 of them ARA*+ has less states expansion than ARA*. Fig. 6 shows experimental data in 600×600 sized maps, which indicates that ARA*+ can effectively reduce the number of expanded states when compared to ARA*. Table II shows a complete set of search comparison data in the third simulation. When ε equals 1.2, ARA*+ has less states expansion than ARA* and plans the optimal path already. In contrast, ARA* cannot search out its optimal path until ε reduces to 1.

TABLE II. COMPARSION DATA IN 600×600 MAP

	ARA*		ARA*+	
ε	*EXPAND*	*COST*	*EXPAND*	*COST*
3	3422	1325	3422	1325
2.8	6	1325	6	1325
2.6	23	1325	26	1325
2.4	43	1325	56	1325
2.2	124	1325	1109	1325
2.0	676	1325	528	1325
1.8	1203	1301	1369	1301
1.6	2218	1301	5583	1289
1.4	18763	1187	33804	1180
1.2	120691	1173	120936	1171
1.0	154562	1171	47781	1171
TOTAL	301731	1171	215620	1171

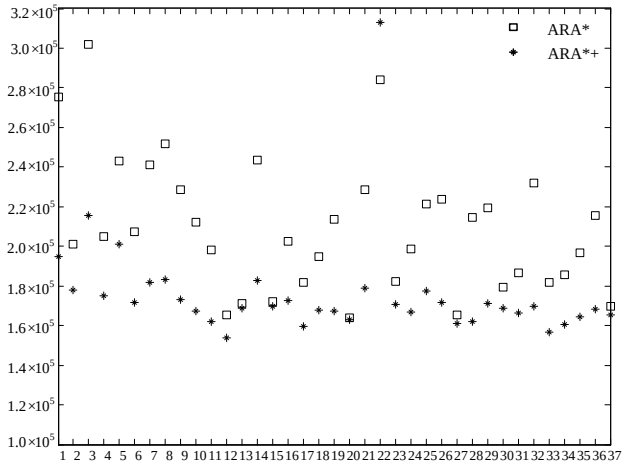

Figure 6. Comparison of the number of expanded states in Map600×600

In the above two comparisons, the obstacle boundary *OB* is set to be in [0.65, 0.69], which means the randomly built maps will obtain more obstacles. In order to compare the performance in maps with fewer obstacles, we set the obstacle boundary *OB* to be in [0.75, 0.85] in the next comparison (in Fig. 7). And we can see that ARA*+ and ARA* almost show the same search efficiency in this kind of simple environment.

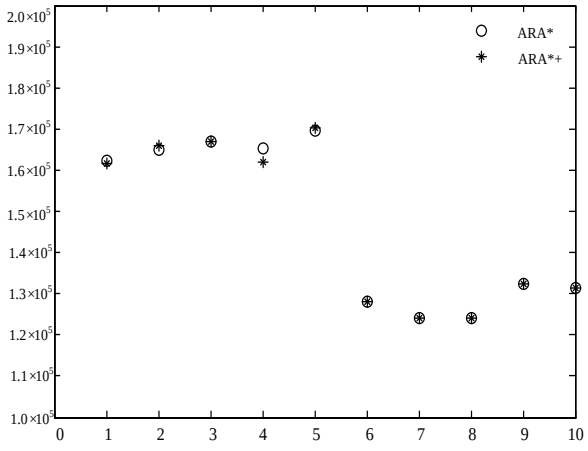

Figure 7. Expand states in Map0.75&0.85

V. CONCLUSIONS

In this paper, a variation of ARA* algorithm, ARA*+, is introduced for robot path planning. ARA*+ will execute multiple Weighted A* to search the solution, and conduct the same first search as ARA*. They both execute Weighted A* with a relatively larger inflation factor ε_0 and no state is expanded more than once for a feasible path quickly. The difference between ARA*+ and ARA* is that ARA*+ allows the re-expansion after the inflation factor ε is decreased. And our robot path planning simulation experiments in cell maps showed that this small change could improve the search efficiency in most cases, especially in complicated environments.

ACKNOWLEDGMENT

The work was partly supported by Project (Nos. 51275041 and Nos. 91120015) of National Science Foundation of China.

REFERENCES

[1] S. M. LaValle, "Planning algorithms," Cambridge University Press. 2006.

[2] S. M. LaValle, "Rapidly-exploring random trees: a new tool for path planning," Technical Report, No. 98-11. 1998.10.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," International Journal of Robotics Research, 2011, pp. 846-894.

[4] P. E. Hart, N. Nilsson and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," IEEE Transactions on Systems Science and Cybernetics(SSC), 1968, pp. 100-107.

[5] I. Pohl, "Heuristic search viewed as path finding in a graph," Artificial Intelligence, 1970, pp. 193–204.

[6] T. Dean and M. Boddy, "An analysis of time-dependent planning," Seventh National Conference on Artificial Intelligence(AAAI 88), St. Paul, MN, USA, 1988, pp. 49-54.

[7] S. Zilberstein and S. Russell, "Approximate reasoning using anytime algorithms," Imprecise and Approximate Computation, Kluwer Academic Publishers, 1995, pp. 1-19.

[8] R. Zhou, E. Hansen and A. Eric, "Multiple sequence alignment using anytime A*," In Proceedings of the National Conference on Artificial Intelligence(AAAI), 2002, pp. 975-976.

[9] M. Likhachev, G. Gordon and S. Thrun, "ARA*: Anytime A* with Provable Bounds on Sub-Optimality," Proceedings of Advances in Neural Information Processing Systems 16(NIPS), 2003, pp. 258-265.

[10] E. A. Hansen and R. Zhou, "Anytime heuristic search," Journal of Artificial Intelligence Research, 2007, pp. 267-297

.