



HAL
open science

DEC-A*: a decentralized extension of A

Mohamad El Falou, Maroua Bouzid, Abdel-Allah Mouaddib

► **To cite this version:**

Mohamad El Falou, Maroua Bouzid, Abdel-Allah Mouaddib. DEC-A*: a decentralized extension of A. "Multiagent pathfinding" workshop of AAAI 2012, 2012, Toronto, Canada. p12-18. hal-00966902

HAL Id: hal-00966902

<https://hal.science/hal-00966902>

Submitted on 15 Apr 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

DEC-A*: a decentralized extension of A*

Mohamad EL Falou and Maroua Bouzid and Abdel-Ilah Mouaddib

University of Caen France

Email: firstname.lastname@unicaen.fr

Abstract

A* is the algorithm of finding the shortest path between two nodes in a graph. When the planning problem is constituted of a set of linked graphs, A* searches solution like if it is face of one graph formed by linked graphs. While researchers have developed solutions to reduce the execution time of A* in multiple cases by multiples techniques, we develop a new algorithm: DEC-A* which is a decentralized version of A*. A* uses a distance-plus-cost heuristic function to determine the order in which the search visits nodes in the tree. Our algorithm DEC-A* extends the evaluation of the distance-plus-cost heuristic be the sum of two functions : local distance, which evaluates the cost to reach the nearest neighbor node s to the goal, and global distance which evaluates the cost from s to the goal through other graphs. DEC-A* accelerates finding the shortest path by reducing the time and space complexity, while ensuring the privacy of graphs.

1 INTRODUCTION

Many problems of engineering and scientific importance can be related to the general problem of finding a shortest path through a graph. Examples of such problems include routing of telephone traffic, navigation through a maze, computer games, travel planning, etc. These kinds of problem can be represented by a graph constituted of a set of nodes and a set of edges. Nodes represents the different states of the domain (i.e. cities in the transportation problem) and edges represents the actions that can be executed to change the states (i.e. take a plane).

Let us take the travel example. Suppose that we have three cities: Montpellier, Lyon and Paris. Montpellier and Lyon are linked by a train; Lyon and Paris are linked by an airplane. In each city, we have a set of places linked by buses and metros. Suppose that a person wants to travel between Paris and Montpellier downtowns. This problem can be defined by a graph where nodes are the places in the cities and edges are the actions performed by buses, metros, airplane or train. In this case, A* algorithm can be used to search the shortest path. A* must be executed on one machine and cannot take advantage of the decentralized nature of the prob-

lem. In addition it supposes that there is no privacy information and the graph of each city is observable from other cities.

In many cases, the systems for which we plan are naturally viewed as Multiagent (MA) systems. For example, the travel company is composed of multiple agents that perform the travel planning: the trains/airplanes that travel the passengers between airports/railway stations, and the local buses/metros that transport within a certain locality. One may pose the following question: "Can a decentralized planner, planning for such a system exploit its MA structure in order to improve its worst case time-complexity and preserves graphs privacy ?". We think that answer is positive and this is what we will try to show you in this paper by our new DEC-A* algorithm.

The travel problem cited above can be modeled and solved in a decentralized manner. Instead of defining one graph that contains all places and links of cities and between them, an individual graph will be associated to each city. A set of links will be then defined between the set of graphs.

Our new algorithm DEC-A* extends the distance-plus-cost heuristic function ($f(s) = g(s) + h(s)$) to determine the order in which the search visits nodes in the tree. It determines also the order of triggering A* in the neighbors graphs.

The extended function $f(s)$ which estimates the cost of the shortest path through s will be defined by the sum of three functions :

- $g(s)$ which is the path-cost from the initial state to s .
- $h_{local}(s)$ which is the evaluation of the cost to reach the neighbor state $s^{neighbor}$ located on the (estimated) shortest path to the goal.
- $h_{global}(s)$ which is equal to the estimation of the shortest path-cost to reach the goal from $s^{neighbor}$.

The paper is structured as follows. The next section defines the preliminaries of the decentralized architecture. The basic idea is illustrated in section 3. Section 4 illustrates a motivating example of our algorithm. The decentralized algorithm is developed in section 5. Section 6 studies the properties of the algorithm. The comparison with the related

¹The neighbor-states of a graph are their states linked by a link to another graph.

works is presented in section ???. The paper is concluded in section 8 by summarizing the contribution of the paper and presents suggestions for future work.

2 preliminaries

The planning domain \mathcal{D} is defined by a set of graphs $\langle G_1, \dots, G_n \rangle$ and a set of links between graphs $\langle l_{ij}, \dots, l_{kl} \rangle$. Each graph G_i is defined by a finite set of states S_i and a finite set of actions $Actions_i$ that can be executed in a state $s \in S_i$. $c_i(s, a)$ denotes the action cost of executing action $a \in Actions_i$ in state $s \in S_i$, and $succ_i(s, a)$ denotes the resulting successor state. Each link $l_{ij} = s_i \rightarrow s_j$ has two states s_i, s_j included in distinct graphs $G_i, G_j; i \neq j$. $c(l_{ij})$ denotes the link cost, $neighbor(G_i)$ denotes all the graphs linked with G_i and $S_{neighbor}^i$ denotes the set of neighbor states of the graph G_i .

