THE DRA^* ALGORITHM

Filippos Gouidis, Theodoros Patkos, Giorgos Flouris and Dimitris Plexousakis

July 2017

Foundation for Research and Technology

Institute of Computer Science

Technical Report No. 468

THE DRA* ALGORITHM

Filippos Gouidis, Theodoros Patkos, Giorgos Flouris and Dimitris Plexousakis

Foundation for Research and Technology
Institute of Computer Science
100 Nikolaou Plastira str
Heraklion, Crete GR - 711 10, Greece
www.ics.forth.gr

July 2017

Technical Report No. 468

Abstract

This technical report introduces the *DRA** algorithm which extends the A^* algorithm and is suited for re-planning, by presenting the basic ideas and concepts behind its implementation and utilization along with a comparison of its performance against A^* in a number of different re-planning scenarios.

Contents

Contents

1 INTRODUCTION

Re-planning is a special case of planning which arises during the deployment of a plan when either a plan being deployed does not longer satisfy certain criteria (usually of time or actions' costs optimality) or some of its pending actions cannot be executed. In that case, a new plan has to be produced, and depending on the way in which this procedure is carried out, the next two categories can be distinguished: **re-planning from scratch** and **plan repairing**. In the former case, all the processed information that was used for the production of the original plan is discarded, whereas, in the latter, a part of the previous computational effort is utilized.

Consequently, applying the latter approach, enables in certain cases, the re-utilization of a part of the already processed data that was used for the original plan, which, in turn, might accelerate the finding of the new plan in comparison to re-planning from scratch. However, in cases where the environment has changed in a significant degree, it is likely that much computational effort will be wasted for the processing of information that is no longer valid, and, hence, plan repairing will result being less efficient than re-planning from scratch in terms of speed.

The re-planning problem does not have just theoretical value, but is also of significant practical use since plan invalidations in many real-world problems are observed frequently due to various reasons. First, a complete knowledge of the environment in realistic scenarios is very often impossible. Besides, even when this is achievable, it results in huge quantities of information, the utilization of which is computationally infeasible.

In addition, neither the agents' behaviour nor the outcome of the actions they execute can be predicted with total accuracy, since the planning agents may break down or perform a task wrongly or partially. Furthermore, an extra factor that frequently hinders the unobtrusive execution of plans, is the dynamic nature of the environments where the latter are deployed. These real-world domains, in contrast to the theoretical models which simulate them, are rarely static. On the contrary, they are susceptible to changes that could render an executing plan sub-optimal or make its full realization impossible. Also, another contingency is that in many domains the original objective for which the plan was produced, may change before its execution is completed.

This line of work *investigates the conditions under which plan repairing is more efficient than re-planning from scratch.* To this end, we focus our attention on A^*algorithm [12, 21, 26, 10], which is one of the most popular and studied algorithms in the field of Artificial Intelligence, and is considered the standard planning algorithm. Specifically, our contribution lies in the development of a novel algorithm, Dynamic Repairing A^* (henceforth *DRA**), which extends A^* in such a way that it can be used for plan repairing. Namely, *DRA** is suited for the repairing of plans in dynamic environments and can address modifications in goal-sets and actions' costs during the execution of a plan, which are two of the most common causes of plan invalidation.

Since many state-of-the-art planning algorithms in a variety of domains are based on A^* the study can provide valuable hints and insights towards the improvement of the existing re-planning methods as well as towards the development of more efficient ones.

Moreover, although it is has been demonstrated in the classic and highly influential work of [20] that *in the worst case* modifying an existing plan is not guaranteed to be more efficient than re-planning from scratch, the goal of a thorough understanding regarding the trade-offs between these two approaches is far from achieved. The current study wishes to explore in more depth this interaction, revealing practically important instances, where repairing is guaranteed to be the optimal choice.

The rest of the paper is organized as follows. In the second section, we describe briefly the A^* algorithm. Next, we discuss related work regarding the re-planning problem. We then present *DRA**. In the fifth section, we continue by presenting our experimental evaluation comparing *DRA** and A^* in standard planning benchmarks. In section 6 we conclude.our conclusions.

2 BACKGROUND

2.1 Notation - Terminology

The objective of classical planning is to find, given an initial world state and a set of goals, a sequence of actions that when executed will fulfill the goal-set. The world is represented by a finite set of states each of which, in turn, consists of a set of propositions. Each proposition corresponds to a certain feature of the world and takes the value *True* or *False* depending on whether this feature holds or not. In addition, there exists a number of actions which can be applied and enable the transition from one state to another. Each action is applicable when certain preconditions are met, it is (usually) associated with a cost and, when executed, produces deterministically some effects.

The most commonly used type of algorithms that are used for planning are the graph search algorithms. In the rest of the section, we describe the basic terminology and notation for this kind of algorithms, since *DRA** belongs in this category. A graph search algorithm builds incrementally a directed graph, usually a tree, starting from the initial state of the problem. Each node of the graph correspond to a different state of the world. Each edge of the graph corresponds to the action that produces the transition from the state, from the corresponding node of which the edge departs, to the state to the corresponding node of which the edge arrives. The weight of the edge corresponds to the cost of the corresponding action.

The **g-value** of a state s_v is equal to the sum of the minimum-cost set of actions that when executed lead from the initial state to s_v. The **h-value** of a state s_v is an estimation of the sum of the minimum-cost set of actions that lead from s_v to a goal-state. The **f-value** of a state is equal to the sum of its g-value and h-value. A **goal_state** is a state that satisfies the problem's goal-set. A goal-set g_1 is called **increased goal-set** of another state g_2 if g_1 is a super-set of g_2, e.g $\forall P \in g_2 \Rightarrow P \in g_1$

A state s_a is a **predecessor state** of another state s_d, if s_a has generated s_d. With the term p-value$_{s_a \rightarrow s_d}$ we denote the resulting g-value of s_d after its generation of another state s_a. Likewise, ac$_{s_a \rightarrow s_d}$ is the corresponding action, that leads from s_a to s_d. Conversely, if s_a is the predecessor state of s_d, then s_d is the **successor state** or **child state** of s_a. The **branching factor** of a state is the number of its successor states. By averaging the branching factors of all the states of a graph, the **average branching factor** is obtained.

If s_1 is predecessor state of s_2, s_2 is predecessor state of state s_3,... and s_{n-1} is predecessor state of s_n, then the sequence of actions $ac_{s_1 \rightarrow s_2}, ac_{s_2 \rightarrow s_3}, \ldots, ac_{s_{n-1} \rightarrow s_n}$ is called **path**$_{\boldsymbol{s_1, s_2, \ldots, s_{n-1}, s_n}}$. The cost of a path p is equal to the g-value of its final state, i.e. it is equal to the sum of its actions' costs.

A state s_p is a **parent state** of another state s_d, if s_p has generated s_d and p-value$_{s_p \rightarrow s_d}$ $\leq$ p-value$_{s' \rightarrow s_d}$ for every other state s' that is predecessor state of s_d. Thus, the g-value of a state s_d is equal to p-value$_{s_p \rightarrow s_d}$, where s_p is the parent state of s_d. The action with which a state s_d is generated by another state s_a, is called **generating action**$_{s_a \rightarrow s_d}$. The generating action of the parent state is called **p-generating action**.

A path p$_{s_1,\ldots,s_n}$ between two states s_1 and s_n is called optimal, if any other path p'$_{s_1,\ldots,s_n}$ between s_1 and s_n has at least the same cost. We use the term **ris** (abbreviation for replanning initial state) to denote the state from where the replanning begins. A state s_v is called **valid** if there is at least one path from *ris* to s_v; otherwise it is called **invalid**. A state s_v is called **uninformed** if it is derived from a previous planning procedure and the informing procedure has not been performed upon s_v; otherwise it is called **informed**.

2.2 A* Algorithm

A^* [9, 23] is one of the most popular algorithms of Artificial Intelligence [12, 10, 21], with some of its most common uses including graph traversal and path-finding. Its key idea is the utilization of a heuristic value, that "guides" the search. As a result, its performance depends on the quality of the function that generates the heuristic values.

A^* can be implemented in two different ways: a) using a **tree search** or b) using a **graph search**. Typically, in both cases, two auxiliary collections are utilized during the execution of the algorithm: a priority queue, called **open list**, containing the states candidate for expansion, and a set, referred to as **closed list**, containing the already expanded states.

At each step of the tree search variation, the state of the open list having the lowest f-value is removed from it. The state is examined for satisfying the goal-set in which case the search stops and the corresponding plan is extracted. Otherwise, the state is expanded by generating all its successor states which are added in the open list, while the expanded state is added in the close list. By following the parent-state pointers from the goal-satisfying state to the initial state, we have the reverse order of the successive states and the corresponding actions of the plan.

If the h-values that are used are consistent[1] then this variation of the algorithm is guaranteed to find an optimal solution, if one exists, and, moreover, not to generate more states than any other algorithm that uses the same h-values[2].

Graph search differs from tree search in two points. First, each time a state is generated it is examined for being contained in the closed and open list respectively. If it is not contained in neither of the lists, the same steps as in the case of tree search are followed. If it is already in the closed list, then its current f-value is compared to its old f-value, e.g. the one with which it was inserted in the closed list. If the new f-value is smaller, the state is removed from the closed list and re-inserted in the open list with the new f-value.

Moreover, the algorithm in this variation does not stop when a plan has been found, but it continues until there is no state in the open list with an f-value that is smaller than the cost of the plan. In this case, it is not required that the h-values are consistent, but it suffices to be admissible, i.e. not to be greater than the cost of the optimal path from the corresponding state to a goal-state.

3 RELATED WORK

3.1 Plan Repairing Based on A*

Over the last years, a significant number of A^*-inspired plan repairing[3] algorithms has been developed, with the majority of them tailored to single-agent robotics problems. These algorithms fall, typically, into two main categories w.r.t their capacities for plan-repairing: a) algorithms that are specialized in addressing modifications of the original goal-set [24, 13, 17] [8] and b) algorithms that are specialized in addressing changes of the actions costs[15, 28, 14]. Finally, there are few other algorithms that can cope with both changes [25, 26, 27].

In general, the efficiency of these algorithms derives from the exploitation of the geometrical properties of the terrain where the agent is situated, since in some single-agent settings, such as navigation or moving-target search, the search tree can be mapped to the problem terrain. However, this mapping cannot be realized in many single-agent settings or in a multi-agent environment and, as a consequence, these algorithms are not applicable for problems of this type.

[1]The h-value of a state s_N is consistent, if for every state s_M that can be generated from s_N, the estimated cost of reaching a goal-state from s_N is not greater than the cost of getting to s_M from s_N plus the estimated cost of reaching a goal-state from s_M

[2]In case of algorithms that use identical h-values, the same tie-breaker criterion for states with same f-values is required.

[3]Typically, these algorithms are referred to as re-planning algorithms. However, we consider that this term might be misleading, since it can be confused with re-planning from scratch. Therefore, we use the term plan repairing instead.

Two of the most influential algorithms of the first category are, Focused D* [24], and D*-Lite [13]. Both of them utilized a backwards-directed search from the goal state to the current state, saving, this way, information, which allows fast plan production when changes in the environment occur. D*-Lite has been further extended since, with some of its extensions having been used in the navigation algorithms for a Mars Rover [4], and in the DARPA Urban challenge competition [16].

The Generalized Adaptive A* (GAA*) is presented in [25]. GAA* learns h-values in order to make them more informed and can be utilized for moving target search in terrains where the action costs of the agent can change between searches. An extension of GAA* that is close to our work, is MP-GAA* [11], where some of the best paths for some nodes of the search graph are stored. More recently, there have been implemented Generalized Fringe-Retrieving A^* [26] and Moving Target D*-Lite [27] which, in the same way as GAA^* can address both goal-set modifications and actions costs changes.

Finally, for each of the algorithms a further distinction can be made depending on whether the optimality of the plan is the main concern, or the production of a new plan within a certain time window after the invalidation of the original plan. In these cases, any plan with a cost which does not transcend a certain threshold w.r.t the cost of the optimal plan is considered a valid solution to the corresponding problem. From the algorithms described in this section, [24, 17, 14, 16, 8] hold this real-time property.

3.2 Other Plan Repairing Algorithms

Komenda et al. (2014) presented a recent work that investigates the recovery after plan failures in multi-agent environments where the objective of plan repairing is the minimization of the exchange of messages between agents and not the plan optimality. A similar work is [19] where a multi-agent decentralized approach to plan repairing is presented. In this case, each agent of the system is responsible for controlling the actions it executes, and for independently repairing its own plan when it detects an action failure, by following a local strategy. As in the previous case, plan optimality is not a central issue.

In [5] the concept of plan stability is introduced. The authors of this work, argue that plan stability is an important property for many planning problems and present a plan repairing algorithm having as first priority the minimal perturbation of the original plan. A similar approach is [2], where the main objective of the plan repairing algorithm is the minimal perturbation of the original plan.

The notion of bookings and commitments are used in [29]. In this case, the plan repairing procedure is considered a sequence of refinement and un-refinement steps, which aim at producing a new plan that does not violate the agents' original bookings and commitments. Again, the plan optimality is not of central importance. Likewise, [3] present a theoretical analysis of the way in which the agents' commitments are linked to the plan repairing procedure.

Finally, [7] are concerned with plan adaptation which can be regarded as an almost identical technique to plan repairing. The planning system, in this case, utilizes specialized heuristic search techniques in order to solve the plan adaptation tasks through the repairing of certain portions of the original plan. The procedure for the plan adaptation is incremental: first, a sub-optimal plan is found and, then, if possible, lest costly solutions are sought.

4 DYNAMIC REPAIRING A*

In this paper, we propose *DRA** an extension of *A**, suited for the repairing of sequential plans. *DRA** can address two types of changes in the environment: a) goal-set modifications and b) actions' costs alterations. *DRA** is based on the graph search variation of *A**, utilizing the same search strategy: the selection, testing and expansion of a state at each step and the utilization of a heuristic value to guide the whole procedure.

Its novelty is that a new search tree is not created from scratch as in *A**. Instead, in the beginning of the algorithm (i.e., at repairing time) the initial search tree is retrieved and used for the subsequent search. As with the case of graph search *A**, the utilization of admissible h-values is required in order for the solutions returned to be optimal. The corresponding pseudo-code is presented in pages 9-12.

Theorem 1. *DRA** *is sound and complete for repairing scenarios of goal-set modifications or actions costs changes if the h-values that are used are admissible*[4].

4.1 Comparing DRA* with A*

DRA* differs from *A** in the following ways:

- **re-Generation of a state**: While in the case of *A** only the pointer to the parent state is kept, in the case of *DRA** pointers to every predecessor state of a given state are stored.
- **Informed and valid states**: A state is checked for being informed before it is tested for satisfying the goal set. If it is found to be uninformed, a special procedure is followed in order to determine if the state is valid or invalid. In the former case, *DRA** continues in the same way as *A**, whereas, in the latter, the current state is discarded and the search continues with the selection of a new state from the open list. By default, when the algorithm begins, all the states of the search tree are considered uninformed except of the new initial state which is set as informed and valid.
- **Informing procedure**: The informing procedure can be achieved in two different ways: **fully** or **lazily**. The former is applied in cases when there exists actions with decreased costs, whereas the latter is applied in the other cases. The procedure for the full informing is the following. First, the state being examined is marked as pending and, consequently, all its predecessor states are examined for being informed. For any predecessor state found not to be informed and not to be pending, the procedure of full informing is followed. If a predecessor state is pending, then the value of a spacial variable that keeps track of the number of the uniformed predecessor states

[4]The proof is presented in the appendix.

increases by 1, while the predecessor state being informed is added to a special list of the pending state, called successors list. This allows for the states contained in successors list to be informed, when the informing of the pending state finishes. If a valid predecessor states is found, the p-value resulting from it is re-calculated and compared with the state's g-value and if found smaller, then this predecessor state is set as the state's parent.

After the examination of the predecessor states finishes, if the state has any valid predecessor, it is marked as informed and valid. Moreover, if the number of its partially informed predecessor states is equal to zero, then the states contained in its successors list are informed . Namely, for each successor state, its g-value is compared to the p-value of the state, and if found greater, then the parent state and its parent action and g-value are updated accordingly. Next, the successor state's number of partially informed parent states is reduced by one.

If it is equal to zero, then the successor state informs with the same procedure the states contained in its own successors list. In case no valid predecessor states have been found, if the number of the state's partially informed predecessor states is equal to zero, then the state is marked as informed and invalid and, next, the predecessor states contained in its successors list are updated. Finally, the state is reset from pending.

The lazy informing is carried out in the same way as the full, with the exception that the procedure stops if a valid predecessor state has been found and the next predecessor state that is to be examined does not have a smaller p-value. The examination of the predecessor states follvalidows their sorting order. That is, it begins with predecessor state having the lowest p-value, i.e. the parent-state, and continues with the one having the second lowest and so forth. Note that in this case, some of the p-values of a state might not be correct and some of its parent states might not have been informed, without this affecting the correctness of the algorithm.

- **Storing of initial closed and open list**: When the search for the plan finishes, the closed and open lists are stored, so that they can be used in case of re-planning. Before the open list is saved, the last removed state is re-inserted in it. Subsequently, when the algorithm is executed, the previously save lists are retrieved and used.

- **Traversal of the closed list**: If the new goal set is the same as the original goal set, the initial closed list is searched for containing solutions before the main part of the algorithm begins. During this traversal, each state is examined. The ones satisfying the new goal-set are lazily informed, when uninformed. In case one or more valid states satisfying the new goal-set have been found, the one having the lowest g-value is returned as solution and the algorithm terminates.