The planning problem is defined by $init$ which denotes the start state, and $goal$ which denotes the end states. $init, goal \in \{S_1 \cup \dots \cup S_n\}$.

In a graph G_i , we refer to an action sequence as local path π_i . In the domain \mathcal{D} , the solution of the planning problem is a global path Π defined by a sequence of local path $\langle \pi_1, \pi_2, \dots, \pi_n \rangle$ linked by links $\langle l_{1,2}, \dots, l_{n-1,n} \rangle$.

3 BASIC IDEA

A planning agent \mathcal{A}_i is associated to each graph G_i . A central agent \mathcal{A}_c plays the role of an interface between users and agents. \mathcal{A}_{init} (resp. \mathcal{A}_{goal}) denotes the agent containing the initial (goal) state.

When a planning problem is submitted, each agent \mathcal{A}_i computes its global heuristic which estimates the cost of its shortest path to the goal through its neighbor agents.

Then, the agent \mathcal{A}_{init} containing $init$ develops A* locally by minimizing $f(s)$ described above until reaching s_{goal} (if $s_{goal} \in G_{init}$) or all neighbor states $s_{neighbor}$ of $S_{neighbor}$ are reached.

Each reached neighbor state s_i activates at the other side a new A* execution on s_j as new initial state and $g(s_j)$ is initialized by the cost to reach it from $init$.

The last step cited above is repeated until reaching s_{goal} .

4 MOTIVATING EXAMPLE

We use a path-planning problem on a connected grid-world of square cells (such as, for example, grid-world used in video games) as examples (Figure 1). All cells are either blocked (=black) or unblocked (=white). Our domain example is defined by sixteen grids delimited by horizontal and vertical red dashed-lines. Each grid G_{ij} associated to an agent \mathcal{A}_{ij} is defined by 9 cells $(a, b, c, d, e, f, g, h, i)$. The agent knows its current cell, the $goal$ cell, and which cells are blocked.

The agent can change its current cell to one of the four neighboring cells by executing one of the four possible actions $\uparrow \rightarrow \downarrow \leftarrow$. Depending on its position and the number of blocked cells, each agent has maximum 12 links to move from one grid to another $(a, b, g, i$ for the agent $\mathcal{A}_{3,2}$). When the agent crosses a red line, it means that it executes a link action. The action cost for moving from an unblocked cell

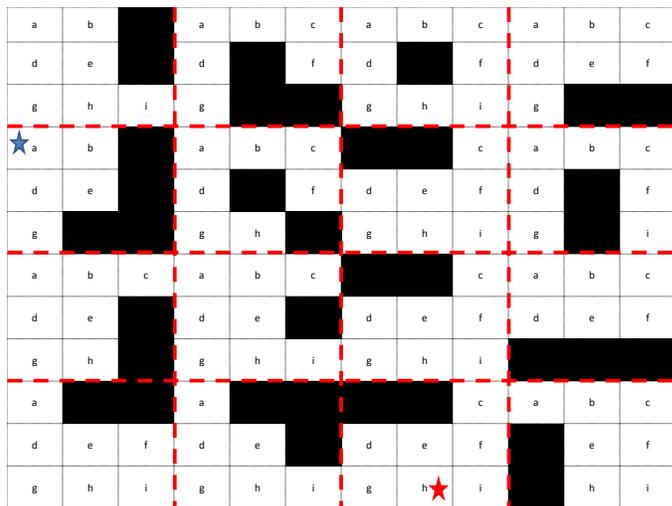


Figure 1: Example

to an unblocked cell is one. The agents must find the shortest path from the start state $a \in \mathcal{A}_{2,1}$ (blue star) to the goal state $h \in \mathcal{A}_{4,3}$ (red star). We use the Manhattan distances as user-supplied consistent heuristic.

5 DECENTRALISED ALGORITHM

In this section, we give two decentralized strategies of DEC-A*. The first one is based on the global and local heuristics as explained in the introduction. The basic idea behind the second strategy is to reduce the time execution of DEC-A*, by computing off-line the lowest cost path between each couple of neighbor states of each agent.

First strategy

Global distributed heuristic

The global distributed heuristic h_g is computed per agent \mathcal{A}_i . Let $\Pi = \langle \pi_1, l_{1,2}, \pi_2, \dots, l_{n-1,n}, \pi_n \rangle$ the shortest path from $init$ to $goal$. Then, h_g estimates the cost of the sub-plan $\langle \pi_i, l_{i,i+1}, \dots, l_{n-1,n}, \pi_n \rangle$.

Algorithm 1 explains the general steps to compute the global heuristic. Firstly, each agent computes the cost of crossing it to reach the goal (i.e. $cost(\pi_i)$ line 2)² by calling Algorithm 2. Then, all the global heuristic are initialized by $+\infty$ (3-5) and the Algorithm 3 is called (7) to compute the global heuristic h_g of each agent (it starts recursively from \mathcal{A}_{goal}).

Algorithm 2 computes firstly the distance between each neighbor state (4) and the init (5) (resp. goal (6)) state. Then, it selects its nearest neighbor state to init (8) and its nearest neighbor state to goal (9). Finally, the cost of crossing \mathcal{A}_i to reach the goal from init is estimated by the distance between this couple of states (10).

Algorithm 3 is a recursive procedure which diffuses the global heuristic from \mathcal{A}_{goal} to \mathcal{A}_{init} through other agents.

²In the rest of this paper, when we explain algorithms, we use (x) and mean (line x)

That's why Algorithm 3 is executed for the first time on (\mathcal{A}_{goal}) in Algorithm 1. Each agent computes its global heuristic by minimizing the cost estimation to reach the goal by crossing it (3). The estimated cost of \mathcal{A} to reach the goal through a neighbor agent (\mathcal{A}_i) is equal to its last global heuristic $h_g(\mathcal{A}_i)$, plus the cost of crossing it $cost(\mathcal{A})$. If $h_g(\mathcal{A})$ is updated (4) then the neighbor agents will check their g_h to see if they can minimize it. Recursive procedure will turn until no heuristic value can be minimized.

Algorithm 1: Global heuristic computation

```

1 HeuristicGlobal( $\mathcal{A}_1, \dots, \mathcal{A}_n$ )
2 Vector Cost < Agent > = costAgents { $\mathcal{A}_1, \dots, \mathcal{A}_n$ }
3 for  $i = 1..n$  do
4   |  $h_g(\mathcal{A}_i) = +\infty$ 
5 end
6  $h_g(\mathcal{A}_{goal}) = Cost(\mathcal{A}_{goal})$ 
7 heuristicAgent( $\mathcal{A}_{goal}$ )

```

Algorithm 2: Cost of crossing agent

```

1 costAgents( $\mathcal{A}_1, \dots, \mathcal{A}_n$ )
2 for  $i = 1..n$  do
3   |  $d_{init} = d_{goal} = Vector(n)$ 
4   | foreach  $s^{min} \in S^{min}$  do
5     |  $d_{init}[s^{min}] = distance(s^{min}, init)$ 
6     |  $d_{goal}[s^{min}] = distance(s^{min}, goal)$ 
7   | end
8   |  $s_{init}^{min} = \arg \min_{s_i} (d_{init}(s_i) / s_i \in S^{min}(\mathcal{A}_i))$ 
9   |  $s_{goal}^{min} = \arg \min_{s_i} (d_{goal}(s_i) / s_i \in S^{min}(\mathcal{A}_i))$ 
10  |  $cost(\mathcal{A}_i) = distance(s_{init}^{min}, s_{goal}^{min})$ 
11 end

```

Algorithm 3: Recursive agent heuristic procedure

```

1 HeuristicAgent( $\mathcal{A}$ )
2  $h = h_g(\mathcal{A})$ 
3  $h_g(\mathcal{A}) = \arg \min_{\mathcal{A}_i} (h_g(\mathcal{A}_i) + cost(\mathcal{A}) / \mathcal{A}_i \in neighbor(\mathcal{A}))$ 
4 if  $h \neq h_g(\mathcal{A})$  then foreach  $\mathcal{A}_j \in neighbor(\mathcal{A})$  do
5   | HeuristicAgent( $\mathcal{A}_j$ )
6 end

```

Example The Figures 2 and 3 illustrates the steps of computing the global decentralized heuristic for the problem illustrated in Figure 1.

Firstly, the first line of algorithm 1 is executed. It calls Algorithm 2 to compute the cost of crossing each agent. Let us take the agent \mathcal{A}_3^2 . It has four neighbor states : a, b, g and i. States a and b are linked to \mathcal{A}_2^2 , g to \mathcal{A}_4^2 and i to \mathcal{A}_3^3 . The distance from each neighbor to the init is placed in the top

left and to the goal on the bottom right. The distance from node a to init is 5 and from node a to goal is 9.

State a is the nearest neighbor state to init and nodes g and i are the nearest states to goal (surrounded by a circle). \mathcal{A}_3^2 has the choice between $distance(a, g) = 2$ and $distance(a, i) = 4$. It minimizes its cost and choice the value 2 placed in the rectangle \mathcal{A}_3^2 , and so on for the other agents.

The agents costs are copied in the top left of Figure 3. Each square designs an agent. In the top right, the global heuristic of each agent is initialized by $+\infty$ (lines 3-5 in Algorithm 1) and the goal agent \mathcal{A}_4^3 initializes its global heuristic with its cost equal to 0 (line 6). The global heuristic h_g of each agent is placed in the top right of the agent square.