- **Validation of the open list**: In cases where the new goal set is not a superset of the original goal set the open list is validated before the main search starts. Namely, every state is informed, fully if there exists actions with decreased costs and lazily

Algorithm 1: Dynamic Repairing A*

```
input : New Initial State, Previous Closed List, Previous Open List, Original Goal set, New goal set
output: The optimal plan for the new goal set
$plan \longleftarrow NULL$
mark newInitialState as valid and informed
$CLOSED \longleftarrow previousCLOSED$
if $originalGoalSet = newGoalSet$ then
    $plan \longleftarrow searchCloseList(CLOSED, newGoalSet)$
if $plan \neq NULL$ & $\nexists$ $action\ with\ decreased\ costs$ then
    return plan
$OPEN \longleftarrow previousOPEN$
if $newGoalSet\ is\ not\ superset\ of\ originalGoalSet$ then
    validateOpenList(OPEN,originalGoalSet,newGoalSet)
while $OPEN\ is\ not\ empty$ do
    $currentState \longleftarrow OPEN.poll()$
    if $currentState\ satisfies\ newGoalSet$ then
        $plan \longleftarrow ExtractPlan(currentState)$
        break
    foreach $applicable\ action\ ac\ of\ currentState$ do
        $successorState \longleftarrow currentState.apply(ac)$
        $pVal \longleftarrow currentState.gValue + ac.cost$
        if $successorState \notin OPEN\ and\ \notin CLOSED$ then
            OPEN.add(successorState)
        else
            if $successorState\ is\ not\ informed$ then
                lazy_inform(successorState)
            if $successorState\ is\ not\ valid$ then
                OPEN.add(successorState)
            else
                if $pVal < successorState.gValue$ then
                    if $OPEN \ni successorState$ then
                        OPEN.remove(successorState)
                    if $CLOSED \ni successorStatet$ then
                        CLOSED.remove(successorState)
                    OPEN.add(successorState)
                else
                    successorState.predQueue.add(currentState )
        CLOSED.add(currentState )
    end
end
OPEN.add(plan.currentState )
$previousOPEN \longleftarrow OPEN$
$previousCLOSED \longleftarrow CLOSED$
return plan
```

otherwise, and, if it is valid, its h-value is re-calculated and it is re-inserted in the open list with its newly updated f-value.

4.2 Applying DRA* in a Toy Example

Let for a domain $\mathcal{D}$, S_{st} and S_p be the corresponding set of states and propositions for which it holds that $S_{st} = \{A,B,C,D,E,F,G,H,I,J\}$ and $S_p = \{P_0,P_1,P_2,P_3,P_4\}$ respectively. Table 1 presents the successor states and the propositions of each state. The number between the parentheses after a successor state is the cost of the corresponding generating action. Let the initial state of the problem be $I = \{A\}$ and the goal state be $G = \{P_2,P_3\}$. The optimal plan is $P = \{ac_{A\to C}, ac_{C\to G}\}$ and its cost is 5. The final search tree is presented in figure 1, with the three numbers at the right of each node being its g-value, its h-value and its f-value respectively. The states of the closed list are depicted in black color and

Algorithm 2: Validation of the Open List

```
input : Open List, Original Goal Set and New Goal Set
1  $newOpenList \longleftarrow new\ Priority\ Queue()$
2  foreach $state\ in\ OPEN$ do
3      if $state\ is\ not\ Informed$ then
4          if $\nexists\ action\ with\ decreased\ costs$ then
5              lazy_inform($state$)
6          else
7              full_infrom($state$)
8          if $state\ is\ Valid$ then
9              state.$updatefValue()$
10             newOpenList.add($state$)
11 end
12 $OPEN \longleftarrow newOpenList$
```

Algorithm 3: Traversal of the Closed List

```
input : Closed List and New Goal Set
output: A plan
1  $plan \longleftarrow null$
2  $cost = \infty$
3  foreach $State\ state\ in\ CLOSED$ do
4      if $state\ satisfies\ goalSet$ then
5          if $state\ is\ not\ Informed$ then
6              lazy_inform($state$);
7          if $state\ is\ Valid$ then
8              if $state.gValue < cost$ then
9                  $plan \longleftarrow ExtractPlan$(state)
10                 $cost$ =state.$gValue$
11 end
12 return plan
```

the states of the open list in blue color respectively. The parent pointers are depicted in continuous black line and the other ancestor states pointer in dashed line. In total, 5 states are expanded and 9 are generated. Notice that the only differences between this tree and the one that would have been produced from a A^* search are the extra pointers to every predecessor state.

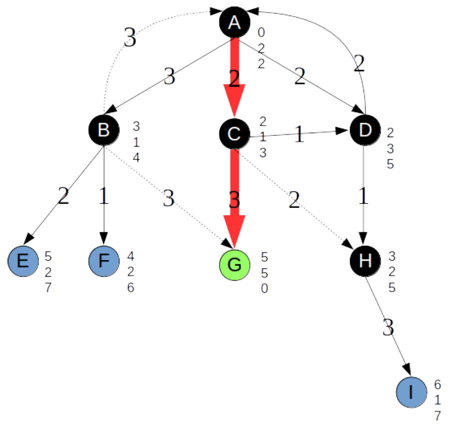

Figure 1: The final search tree for the initial problem after the execution of *DRA**

Supposing now that after first action of the aforementioned plan has been executed in addition to the satisfaction of the original propositions, the satisfaction of the proposition

Algorithm 4: Full informing

```
input : A state stateInf
set $stateInf$ as pending
$parentState \longleftarrow stateInf.getParent()$
$stateInf.gValue = \propto$
$nonInformedPredecessors = 0$
if $parentState$ not informed & not pending then
    full_inform(parentState)
if $parentState$ is Valid then
    $stateInf.gValue = parentState.gValue + action.cost$
if $parentState$ is pending then
    $nonInformedPredecessors{+}{+}$
    parentState.StatesList.add(stateInf )
if $parentState.nonInformedPredecessors > 0$ then
    $nonInformedPredecessors{+}{+}$
    parentState.StatesList.add(stateInf )
foreach predState in stateInf.predQueue do
    if predState not informed & not pending then
        full_inform(predState )
    if $predState$ is pending then
        $nonInformedPredecessors{+}{+}$
        predState .StatesList.add(stateInf )
    if $predState.nonInformedPredecessors > 0$ then
        $nonInformedPredecessors{+}{+}$
        predState .StatesList.add(stateInf )
    if $predState$ is Valid then
        if $predState.pValue < stateInf.gValue$ then
            $stateInf.lgvParent \longleftarrow predState$
            $stateInf.gValue \longleftarrow predState.pValue$
end
if $stateInf.gValue \neq \propto$ then
    mark stateInf as valid and informed
else
    mark stateInf as invalid and informed
updateSuccessorStates(stateInf )
reset stateInf from pending
```

Algorithm 5: Lazy informing

```
input : A state stateInf
1  set stateInf as pending
2  $parentState \longleftarrow stateInf.getParent()$
3  $stateInf.gValue = \infty$
4  $nonInformedPredecessors = 0$
5  if parentState not informed & not pending then
6      full_inform(parentState)
7  if parentState is Valid then
8      $stateInf.gValue = parentState.gValue + action.cost$
9  if parentState is pending then
10     $nonInformedPredecessors{+}{+}$
11     parentState.StatesList.add(stateInf )
12 if $parentState.nonInformedPredecessors > 0$ then
13     $nonInformedPredecessors{+}{+}$
14     parentState.StatesList.add(stateInf )
15 foreach predState in stateInf.predQueue do
16     if $predState.pValue \geq stateInf.gValue$ then
17         break
18     if predState not informed & not pending then
19         full_inform(predState )
20     if predState is pending then
21         $nonInformedPredecessors{+}{+}$
22         predState .StatesList.add(stateInf )
23     if $predState.nonInformedPredecessors > 0$ then
24         $nonInformedPredecessors{+}{+}$
25         predState .StatesList.add(stateInf )
26     if predState is Valid then
27         if $predState.pValue < stateInf.gValue$ then
28             $stateInf.lgvParent \longleftarrow predState$
29             $stateInf.gValue \longleftarrow predState.pValue$
30 end
31 if $stateInf.gValue \neq \infty$ then
32     mark stateInf as valid and informed
33 else
34     mark stateInf as invalid and informed
35 updateSuccessorStates(stateInf )
36 reset stateInf from pending
```

Algorithm 6: Successor States Update

```
input : A state stateUpd
1  if stateUpd is informed and valid then
2      foreach successor_state in stateUpd.StatesList do
3          $ac \longleftarrow successor\_state.generatingAction$
4          $pVal \longleftarrow stateUpd.gValue + ac.cost$
5          successor_state.nonInformedPredecessors−−
6          if $pVal < successor\_state.gValue$ then
7              $successor\_state.gValue = pVal$
8              $genState.parent \longleftarrow stateUpd$
9              if successor_state.nonInformedPredecessors=0 then
10                 marked successor_state as informed and valid
11                 successor_state.updateSuccessorStates()
12     end
13 else
14     foreach successor_state in stateUpd.StatesList do
15         successor_state.nonInformedPredecessors−−
16         if successor_state.nonInformedPredecessors=0 then
17             marked successor_state as informed and invalid
18             successor_state.updateSuccessorStates()
19     end
```

Table 1: States Table

State	Successor States	Propositions
A	B(3), C(2), D(2)	P_0
B	E(2), A(3), F(1), G(3)	P_0, P_1
C	G(2), H(2), D(1)	P_2
D	H(1), A(2)	P_0,P_2
E	$\emptyset$	P_1, P_3
F	J(2)	P_2, P_4
G	J(3)	P_2, P_3
H	I(2)	P_2, P_4
I	$\emptyset$	P_3, P_4
J	$\emptyset$	$P_2.P_3, P_4$

P_4 is also required. Therefore, the new initial state is $I_{new} = \{C\}$ and the new goal state is $G_{new} = \{P_2, P_3, P_4\}$.

DRA^* is executed in the following way. State G is removed from the open list. It is informed lazily and, then, being valid, expanded and added to the closed list. State J is generated and inserted in the open list. State F is removed from the open list. It is informed (States B, A and D are informed lazily during the procedure). Although state F is valid, its updated f-value is greater than the f-value of the head of the open list (state J), and, therefore, it is re-inserted in the open list. State J is removed from the open list and the algorithm terminates. The corresponding plan is $P = \{acc_{C\to G}, acc_{G\to J}\}$ and its cost is 6.