\mathcal{A}_4^3 diffuses its global heuristic to its neighbors $\mathcal{A}_3^3, \mathcal{A}_4^2, \mathcal{A}_4^4$ (Algorithm 3). For example, $h_g(\mathcal{A}_3^3) = \min(\infty, 2 + 0) = 2$ (line 3); Then each agent having updated g_h diffuses it to its neighbors (line 4-6). For example, \mathcal{A}_3^3 diffuses its g_h to $\mathcal{A}_4^3, \mathcal{A}_3^2, \mathcal{A}_3^4, \mathcal{A}_2^3$.

\mathcal{A}_4^3 minimizes its g_h by minimizing its old g_h and the g_h of reaching it from its neighbors \mathcal{A}_4^2 with $g_h=4$, \mathcal{A}_3^3 with $g_h=2$ and \mathcal{A}_4^4 with $g_h=0$ added to its cost (0) ($\min(0, 4+0, 2+0, 0+0, 0) = 0$).

Agents coordination

In this section, we explain how agents proceed and coordinate to reach s_{goal} .

Minimizing the cost to reach goal The basic idea behind the local heuristic h_l is to help agent \mathcal{A}_i to leave its graph via the neighbor agent having the minimal cost to reach the goal.

To do this, the agent executes A* on its own graph from its initial state by using the function $f(s)$ which estimates the cost of the cheapest solution through s .

Let s_i to be the neighbor state linked by l_{ij} to \mathcal{A}_j via s_j where \mathcal{A}_j is the agent having the minimal global heuristic, then $f(s)$ is defined by $f(s) = h_l(s) + c(l_{ij}) + h_g(s)$ where :

- $h_l(s) = distance(s, s_i)$,
- $c(l_{ij})$ the cost of l_{ij} ,
- $h_g(s_j)$ the global heuristic of \mathcal{A}_j .

When the agent \mathcal{A} reaches a neighbor state s_i , it continues developing its graph to leave it by the second nearest neighbor state s_2 and so on until leaving all its neighbor states.

To do this, it does not re-executes A* from scratch but it uses the tree developed by A* to reach s_i by updating the local and global heuristics (h_l and h_g) values of developed states with respect to state s_2 . \mathcal{A} continues executing its A* until reaching all the neighbor states.

Coordination between agents at the neighbor states level

When an agent \mathcal{A}_i reaches a neighbor state s_i linked to another neighbor state $s_j \in \mathcal{A}_j$, it activates a new A* execution on \mathcal{A}_j . \mathcal{A}_j sets s_j as its initial state and sets its path cost to: $g(s_j) = g(s_i) + c(l_{ij})$.

Let T_j to be the tree-search of \mathcal{A}_j . \mathcal{A}_j might be reached by another neighbor state s_k . This means that our agent will

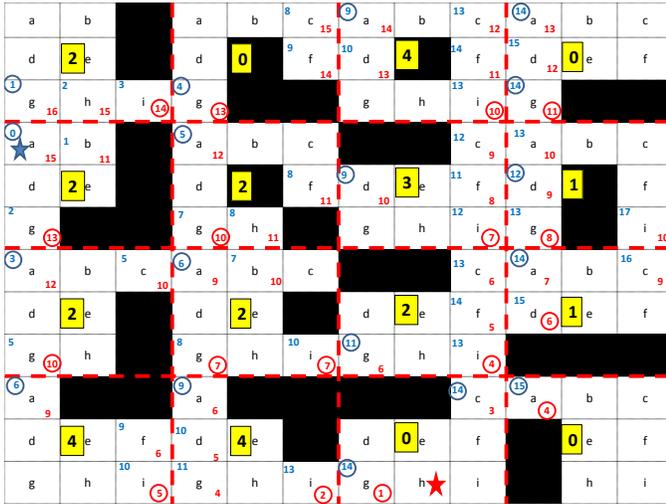


Figure 2: Global heuristic computation, step 1

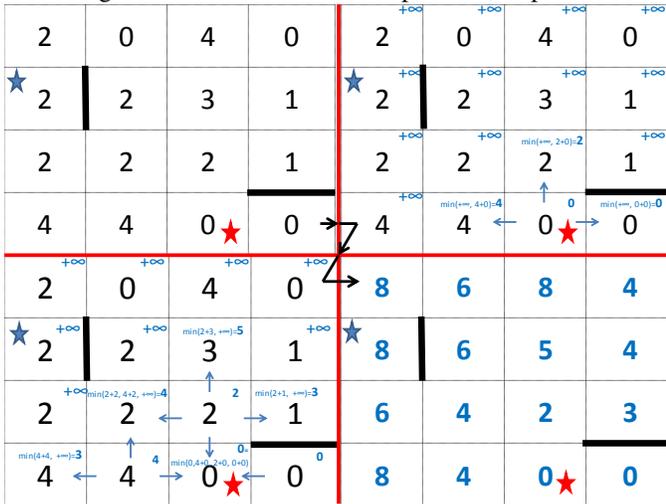


Figure 3: Global heuristic computation, step 2

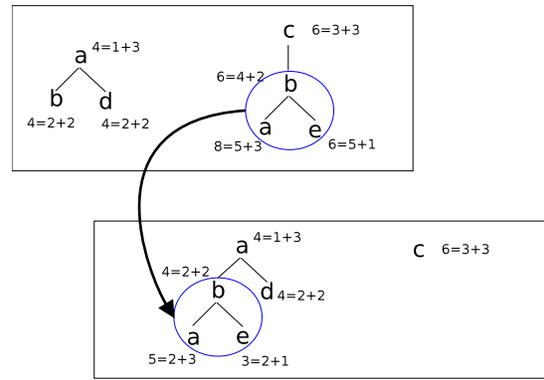


Figure 4: Graph agent

have more than one initial (neighbor activation) state (s_j and s_k). In this situation, two cases can be distinguished:

- $s_k \notin T_j$, in this case the agent adds s_k to T_j even if it does not have any link with the other nodes. Then, A* continues its execution on the set of node by minimizing function $f(s)$ as above (section 5), until reaching all its neighbor states. Let T_k the sub-tree-search of the tree-search of \mathcal{A}_j developed from s_k .

During states expansion, one tree-search, let us say T_j , may create a state s including in the second, i.e. T_k . Let $g_j(s)$ (resp. $g_k(s)$) the path-cost of reaching s in T_j (resp. T_k). If $g_j(s) \geq g_k(s)$ then the link that creates s in T_j is pruned from its entering link in T_j and linked to s in T_k (vice-versa). By doing that, a new sub-tree-search is added to T_k (state s and its child states). Finally, the path-cost of all s childs are updated.

Example: Figure 4 illustrates such example (path-code is the left operand). Let us suppose that initially, the agent develops its tree from c , then develops node b (on the right of Figure 4). Suppose now that the agent is activated via node a which creates nodes b and d (On the left of Figure 4). We can see that node b is created by the two sub-tree. Since $6 = g_{right}(b) > g_{left}(b) = 4$, then b is pruned from the right tree and linked to the left one, and the path-costs are updated.

- $s_k \in T_j$, then $g(s_k) = \min(g_j(s_k), g_k(s_k))$.

When the cost-path of a state is changed, the costs of its child in the tree-search are updated (intra and inter agent).

All agents progress in parallel and coordinates until reaching the goal state s_{goal} and having a first approximate solution Π_i . Let $g(\pi_i)$ its cost.

Then the goal agent diffuses this cost to all agents. Each one stops developing its states s having $f(s) > g(\Pi_1)$. We call active agent an agent having at least a non-developed state.

When the goal state is reached by a new plan Π_{i+1} having a cost $g(\Pi_{i+1}) < g(\Pi_i)$, then $g(\Pi_{i+1})$ is diffused to all active agents until reaching a step where no active agent exists. Then the optimal agent can be easily extracted.

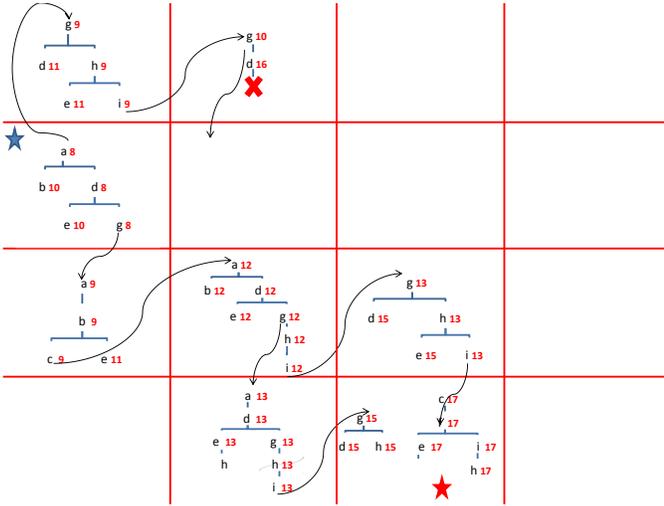


Figure 5: Search graphs

Example The figure 5 illustrates a scenario of DEC-A* execution. For readability reason, we will not give the complete execution, but multiple situations explaining the algorithm.

The agent $\mathcal{A}_{2,1}$ triggers its A*. Between its neighbor agents, $\mathcal{A}_{3,1}$ has the minimal global heuristic (6). Since $\mathcal{A}_{2,1}$ is linked to $\mathcal{A}_{3,1}$ with g , $\mathcal{A}_{2,1}^2$ develops its graph as illustrated in the Figure 5 until reaching g .