The final search trees for DRA^* is presented in figure 2 (a state with red-colored values is informed; otherwise it is not informed). Notice that 2 states of the open list and one state of the closed list are not informed. In total, 5 states are informed, 1 is re-inserted in the open list, 1 is expanded and 1 is generated. The final search trees for the case of A^* is presented in figure 3. In this case, 4 states are expanded and 7 states are generated.

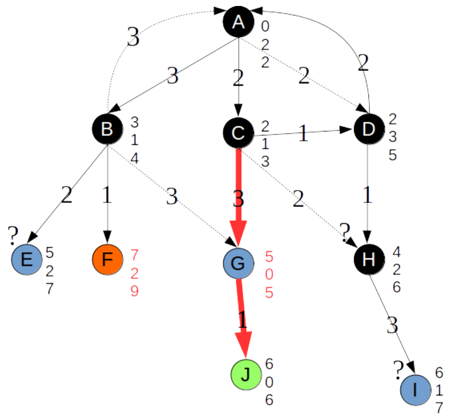

Figure 2: The final search tree of DRA^* for example 1

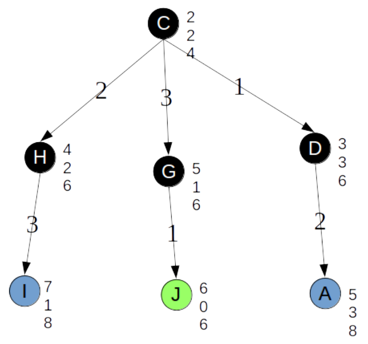

Figure 3: The final search tree of A^* for example 1

5 EXPERIMENTAL EVALUATION

In order to compare the performance of DRA^* against A^* in terms of speed, we devised four different re-planning scenarios, where we compared the ratio of the runtime of the two algorithms by varying the following characteristics of the scenarios:

- The percentage of the original plan that was already executed at the time when the need for replanning occurred (Scenarios 1,2,3 and 4);
- The percentage of the modification of the original goal-set (Scenarios 1 and 2);
- The percentage of the actions whose costs decreased (Scenario 3);
- The percentage of the actions whose costs increased (Scenario 4).

The experiments focus on time performance; memory requirements are not measured, because both A^* and DRA^* exhibit a linear complexity in the number of states in the state space.

5.1 Experimental Setup

The structure of the experiments is the same in every case. First, a plan is produced for the initial conditions of the problem, i.e. initial state, goal-set and actions' costs. Next, a parameter of the environment, according to the type of the experiment, is modified: in scenarios 1 and 2 the goal-set, and in scenarios 3 and 4 the costs of some actions. Finally, a new plan is produced for the modified conditions using both A^* (replanning from scratch) and DRA^* (repairing). The new initial state of the re-planning problems is a randomly-selected state of the initial plan. The changes for each scenario are the following:

- Scenario 1
  The new goal set is produced by the removal of k goals from the initial goal set consisted of n goals, and the insertion of m goals in it respectively, where $k \leq n$.

Table 2: Experiments of scenario 1

Experiment	Problem	Size of Initial Goal Set	Number of Removed Goals	Number of Added Goals	Average Branching Factor	Number of Conducted Experiments
1.1	Blocks 92	6	1	1	4.61	5
1.2	Depots 1935	5	1	1	8.77	5
1.3	Gripper x6	12	2	3	4.62	5
1.4	Logistics 62	5	1	1	8.33	5
1.5	Logistics 63	5	1	1	8.44	5

- Scenario 2
  The new goal set is produced by the addition of *k* goals in the original goal-set. .

- Scenario 3
  A *p%* percentage of the actions costs are decreased, none of which belongs to the initial plan. The maximum decrease for an action cost is a 90% of its initial cost.

- Scenario 4
  A p% percentage of the actions costs are increased. *q%* of the actions with increased costs belongs to the initial plan. The maximum increase for an action cost is a 200% of its initial cost.

The specifications of the scenarios are shown in Tables 2-5.

The benchmarks that were used for the evaluation are: Blocks, Depots, Gripper, Logistics, Miconic and Transports which derive from the 3^{rd}, 4^{th} and 8^{th} International Planning Competitions [6, 18, 1]. In addition, since in the majority of the planning domains the actions costs are uniform, we created two variations of the domains Logistics and Depots, Logistics-cost and Depots-cost respectively, with actions of varied costs.

The experiments were conducted on a 64-bit Ubuntu Workstation with two 8-core® Xeon® CPU E5-2630 processors running at a 2.30GHz server with 384 GB RAM, from which 10 GB were allocated for each experiment.

5.2 Experimental Results

The experimental results, presented in Table 6, suggest that *DRA** outperforms A^* in most of the goal set modification cases, provided that the next conditions are met. First, the percentage of the original plan that has been already executed, should not be greater than 50%. Moreover, the change in the goal-set should not be greater than 20% to 50%. The corresponding thresholds, for the previous two parameters, below which *DRA** performs better, depend on the average branching factor of the re-planning problem, with average higher branching factors corresponding to thresholds of lower values. Likewise, in the cases of modified actions costs, *DRA** outperforms A^* always.

Furthermore, we can make the following observations regarding the performance of *DRA** compared to A^*:

1. As the percentage of the executed plan decreases, the performance is improved.

Table 3: Experiments of scenario 2

Experiment	Problem	Size of Initial Goal Set	Number of Added Goals	Average Branching Factor	Number of Conducted Experiments
2.1	Blocks 91	6	2	4.92	228
2.2	Blocks 91	7	1	4.95	28
2.3	Blocks 92	6	2	4.62	28
2.4	Blocks 92	7	1	4.69	8
2.5	Logistics 61	4	2	8.38	15
2.6	Logistics 61	5	1	8.46	6
2.7	Logistics 62	4	2	8.51	15
2.8	Logistics 62	5	1	8.49	6
2.9	Depot 1345	3	2	10.88	10
2.10	Depot 1345	4	1	10.84	5
2.11	Depot 1935	3	3	4.62	20
2.12	Depot 1935	4	2	4.65	15
2.13	Gripper x5	8	4	4.38	40
2.14	Gripper x6	9	5	4.35	40
2.15	Gripper x6	11	3	4.66	40
2.16	Miconic 10	7	3	19.83	10
2.17	Miconic 10	9	1	19.97	10
2.18	Miconic 11	9	2	21.72	10
2.19	Miconic 12	8	4	23.76	10

Table 4: Experiments of scenario 3

Experiment	Problem	Percentage of Decreased Actions Costs	Max Percentage of Plan's Decreased Actions Costs	Average Branching Factor	Number of Conducted Experiments
3.1a-3.1b-3.1c	Transport 2533	5-25-50	90	8.93	10
3.2a-3.2b-3.2c	Depots-cost 1935	5-25-50	90	4.62	10
3.3a-3.3b-3.3c	Logistic-cost 63	5-25-50	90	8.23	10

Table 5: Experiments of scenario 4

Experiment	Problem	Percentage of Increased Actions Costs	Max Percentage of Plan's Increased Actions Costs	Average Branching Factor	Number of Conducted Experiments
4.1a-4.1b-4.1c	Transport 2533	5-25-50	200	8.94	10
4.2a-4.2b-4.2c	Depots-cost 1935	5-25-50	200	4.57	10
4.3a-4.3b-4.3c	Logistic-cost 63	5-25-50	200	8.16	10

Table 6: Mean value of the ratio of the runtime of DRA^* to A^* for scenarios 1-4