By the same way, A* is triggered sequentially by agents \mathcal{A}_3^1 , \mathcal{A}_3^2 , \mathcal{A}_3^3 , and \mathcal{A}_4^1 until reaching the goal states h with 17 as cost. This cost is diffused to all agents.

At the same time:

- \mathcal{A}_3^2 reevaluates $f(s)$ for all its developed nodes to reach the second nearest state to goal, i . So A* is re-executed by \mathcal{A}_3^2 from state a until reaching i and \mathcal{A}_4^1 do the same to reach the goal. Since the cost of the new plan solution is 15, it is diffused and the new plan replace the first one.
- A* is triggered on \mathcal{A}_1^1 from state g to reach i . Then from i , \mathcal{A}_1^1 triggers A* on g . During A* execution, \mathcal{A}_1^1 will reach state g and receive the new lowest cost-path founded, (15). Since, $f(d) = 16 > 15$, \mathcal{A}_1^2 stops developing d .

Second strategy

Offline local path cost

In this section, we develop another strategy to be used by DEC-A*. Its basic idea is to compute off-line the minimal cost path between each couple of neighbor states of each agent.

This strategy is illustrated in Algorithm 4 and Algorithm 5. DEC-A* is initialized on the init and the goal states.

Fist of all, Algorithm 4 initializes by $+\infty$ the $g_{init}(s_n)$ and $g_{goal}(s_n)$ for each agent. $g_{init}(s_n)$ and $g_{goal}(s_n)$ are the costs to reach each neighbor state s_n from $init$ and $goal$ respectively.

Then, the initial (resp. goal) agent computes the optimal path from the initial (goal) state to each neighbor state lines 3-4 in Algorithm 4.

From each neighbor state in \mathcal{A}_{init} and \mathcal{A}_{goal} , the costs are diffused to the neighbor agents via links between their neighbor states (Algorithm 5).

Algorithm 5 explains how diffusing the costs from $init$ to $goal$. The same algorithm is used to diffuse the costs from $goal$ to $init$ by replacing $init$ with $goal$.

Since costs g_{init} , g_{goal} are diffused in parallel, it may intersect in a state s . This mean from s , we can construct a plan g_{Π} constituted of two sub-path. The first one is constituted from $init$ to s and the second from s to $goal$.

Let $S_{neighbor}$ to be the set of neighbor states of agent \mathcal{A} and g_{Π}^{min} to be the minimum cost founded by the agents $\{\mathcal{A}_1, \dots, \mathcal{A}_n\}$ at instant t .

During diffusion, the costs are minimized for each s_n (2-3). If a state is reached from $init$ and $goal$, then the diffusion by this neighbor states is stopped. If g_{Π}^{min} can be re-minimized (6-9) then its new value is diffused to all agents.

In lines (11-14), the agent continues diffusing its g_{init} via all its neighbor agents if it does not exceed g_{Π}^{min} . Otherwise, s_n cannot reduce the optimal solution and it is deleted from $S_{neighbor}$ (16). In line 17, the agent is deactivated which mean that it cannot contribute to improve the solution if its $S_{neighbor}$ becomes empty.

When all the agents become inactivated (line 6 in Algorithm 4, the goal agent which stoks the best plan (line 9 in Algorithm 5) extracts the optimal path.

This strategy supposes that the planning domain is static to be efficacy. It can be extended to find the optimal plan between a state in a set of initial state to one state in a set of goal state.

Algorithm 4: DEC-A* : second strategy

```

1 DEC – A*(init, goal)
2 initialiseAgents()
3 diffuseinit( $\mathcal{A}_{init}$ , init)
4 diffusegoal( $\mathcal{A}_{goal}$ , goal)
5 while true do
6   | if  $\forall i, -activated(\mathcal{A}_i)$  then  $\Pi =$ 
7     |    $\mathcal{A}_{goal}.extractSolution()$ 
7 end

```

Example In Figure 6, we illustrate a part of the execution of the second strategy to resolve the problem illustrated in 4. Firstly, each agent computes **offline** the cost of the optimal path between each couple of its neighbor states. For example, agent \mathcal{A}_{21} has three neighbor states, a, b, g . It computes $cost(a, b) = 1$, $cost(a, g) = 2$, $cost(b, g) = 3$.

The initial agent \mathcal{A}_{21} computes the distance between the initial state a and its neighbor states b and c which are equal to 1 and 2. the goal agent \mathcal{A}_{43} computes also the distance between the goal state h and its neighbor states g and c which are equal to 1 and 3. As we can see, \mathcal{A}_{21} sets the value for reaching g from the intial state a with 2 and for reaching b with 1. \mathcal{A}_{43} also sets the value for reaching g from the goal state g with 1 and for reaching c with 3.

At the same time, \mathcal{A}_{21} and \mathcal{A}_{43} diffuses their costs :

Algorithm 5: Diffusing costs via neighbor states of agents

```

1 diffusesinit( $\mathcal{A}, s$ )
2 foreach  $s_n \in S_{neighbor}(\mathcal{A}) \setminus s$  do
3    $g_{init}(s_n) = \min(g_{init}(s_n), g_{init}(s) + cost(s, s_n))$ 
4   if ( $g_{goal}(s_n) \neq +\infty$ ) then
5      $g = g_{init}(s_n) + g_{goal}(s_n)$ 
6     if  $g < g_{\Pi}^{min}$  then
7        $g_{\Pi}^{min} = g$ 
8       broadcast( $g_{\Pi}^{min}$ )
9       notify( $\mathcal{A}_{goal}, s_n, g_{\Pi}^{min}$ )
10    else
11      if ( $g_{init}(s_n) < g_{\Pi}^{min}$ ) then
12        foreach  $\mathcal{A}_i \in neighbor(\mathcal{A}, s_n)$  do
13          | diffusesinit( $\mathcal{A}_i, s_n$ )
14          end
15        else
16           $S_{neighbor} = S_{neighbor} \setminus s_n$ 
17          if  $S_{neighbor} == \emptyset$  then deactivate( $\mathcal{A}$ )
18          end
19    end
20 end

```

1. \mathcal{A}_{21} diffuses costs respectively from a and b to g and h in \mathcal{A}_{11} (we omit this part here), and from g to a in \mathcal{A}_{31} . From the initial state, the cost of reaching a in \mathcal{A}_{31} is equal to '3': cost of $g(g) = 2$ in \mathcal{A}_{21} , plus cost of link $c(l_{ga}) = 1$.
2. \mathcal{A}_{43} diffuses costs respectively from g to i in \mathcal{A}_{42} , and from c to i in \mathcal{A}_{33} . From the initial state, the cost of reaching i in \mathcal{A}_{33} is equal to '3': cost of $g(c) = 3$ in \mathcal{A}_{43} , plus cost of link $c(l_{ci}) = 1$ (similarly $g(i) = 2$).

\mathcal{A}_{31} , \mathcal{A}_{42} and \mathcal{A}_{33} continue diffusing their cost. In \mathcal{A}_{32} , costs from init and goal intersect in states b ($[7, 10]$) and g ($[8, 7]$). From this intersection, the optimal path having 15 as cost can be extracted from state g .

6 Completeness, optimality and complexity

In this section, we compare the complexity, optimality and completeness of A^* and DEC- A^* to prove the efficacy of our algorithm.

Complexity

The time and the space complexity of A^* is exponential. It are equal to $O(b^m)$ where b is the branching factor and m is the solution depth. Computation time is not, however, A^* 's main drawback. Because it keeps all generated nodes in memory, A^* is not practical for many large-scale problems.

The importance of our decentralized algorithm DEC- A^* is to overcome the space problem by decomposing the computation space while also reducing the execution time.

In our approach, each graph can be associated to an agent. Agents can be deployed at multiple machines to execute their local A^* in parallel. At first glance, the state space and the time execution will be divided between agents. Let l be

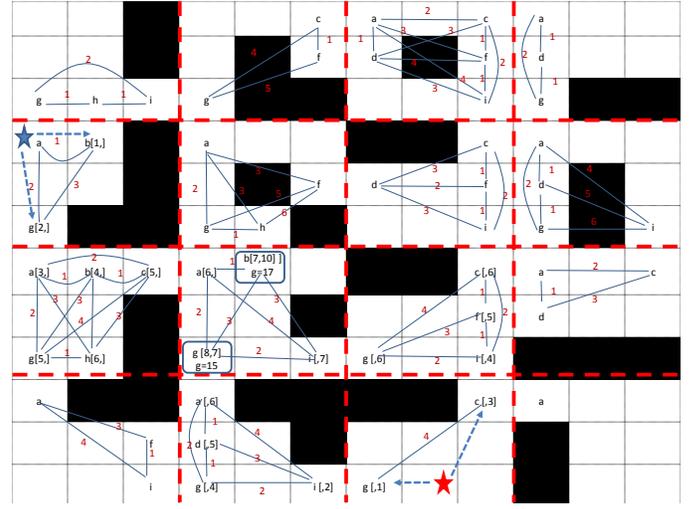


Figure 6: Second strategy

the number of agent that contains a fragment of the optimal solution. Then the space complexity of DEC- A^* is equal to $O(b^{d/l})$ and the time complexity is equal to $O(n * b^{d/l})$. It is clear that from the theoretical point of view, our Dec- A^* reduce greatly the complexity.

Completeness

The completeness of A^* is ensured because that, in the worst case, it develops all the possible states. DEC- A^* is also complete by using the first or the second strategy, because it develops all the possible states except those having $f(s) > cost(\Pi)$ where Π is the shortest founded plan. In this case, DEC- A^* prunes state after finding a plan, so DEC- A^* is complete.

Optimality

The optimality of A^* is ensured if the heuristic function h is admissible. This means that it never overestimates the cost to reach the goal (i.e. $foreach\ nodes, h(s) < g(s)$). A^* finds the optimal path more quickly that h is close to g .