		Percentage of Executed Plan								
		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Experiments	1.1	0.85±0.10	0.83±0.21	0.78±0.14	0.76±0.19	0.97±0.21	0.94±0.23	1.23±0.22	1.12±0.31	0.63±0.28
	1.2	0.44±0.10	0.44±0.05	0.61±0.05	1.33±0.05	1.78±0.35	2.58±0.31	2.95±0.32	3.21±0.41	4.04±0.54
	1.3	0.22±0.05	0.23±0.03	0.21±0.06	0.24±0.03	1.21±0.53	1.86±1.01	2.30±1.62	2.92±1.57	3.36±2.01
	1.4	0.41±0.07	0.29±0.06	0.58±0.09	1.36±0.16	1.28±0.14	2.70±0.37	4.82±0.45	5.67±0.64	7.03±0.82
	1.5	0.23±0.03	0.32±0.13	0.53±0.14	1.17±0.21	1.52±0.94	1.68±0.00	2.92±1.34	4.20±1.76	6.04±2.54
	2.1	0.85±0.09	0.84±0.11	0.81±0.12	0.80±0.13	0.88±0.13	0.94±0.12	0.99±0.06	0.98±0.08	0.92±0.30
	2.2	0.65±0.06	0.62±0.06	0.59±0.06	0.48±0.13	0.77±0.15	1.07±0.17	0.93±0.08	1.08±0.15	0.85±0.08
	2.3	0.85±0.14	0.84±0.15	0.78±0.13	0.78±0.19	0.77±0.14	1.47±1.23	1.01±0.33	2.85±2.58	1.72±1.53
	2.4	0.67±0.13	0.60±0.17	0.59±0.17	0.60±0.27	0.69±0.18	2.53±1.89	1.29±0.71	1.66±1.42	1.65±1.15
	2.5	0.96±0.16	0.96±0.19	0.90±0.18	0.86±0.13	0.99±0.12	1.10±0.25	1.89±1.62	2.47±1.32	3.19±1.66
	2.6	0.60±0.26	0.59±0.24	0.70±0.13	0.75±0.10	2.06±2.14	2.99±2.36	3.46±1.76	4.50±1.93	6.10±2.44
	2.7	0.99±0.12	1.00±0.10	0.96±0.13	0.93±0.15	0.96±0.10	1.03±0.09	1.39±0.46	2.59±1.53	3.33±2.23
	2.8	0.58±0.22	0.55±0.20	0.57±0.15	0.65±0.21	0.85±0.14	1.64±0.82	3.54±0.88	4.73±1.35*n*	5.74±1.66
	2.9	0.87±0.15	0.85±0.17	0.82±0.14	0.86±0.16	0.82±0.13	1.01±0.07	2.16±1.22	3.63±2.27	4.27±2.06
	2.10	0.57±0.02	0.51±0.09	0.49±0.06	0.45±0.17	1.09±0.81	1.09±0.89	1.50±1.23	2.42±1.76	3.45±2.03
	2.11	0.83±0.04	0.85±0.06	0.79±0.08	0.83±0.07	0.84±0.08	0.82±0.04	1.01±0.17	2.64±1.18	3.41±0.34
	2.12	0.65±0.19	0.64±0.21	0.60±0.20	0.59±0.18	0.71±0.16	1.40±1.49	2.78±1.64	3.65±2.63	3.87±1.96
	2.13	0.48±0.04	0.46±0.04	0.46±0.03	0.45±0.04	0.43±0.04	0.44±0.05	0.49±0.06	0.74±0.15	1.26±0.29
	2.14	0.80±0.14	0.78±0.09	0.73±0.11	0.69±0.08	0.71±0.09	0.77±0.07	0.87±0.08	1.37±0.38	2.75±1.83
	2.15	0.32±0.02	0.30±0.01	0.28±0.01	0.29±0.02	0.26±0.05	0.56±0.11	1.10±0.17	3.60±0.19	4.56±0.25
	2.16	0.53±0.15	0.72±0.10	1.44±0.35	2.11±0.86	4.01±0.70	5.10±1.46	6.75±0.00	8.43±3.21	10.03±3.42
	2.17	0.21±0.05	0.54±0.15	0.95±0.05	2.08±0.57	4.85±1.76	5.72±1.87	6.44±2.22	8.32±2.42	9.84±3.27
	2.18	0.17±0.03	0.63±0.07	1.03±0.34	3.70±0.60	5.30±1.71	6.90±2.41	7.72±2.74	9.24±3.03	10.11±2.87
	2.19	0.62±0.13	1.13±0.60	1.56±0.99	2.68±1.04	4.43±1.44	6.10±2.54	8.68±3.04	9.52±3.04	11.30±3.32
	3.1a	0.43±0.13	0.68±0.17	0.73±0.14	0.59±0.16	0.64±0.17	0.83±0.21	0.72±0.21	0.85±0.23	0.81±0.27
	3.1b	0.50±0.12	0.54±0.14	0.59±0.14	0.62±0.21	0.54±0.22	0.69±0.25	0.69±0.22	0.92±0.26	0.78±0.25
	3.1c	0.49±0.15	0.55±0.16	0.53±0.19	0.70±0.22	0.64±0.24	0.62±0.22	0.71±0.25	0.90±0.25	0.84±0.26
	3.2a	0.17±0.03	0.18±0.04	0.24±0.04	0.22±0.05	0.24±0.06	0.21±0.07	0.20±0.06	0.24±0.08	0.30±0.07
	3.2b	0.25±0.05	0.29±0.05	0.23±0.04	0.29±0.06	0.34±0.07	0.31±0.09	0.30±0.09	0.35±0.10	0.32±0.11
	3.2c	0.26±0.07	0.32±0.06	0.28±0.06	0.29±0.07	0.37±0.04	0.39±0.11	0.35±0.12	0.36±0.11	0.39±0.12
	3.3a	0.41±0.08	0.44±0.07	0.56±0.11	0.54±0.13	0.46±0.14	0.48±0.18	0.52±0.19	0.61±0.21	0.59±0.20
	3.3b	0.45±0.08	0.46±0.08	0.48±0.12	0.50±0.15	0.54±0.18	0.52±0.17	0.61±0.22	0.60±0.24	0.59±0.22
	3.3c	0.49±0.10	0.56±0.12	0.52±0.14	0.62±0.16	0.55±0.17	0.55±0.19	0.69±0.24	0.65±0.23	0.68±0.25
	4.1a	0.21±0.03	0.23±0.07	0.19±0.34	0.25±0.07	0.30±0.10	0.65±0.15	0.77±0.14	0.64±0.13	0.80±0.17
	4.1b	0.48±0.13	0.62±0.14	0.75±0.19	0.79±0.18	0.87±0.19	0.79±0.21	0.77±0.20	0.73±0.24	0.85±0.23
	4.1c	0.54±0.13	0.50±0.16	0.65±0.19	0.68±0.24	0.73±0.24	0.70±0.25	1.00±0.28	0.83±0.26	0.84±0.22
	4.2a	0.10±0.03	0.08±0.04	0.12±0.04	0.10±0.06	0.24±0.11	0.35±0.14	0.27±0.13	0.28±0.13	0.30±0.12
	4.2b	0.09±0.04	0.13±0.05	0.13±0.06	0.18±0.07	0.26±0.10	0.25±0.12	0.27±0.11	0.37±0.14	0.33±0.13
	4.2c	0.21±0.10	0.27±0.13	0.25±0.11	0.35±0.17	0.30±0.15	0.29±0.16	0.37±0.16	0.35±0.15	0.39±0.13
	4.3a	0.14±0.05	0.24±0.09	0.39±0.14	0.45±0.16	0.37±0.17	0.34±0.14	0.28±0.14	0.43±0.13	0.36±0.18
	4.3b	0.26±0.05	0.33±0.11	0.38±0.12	0.48±0.13	0.41±0.15	0.44±0.15	0.42±0.16	0.39±0.17	0.45±0.15
	4.3c	0.41±0.08	0.54±0.15	0.43±0.16	0.67±0.15	0.54±0.18	0.70±0.19	0.75±0.21	0.80±0.22	0.74±0.21

2. As the percentage of the modified goal-set decreases, the performance of is improved.

3. As the average branching factor, i.e. the average number of predecessor states that a state has, decreases, the performance is improved.

4. The performance does not vary significantly as the percentage of actions with decreased costs increases.

5. The performance does not vary significantly as the percentage of actions with increased costs increases.

6. For a given problem instance, *DRA** performs better in increases of the goal-set than in general modifications of the goal-set.

7. For a given problem instance, *DRA** performs better in cases of increased actions costs than of decreased actions costs.

We consider that the previous findings can be explained by the following reasons. First, *DRA** expands at most the same number of states as A^*, since a part of the search tree with which the search begins, is already constructed. Moreover, during *DRA** execution, the procedures of states informing, open list validation and closed list traversal, which are absent from A^*, might take place. Therefore, it can be concluded, that the trade-off between the previous two factors determines *DRA** performance against A^*.

Regarding the first finding, it can be due to the fact that as the percentage of the executed plan increases, the new root of the search tree recedes further from the root of the original search tree, which, as a result, has one of the following two outcomes: a larger part of the search tree leaves would either become invalid or would have its f-values increased. In either case, time is consumed for the informing of states that do not affect the search.

Likewise, the fact that the traversal of the closed list and the validation of the open list, is not carried out in the case of an increased goal-set seems to explain the better performance of *DRA** in such cases in comparison to the general case of handling modified goal-sets (observation 6). A similar line of reasoning can be applied in the case of modified actions costs (observation 7). In the case of decreased costs, the open list is validated. Furthermore, the informing of the states is full, whereas, in the case of increased costs, the lazy informing is utilized, which, at worst case, requires the same time. Findings 4 and 5 can be ascribed to the fact that greater percentages of modified actions costs does not affect the execution of *DRA**.

Finally, the deterioration of *DRA** performance with higher average branching factors (observation 3) can be attributed to the greater time that is necessary for the informing procedure. Namely, large average branching factors correspond to a large average number of predecessor states, which results in a greater number of examined ancestor states during the informing procedure.

6 Conclusions

In this work we presented a novel plan repairing algorithm, *DRA**, that extends one of the most popular and studied planning algorithms, *A**, by addressing modifications in the goal-set and in the actions costs. *DRA**, therefore, is suitable for plan repairing in dynamic environments, where changes of the aforementioned kinds take place.

The conducted experimental evaluation showed that *DRA** outperformed *A** in most of the cases with modified goal-sets, provided that the percentage of the original plan that has been already executed, is not greater than 40% to 50%. and the change in the goal-set is not be greater than 20% to 50%. The overall performance depends on the average branching factor of the problem, with average higher branching factors corresponding to thresholds of lower values. For re-planning scenarios of modified actions costs, the experimental outcome was that *DRA** outperformed *A** in all experiments.

We believe that the experimental results provide a strong support for the utilization of *DRA** in re-planning scenarios. Nevertheless, a more thorough experimental analysis could provide more useful hints and insights and help us to gain a more elaborate understanding of the underlying mechanisms which determine the strengths and weaknesses of the algorithm.