Our DEC- A^* is also optimal because, except pruned path, it computes all the possible path and choice the optimal one. Pruned path does not prevent the optimality since they are pruned because their costs are more than the cost of the shortest path.

Messages number

7 RELATED WORKS

?? A number of classical graph search algorithms have been developed for resolving the problem of calculating the optimal sequence of actions that reach a goal state from an initial state on a weighted graph; two popular ones are: Dijkstras algorithm ((Dijkstra 1959)) and A^* ((P. E. Hart and Raphael 1968)).

Since discovering these algorithms, researchers working on our problem concentrate their interest on operating in

real world scenarios when the planner do not have complete information. In this case, path generated using the initial information about the graph may turn out to be invalid or suboptimal as it receives updated information. Using the classical algorithms to reach the goal state by re-planning from scratch is a waste of computation. That's why, A number of extended algorithms from A* like (D*(Stentz 1995) D* light(Koenig and Likhachev 2002b) ARA* (Likhachev, Gordon, and Thrun 2004)) was developed to take the previous solution and repair it to account for the changes to the graph. In the last decade, Sven Koenig et al. are interested to solve a series of similar planning tasks faster than by solving the individual planning task in isolation, see: Incremental A* (Koenig and Likhachev 2002a), Real-Time Adaptive A* (Sun, Koenig, and Yeoh 2006), Tree Adaptive A* (Hernández et al. 2011).

To the best of our knowledge, no one has worked on extending A* when the planning domain is constituted of a set of linked graphs. This means that finding the optimal path between two nodes in the domains should use A* without taking advantage of the decentralization of the domain.

In contrast, there is a lot of works in artificial intelligence on developing distributed algorithms to resolve centralized or decentralized problems. We can cite the DEC-MDP (Sigaud and Buffet 2010) modeling decision-making in Markovian process, multiagent automated planning based on STRIPS model (Shoham and Leyton-Brown 2009) or based on distributed CSP to coordinate between agents and local planning to ensure the consistency of these coordination points (Nissim, Brafman, and Domshlak 2010).

Our extension algorithm DEC-A* is inspired from a new distributed multiagent planning approach proposed in (Falou et al. 2010). In this approach, agents coordinate by proposing their best plans evaluated basing on local heuristic and global heuristic. The local heuristic estimates the distance to the goal. The global heuristic evaluates the importance of the plan taking into account the plans proposed by the other agents.

8 CONCLUSION AND FUTURE WORKS

In this paper, we proposed a decentralized extension of A*: the algorithm of finding the shortest path. The decentralized algorithm DEC-A* extends the heuristic function to be the sum of two functions: a local heuristic which estimates the cost of the local fragment of the solution, and a global heuristic which estimates the sum of the costs of the next solution fragment.

Our DEC-A* is complete, optimal and its complexity is less than the complexity of A*.

In the future, we must implement and test DEC-A* to show its effectiveness.

If it is the case, we will see how can we extend DEC-A* to improve its efficacy to solve similar planning problems or planning with incomplete information by extending the algorithms like Incremental A* (Koenig and Likhachev 2002a) (to obtain Incremental DEC-A*), Real-Time Adaptive A* (Sun, Koenig, and Yeoh 2006) (to obtain Real-Time Adaptive DEC-A*), Tree Adaptive A* (Hernández et al. 2011) (to obtain Tree Adaptive DEC-A*).

Finally, we must see how to extend all versions of DEC-A* from one initial and goal states, to multi initial and goal states.

References

- Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *NUMERISCHE MATHEMATIK* 1(1):269–271.
- Falou, M. E.; Bouzid, M.; Mouaddib, A.-I.; and Vidal, T. 2010. A distributed planning approach for web services composition. In *ICWS*, 337–344.
- Hernández, C.; Sun, X.; Koenig, S.; and Meseguer, P. 2011. Tree adaptive A*. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '11, 123–130. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.
- Koenig, S., and Likhachev, M. 2002a. Incremental A*.
- Koenig, S., and Likhachev, M. 2002b. Improved fast replanning for robot navigation in unknown terrain. 968–975.
- Likhachev, M.; Gordon, G.; and Thrun, S. 2004. Ara*: Any-time A* with provable bounds on sub-optimality.
- Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *AAMAS*, 1323–1330.
- P. E. Hart, N. J. N., and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics* SSC-4(2):100–107.
- Shoham, Y., and Leyton-Brown, K. 2009. *Multiagent Systems - Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press.
- Sigaud, O., and Buffet, O., eds. 2010. *Markov Decision Processes and Artificial Intelligence*. ISTE - Wiley. ISBN: 978-1-84821-167-4.
- Stentz, A. 1995. The focussed D* algorithm for real-time replanning. 1652–1659.
- Sun, X.; Koenig, S.; and Yeoh, W. 2006. Real-time adaptive A*. 281–288.