Concluding, we would like to make the following remarks concerning the potential future work. First, we would like to assess *DRA** performance in scenarios of repeated repairing and in scenarios where both the goal-set and the actions costs are modified, which seem to represent more faithful certain dynamic environments. Another direction that we wish to investigate is the addressing of other types of dynamicity that can be observed in real-world domains, such as altered preconditions and effects for actions, additions and removals of planning agents and invalidations or insertions of new actions.

Finally, since the worst performance of *DRA** is observed in domains with large branching factors which are directly related to the number of the agents activated for the re-planning procedure, we consider that a distributed implementation of *DRA**, where each agent performs an independent search, could be improve substantially the performance of the algorithm. This intuition is further corroborated by the fact that a recent distributed implementation of *A** [22] managed to speed up significantly the runtime performance of the latter.

References

[1] F. Bacchus, “AIPS 2000 planning competition: The fifth international conference on artificial intelligence planning and scheduling systems,” *Ai magazine*, vol. 22, no. 3, p. 47, 2001.

[2] J. Bidot, B. Schattenberg, and S. Biundo, “Plan repair in hybrid planning,” in *Annual Conference on Artificial Intelligence*, pp. 169–176, Springer, 2008.

[3] W. Cushing and S. Kambhampati, “Replanning: A new perspective,” *Proceedings of the International Confer-ence on Automated Planning and Scheduling. Monterey, USA*, pp. 13–16, 2005.

[4] D. Ferguson and A. Stentz, “Field D* : An interpolation-based path planner and replanner,” in *Robotics research*, pp. 239–253, Springer, 2007.

[5] M. Fox, A. Gerevini, D. Long, and I. Serina, “Plan stability: Replanning versus plan repair.,” in *ICAPS*, vol. 6, pp. 212–221, 2006.

[6] A. E. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos, “Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners,” *Artificial Intelligence*, vol. 173, no. 5, pp. 619–668, 2009.

[7] A. E. Gerevini and I. Serina, “Efficient plan adaptation through replanning windows and heuristic goals,” *Fundamenta Informaticae*, vol. 102, no. 3-4, pp. 287–323, 2010.

[8] E. A. Hansen and R. Zhou, “Anytime heuristic search.,” *J. Artif. Intell. Res.(JAIR)*, vol. 28, pp. 267–297, 2007.

[9] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[10] M. Helmert, “The fast downward planning system.,” *J. Artif. Intell. Res.(JAIR)*, vol. 26, pp. 191–246, 2006.

[11] C. Hernández, R. Asín, and J. A. Baier, “Reusing previously found A^* paths for fast goal-directed navigation in dynamic terrain.,” in *AAAI*, pp. 1158–1164, 2015.

[12] J. Hoffmann, “FF: The fast-forward planning system,” *AI magazine*, vol. 22, no. 3, p. 57, 2001.

[13] S. Koenig and M. Likhachev, “D* Lite.,” in *AAAI/IAAI*, pp. 476–483, 2002.

[14] S. Koenig and M. Likhachev, “Real-time adaptive A^*,” in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pp. 281–288, ACM, 2006.

[15] S. Koenig, M. Likhachev, and D. Furcy, “Lifelong planning A^*,” *Artificial Intelligence*, vol. 155, no. 1, pp. 93–146, 2004.

[16] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, “Anytime dynamic A^*: An anytime, replanning algorithm.,” in *ICAPS*, pp. 262–271, 2005.

[17] M. Likhachev, G. J. Gordon, and S. Thrun, “ARA* : *Anytime*A* with provable bounds on sub-optimality,” in *Advances in Neural Information Processing Systems*, p. None, 2003.

[18] D. Long and M. Fox, "The 3rd international planning competition: Results and analysis," *J. Artif. Intell. Res.(JAIR)*, vol. 20, pp. 1–59, 2003.

[19] R. Micalizio, "A distributed control loop for autonomous recovery in a multi-agent plan.," in *IJCAI*, pp. 1760–1765, 2009.

[20] B. Nebel and J. Koehler, "Plan reuse versus plan generation: A theoretical and empirical analysis," *Artificial Intelligence*, vol. 76, no. 1-2, pp. 427–454, 1995.

[21] R. Nissim and R. I. Brafman, "Multi-agent A^* for parallel and distributed systems," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pp. 1265–1266, 2012.

[22] R. Nissim, R. I. Brafman, and C. Domshlak, "A general, fully distributed multi-agent planning algorithm," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pp. 1323–1330, 2010.

[23] S. J. Russell and P. Norvig, "Artificial intelligence (a modern approach)," pp. 93–99, 2010.

[24] A. Stentz *et al.*, "The focussed D$*$ algorithm for real-time replanning," in *IJCAI*, vol. 95, pp. 1652–1659, 1995.

[25] X. Sun, S. Koenig, and W. Yeoh, "Generalized adaptive A^*," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pp. 469–476, 2008.

[26] X. Sun, W. Yeoh, and S. Koenig, "Generalized fringe-retrieving A^*: faster moving target search on state lattices," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pp. 1081–1088, 2010.

[27] X. Sun, W. Yeoh, and S. Koenig, "Moving target D* Lite," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pp. 67–74, 2010.

[28] J. Van Den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 2366–2371, IEEE, 2006.

[29] R. Van Der Krogt and M. De Weerdt, "Plan repair as an extension of planning," in *ICAPS*, vol. 5, pp. 161–170, 2005.

Appendix A Proof of soundness and completeness of *DRA**

In the theorems and lemmas that follow, it is assumed that a part of the original plan has been executed. Consequently, the g-value of the new initial state which is the final state of the executed plan is not set equal to zero, but equal to the cost of the part of the plan that has been executed. Therefore, all the g-values refer to the cost of the path that starts from the original initial state. Note that this adjustment does not affect the validity of the theorems and lemmas that have to do with the optimality of the solution, which would still hold if the g-value of the new initial state was set equal to zero. That is because this convention has as a result that to every g-value, and, therefore, to every path the same constant is added, e.g. the cost of the already executed plan and, therefore, an optimal path will remain optimal. Finally, it is assumed that the number of actions is finite and all the cost actions are positive.

Theorem A.1. *Let s_{init} be the initial state of a DRA* search sr_1 with gs_{init} as goal set, for which a plan has been found. If DRA* is used for a new search sr_2 with $s_{newinit}$ as new initial state, then any state s_K will be marked as valid by the informing procedure, iff there is a path from $s_{newinit}$ to s_K.*

Proof. A state is marked as valid, only if at least one of its predecessor states is valid. By default, only $\mathrm{s_{newinit}}$ is marked as valid when $\mathrm{sr_2}$ begins. Therefore, the first other state that will be marked as valid, will be a successor state of $\mathrm{s_{newinit}}$, to which, by definition, there exists a path from $\mathrm{s_{newinit}}$. Consequently, the next state that will be marked as valid, will be a successor state of one of the aforementioned states. From the previous, it follows by deduction that if there is a path from $\mathrm{s_{newinit}}$ to a state, then this state will be marked as valid by the informing procedure. According to the previous, if there is no path from $\mathrm{s_{newinit}}$ to $\mathrm{s_K}$, then the informing procedure will not find any state marked as valid in the paths that are examined. Therefore $\mathrm{s_K}$ will not be marked as valid.

□

Lemma A.1. *Let s_{init} be the initial state of a DRA* search sr_1 with gs_{init} as goal set, for which a plan has been found. If DRA* is used for a new search sr_2 with $s_{newinit}$ as new initial state and a state s_K has been marked as invalid by the informing procedure, then there is no path from $s_{newinit}$ to s_K.*

Proof. A state $\mathrm{s_K}$ is marked as invalid if the informing procedure has not found any valid predecessor states of $\mathrm{s_K}$. This according to theorem A.1 means that there is no path from $\mathrm{s_{newinit}}$ to $\mathrm{s_K}$'s predecessor states and, therefore, there is no path from $\mathrm{s_{newinit}}$ to $\mathrm{s_K}$. □

Theorem A.2. *Let s_{init} be the initial state of a DRA* search sr_1 with gs_{init} as goal set, for which a plan has been found. If DRA* is used for a new search sr_2 with $s_{newinit}$ as new initial state, then when a state s_K is informed the informing procedure computes the accurate cost of each valid path it examines.*

Proof. If during the informing of a state $\mathrm{s_K}$, its predecessor state state $\mathrm{s_L}$ is valid, then the corresponding p-value that refers to cost of the corresponding path is set equal to the g-value of $\mathrm{s_L}$ plus the cost of the generating action from $\mathrm{s_L}$ to $\mathrm{s_K}$. Therefore, if the g-value of s_L is accurate, then the p-value and, hence, the cost of the path will also be accurate. By default, the g-value of the $s_{newinit}$ is accurate since it corresponds to the cost of the optimal path from s_{init} to $s_{newinit}$. The first p-value that will be computed, will be that of the first state that will be marked as valid, which according to theorem A.1 will be one of $s_{newinit}$ succesor states. Since $s_{newinit}$ g-value and the generating action are also accurate, the corresponding p-value will be accurate. Consequently, the next p-value that will be computed will be that which corresponds to the cost of the path that leads to a successor state of one of the two aforementioned states. From the previous, it follows by deduction that the computing of the cost of each valid path results always in an accurate value. □

Theorem A.3. *Let s_{init} be the initial state of a DRA* sr_1 search with gs_{init} as goal set, for which a plan has been found. If DRA* is used for a new search sr_2 with $s_{newinit}$ as new initial state and s_K is a state to which there exists a path from $s_{newinit}$, then the full informing procedure will find the optimal path from $s_{newinit}$ to s_K.*

Proof. For a given state being informed, the full informing procedure examines all the corresponding paths that lead to it. The examination of each path stops when one of the following three condition is met: a) the parent-pointer is marked as valid, b) the parent-pointer is marked as invalid or c) a circle has been detected. Case b) means that the path being examined is not valid. Case c) means that the path being examined is not optimal, since all actions have positive costs and therefore an optimal path cannot contain circles. Since according to theorem A.2, informing procedure computes the accurate cost of each path it examines, it follows that the valid path with the cost will be the optimal path from $s_{newinit}$ to s_K. □

Theorem A.4. *Let s_{init} be the initial state of a DRA* search sr_1 with gs_{init} as goal set, for which a plan has been found. If DRA* is used for a new search sr_2 with $s_{newinit}$ as new initial state, no actions costs have been decreased and there is at least one path from $s_{newinit}$ to s_K, then the lazy informing procedure will find the optimal path from $s_{newinit}$ to s_K.*

Proof. For a given state being informed, lazy informing procedure examines one by one the corresponding paths that lead to in an increasing order with respect to to their original costs and stops if the cost of the path p_{min} which has the smaller updated cost of the already examined paths is not greater than the original cost of the path that is to be examined next. The examination of each path stops when one of the following three condition is met:a) the parent-pointer is marked as valid, b) the parent-pointer is marked as invalid or c) a circle has been detected. Case b) means that the path being examined is not valid. Case c) means that this path being examined is not optimal, since all actions have positive costs and therefore an optimal path cannot contain circles. Since no actions costs have been decreased, the cost of any of these paths cannot have been decreased. Therefore, since the updated cost of p_{min} is not greater than the original cost of another plan p_l, then it will be not greater than its updated cost. Also, since the paths were ordered in an increasing order according to their original costs, it follows that if the updated cost of p_{min} is not greater than the the original cost of the path that was to be examined after it, then it cannot be greater than the original cost of any other non-examined path. From the previous, it follows that there is no path with smaller cost than p_{min}, and therefore p_{min} is optimal, since, according to theorem A.2, informing procedure computes the accurate cost of each path it examines. □

Lemma A.2. *The informing procedure terminates.*

Proof. There are two ways that could lead to the non-termination of the informing procedure: a)an infinite number of examined paths or b) an infinite loop during the examination of a path. Since the number of the states in the open and closed list is finite, it follows that the number of paths that do not contain circles is finite too. Also, the examination of a path is aborted when a circle is detected. Therefore, neither the first nor the second condition that can lead to the non-termination of the informing procedure can hold and, therefore, the informing procedure always terminates.

□

Theorem A.5. *A^* is sound, if the heuristic function used produces h-values that are admissible.*

Proof. Assume that the cost of the optimal plan p_{op} for a given problem is equal to c_{op}. A^* expands at each step the state from the open list that has the minimum f-value. Supposing that a plan p' is found which is sub-optimal and the cost of which is equal to c'. The final state of the plan p_{op} was not expanded, otherwise p_{op} would have been found. If s_{Fop} and $s_{F'}$ are the final states of p_{op} and p respectively, then it must hold that fval $_{F_{op}} \geq$ fval $_{F'}$ (1). Also, since the heuristic function produces admissible h-values, then it holds that

$$fval_{F_{op}} = gval_{F_{op}} + hval_{F_{op}} = gval_{F_{op}} + 0 = c_{op}\ (2)$$
$$fval_{F'} = gval_{F'} + hval_{F'} = gval_{F'} + 0 = gval_{F'} = c'(3)$$

From (1),(2) and (3) it follows that $c_{op} \geq c'$ which is a contradiction, since we assumed that p' is sub-optimal and, thus, its cost must be grater than the cost of p_{op}. □

Theorem A.6. *A^* is complete, if the heuristic function used produces h-values that are admissible.*

Proof. Assuming that c_{op} is the cost of the optimal path, then in order for the path's final state to be expanded there must be only finitely many states with cost less than or equal to c_{op}. Due to the fact that every action's cost is positive and the number of actions is finite, it follows that the number of the paths from the initial state to to any other state which have cost less than c_{op} must be finite. Since the f-value of any state s_K is always greater than the cost of the corresponding cost of the path from the initial state to s_K, it follows that the number of states with a f-value less than c_{op} is finite, and, therefore, A^* is complete.

□

Lemma A.3. *If DRA* begins with an empty closed list and an open list which contains only its initial state, then it is equivalent to an A^* search that starts from the same initial state.*

Proof. Each step of *DRA** executed in the same way as each step of A^* : the state with the lowest f-value from the open list is selected and, if it is not a goal-state, it is generated and its succesor states are inserted in the open list, while the expanded state is inserted in the closed list. There are two differences between *DRA** and *A**. The first is that in the beginning of the execution its closed list might not be empty and its open list might contain other states and not only the initial state of the search as in the case of *A**. According to the assumption none of the two previous facts do not hold. The other difference has to do with the procedure of informing for states that lie in the closed or open list. In this case, since the only state inside the original open list is valid, all other states that are inserted in the closed or open list are also valid by default, which means that the informing procedure does not take place for any state. Therefore, in this setting *DRA** is equivalent to *A**.

□

Lemma A.4. *Let sr_1 be an A* search. Suppose that during sr_1 a finite number of states is inserted in the open list and a finite number of states is inserted in the closed list. Also, suppose that some of the aforementioned states may have wrong f-values or may be invalid, i.e. not be accessible from the initial state. A* is sound and complete if the following three conditions are met:i) the expanding of the states in the open list is carried out in a non-decreasing way with respect to to their f-values, ii) none of the inserted states in the closed list satisfies the goal-set and iii) the f-value of any state is never greater than the cost of the optimal path from this state to a goal-state.*

Proof. Since the invalid states are not utilized in any way, the execution of the algorithm is not affected by their insertion. Also, since the states are expanded in a non-decreasing way with respect to to their f-values and the f-values are never greater the cost of the optimal path that leads to a goal-state, if a plan is found, then it will be optimal in regard to all the states in the open list, since all other non-expanded states have at least the same f-value and, if any of them is a goal-state then the cost of the corresponding path will have at least the same cost. Moreover, since none of the inserted states in the closed list satisfies the goal-set, there can be no state in the closed list that is a goal-state. Consequently, the plan found will be also optimal with respect to all the states in the closed list. Therefore *A** is sound in this case. Finally, since the number of the states inserted in the open list is finite, following the same line of reasoning as in theorem A.6, we reach the conclusion that *A** is also complete in this case.

□

Lemma A.5. *Let s_{init} be the initial state and gs_{init} the goal set for which DRA* has found a plan p_1. Let $s_{newinit}$ be the new initial state that lies in p_1 and gs_{new} the new goal-set. Then, the new DRA* search corresponds to an A* search from $s_{newinit}$ in the open and closed list of which a finite number of states has been inserted which may have wrong f-values or may be invalid, i.e. not be accessible from the initial state.*

Proof. According to lemma A.3 the initial search of *DRA** is equivalent to an *A** search that starts from $\mathrm{s}_{\mathrm{init}}$. The new *DRA** search utilizes the final closed and open list of the original search. Consequently, the new search from $\mathrm{s}_{\mathrm{newinit}}$ in this case begins using the open and closed lists from the previous search, which is equivalent to an *A** search from $\mathrm{s}_{\mathrm{newinit}}$ with a finite number of states inserted in its open and closed list which may have wrong f-values or may be invalid. Also, according to lemma A.2 the informing procedure of *DRA** always terminates, and, therefore, there can be no difference in the behaviour of the new *DRA** search and its corresponding A^* search because of this procedure.

□

Lemma A.6. *Let s_K and s_L be states respectively. If there is a path from s_K to s_L, then any state reachable from s_L is also reachable from s_K .*

Proof. Any state $\mathrm{s_M}$ reachable from $\mathrm{s_L}$, can be reached from $\mathrm{s_K}$ by following the path from $\mathrm{s_K}$ to $\mathrm{s_L}$ and, then, the path from $\mathrm{s_L}$ to $\mathrm{s_M}$. □

Lemma A.7. *Let s_K be a state that is located in the open list during the execution of a DRA* search. Its g-value $gval_{S_K}$ corresponds to the cost of the optimal path from s_{init} to s_K for the given expanded states, i.e. there is no path from s_{init} to s_K with a smaller cost.*

Proof. $gval_{\mathrm{s_K}}$ is equal to the cost of the path from $\mathrm{s_{init}}$ to $\mathrm{s_K}$. If only one state has generated $\mathrm{s_K}$, then there is only one path from $\mathrm{s_{init}}$ to $\mathrm{s_K}$, which is, by definition, optimal. If there are more than one states, then each time $\mathrm{s_K}$ is re-generated, its current g-value is compared to the g-value that results from the new predecessor state and the smaller value is kept and, therefore, $\mathrm{gval_{S_K}}$ is equal to the cost of the optimal path from $\mathrm{s_{init}}$ to $\mathrm{s_K}$. □

Lemma A.8. *Let s_K be a state that is located in the open list during the execution of a DRA* search and s_M another state that lies in the optimal path from the initial state s_{init} to s_K. The cost of the optimal path from s_M to s_K is equal to to $gval_{s_K} - gval_{s_M}$.*

Proof. If there is a path from $\mathrm{s_M}$ to $\mathrm{s_K}$ with a cost smaller than $\mathrm{gval_{S_K}}$ - $\mathrm{gval_{S_M}}$, then there will be a path from $\mathrm{s_{init}}$ to $\mathrm{s_K}$ with a cost smaller than $\mathrm{gval_{S_K}}$, which cannot hold according to lemma A.7 . □

Lemma A.9. *Let s_M be a state which is located in the final open list of a DRA* search that has been completed and let $s_{newinit}$ be the new initial state for a new DRA* search. If $s_{newinit}$ was in the plan of the original search and if no actions costs have been decreased, then during the new DRA* search, the g-value of any state s_L that was located in the final open list of the original search can never decrease, unless s_L is generated by a state which was not located in the final closed list of the original search.*

Proof. According to lemma A.8 , the difference $gval_{\mathrm{s_L}} - gval_{\mathrm{s_{newinit}}}$ is equal to the cost of the optimal plan from $\mathrm{s_{newinit}}$ to $\mathrm{s_L}$. Also, $\mathrm{gval_{s_{newinit}}}$ is minimum since it lies in the optimal path from $\mathrm{s_{init}}$ to $\mathrm{s_{newinit}}$. Since no actions costs are decreased, then it follows than the cost of any path cannot decrease. Therefore, if there was a state in the closed list which generated $\mathrm{s_L}$ and resulted in a smaller $\mathrm{gval_{s_{newinit}}}$ it would correspond to a path with a decreased cost which is a contradiction according to the previous. □

Theorem A.7. *Let s_{init} be the initial state and gs_{init} the goal set of a DRA* search that has been completed. If s_{new} is the new initial state and gs_{new} the new goal-set of a new DRA* search, then DRA* is sound and complete if the h-values that are used are admissible.*

Proof. According to lemma A.5, *DRA** corresponds in this case to an *A** search from $\mathrm{s_{newinit}}$ which instead of using a new open and closed list, utilizes the final closed and open lists from the previous search and which utilizes an informing procedure in order to compute the f-values of the states with respect to to $\mathrm{s_{newinit}}$ and determine which states are valid. *DRA** repairs lazily each state of the open list by re-computing its g-value and its h-value that correspond to the new goal-set, which results in an updated f-value for all the valid states, while all the invalid states are removed from the open list. Consequently, the open list is sorted according to the new f-values of its states. According to theorem A.4, lazy informing finds the optimal path for any state to which there is a path from $\mathrm{s_{newinit}}$ and, therefore, since *DRA** utilizes admissible heuristic, the f-value of every state $\mathrm{s_M}$ cannot be greater than the cost of the optimal path. Also according to the same theorem, lazy informing marks as invalid and, hence, does not expand the states to which no past exist from $\mathrm{s_{newinit}}$. Let's assume that there is state $\mathrm{s_A}$ with $\mathrm{fval_A}$, which is expanded before a state $\mathrm{s_B}$, which has a smaller $\mathrm{fval_B}$. In order for this to be possible, there must be another state $\mathrm{s_K}$ in the open list which generates $\mathrm{s_B}$ and has not been expanded before $\mathrm{s_A}$, which will result to $fval_{\mathrm{B}} < fval_{\mathrm{A}}(1)$. Since all actions costs are positive it holds that $gval_{\mathrm{K}} < gval_{\mathrm{B}}(2)$. Also since h-values are admissible, it holds that $hval_{\mathrm{K}} \leq hval_{\mathrm{B}} + c_{\mathrm{action_{KB}}}(3)$. From (2) and (3), it is derived that $hval_{\mathrm{K}} + gval_{\mathrm{K}} < hval_{\mathrm{B}} + gval_{\mathrm{B}} \rightarrow fval_{\mathrm{K}} < fval_{\mathrm{B}}$ (4). From (1) and (4), it follows that $fval_{\mathrm{K}} < fval_{\mathrm{A}}$ and, therefore, $\mathrm{s_K}$ will be expanded before $\mathrm{s_A}$. But this is a contradiction, since we assumed that $\mathrm{s_K}$ has not been expanded before $\mathrm{s_A}$. From the previous, it is derived that the expanding of the states in the open list is carried out in a non-decreasing way with respect to to the f-values. Moreover, *DRA** does not utilize any invalid states. If none of the states in the final closed list satisfy $\mathrm{gs_{new}}$, *DRA** is sound and complete according to theorem A.4. Besides, before the execution of *DRA** begins, the original closed list is traversed and searched for goal-satisfying states in which case the one with the lowest g-value is kept. The path leading to this state is returned as a solution when the priority of the open list becomes greater that this g-value. Since the f-value of every state $\mathrm{s_M}$ cannot be greater than the cost of the optimal path, the solution returned from the algorithm in this case is optimal. Therefore, *DRA** is sound and complete in this case also. □

Lemma A.10. *Let s_{init} be the initial state and gs_{init} the goal set of a DRA* search that has been completed. If s_{new} is a new initial state and gs_{new} a new goal-set which is a super-set of gs_{init}, no state in the final closed list of the original search can satisfy gs_{new}.*

Proof. Since $\mathrm{gs_{new}}$ is a superset of $\mathrm{gs_{init}}$, every state satisfying $\mathrm{gs_{new}}$ satisfies also $\mathrm{gs_{init}}$. Therefore, if a state exists in the final closed list that satisfies $\mathrm{gs_{new}}$ it will also satisfy $\mathrm{gs_{init}}$. But this is impossible, since, by definition, this state would correspond to a solution for the original search in which case, it is not expanded and, therefore, it is not inserted in the closed list.

□

Theorem A.8. *Let s_{init} be the initial state and gs_{init} the goal set of a DRA* search that has been completed. Supposing a new initial state s_{new}, unchanged actions' costs and a new goal-set gs_{new}, which is a super-set of gs_{init}, DRA* is sound and complete if the h-values that are used are admissible.*

Proof. According to lemma A.5, *DRA** corresponds in this case to an A^* search from $\mathrm{s_{newinit}}$ which instead of using a new open and closed list, utilizes the final closed and open lists from the previous search and which utilizes an informing procedure in order to compute the f-values of the states with respect to to $\mathrm{s_{newinit}}$ and determine which states are valid. In this case, the closed list is not traversed since the new goal-set is a superset of the original goal-set(a set S is always a superset of itself), and according to lemma A.10 it is impossible that a state in the closed list satisfies the goal set. *DRA**, repairs lazily each state of the open list that is selected for expansion by re-computing a new g-value, which results in an updated f-value for it. According to theorem A.4, lazy informing finds the optimal path for any state to which there is a path from $\mathrm{s_{newinit}}$ and, therefore, since *DRA** utilizes admissible heuristic, the f-value of every state $\mathrm{s_M}$ cannot be greater than the cost of the optimal path. Also according to the same theorem, lazy informing marks as invalid and, hence, does not expand the states to which no past exist from $\mathrm{s_{newinit}}$.

According to lemma A.9 , since no actions costs changes are decreases, the g-value of a state can only decrease, if it is generated by another state that lies in the open list. Also, since $\mathrm{gs_{new}}$ is a superset of $\mathrm{gs_{init}}$, then the initial h-value of a state is still smaller than the cost of the state from the state to a goal-state. From the previous, it is derived the f-value of a state can only decrease, if it is generated by a state that lies in the open list. Let's assume that there is state $\mathrm{s_A}$ with $\mathrm{fval_A}$, which is expanded before a state $\mathrm{s_B}$, which has a smaller $\mathrm{fval_B}$. In order for this to be possible, there must be another state $\mathrm{s_K}$ in the open list which generates $\mathrm{s_B}$ and has not been expanded before $\mathrm{s_A}$, which will result to $fval_{\mathrm{B}} < fval_{\mathrm{A}}(1)$. Since all actions costs are positive it holds $gval_{\mathrm{K}} < gval_{\mathrm{B}}(2)$. Also since h-values are admissible, it holds that $hval_{\mathrm{K}} \leq hval_{\mathrm{B}} + c_{\mathrm{action_{KB}}}(3)$. From (2) and (3), it derives that $hval_{\mathrm{K}} + gval_{\mathrm{K}} < hval_{\mathrm{B}} + gval_{\mathrm{B}} \rightarrow fval_{\mathrm{K}} < fval_{\mathrm{B}}$ (4). From (1) and (4), it follows that $fval_{\mathrm{K}} < fval_{\mathrm{A}}$ and therefore, $\mathrm{s_K}$ will be expanded before $\mathrm{s_A}$. But this is a contradiction, since we assumed that $\mathrm{s_K}$ has not been expanded before $\mathrm{s_A}$. From the previous, it is derived that the expanding of the states in the open list is carried out in a non-decreasing way with respect to to the f-values. Therefore, according to theorem A.4 , *DRA** is sound and complete.

□

Theorem A.9. *Let s_{init} be the initial state and gs_{init} the goal set of a DRA* search that has been completed. Supposing a new initial state, the same goal-set and a number of actions costs' changes, DRA* is sound and complete if the h-values that are used are admissible.*

Proof. According to lemma A.5, *DRA** corresponds in this case to an A^* search from $\mathrm{s_{newinit}}$ which instead of using a new open and closed list, utilizes the final closed and open lists from the previous search and which utilizes an informing procedure in order to compute the f-values of the states with respect to to $\mathrm{s_{newinit}}$ and determine which states are valid. In this case, the closed list is not traversed since the new goal-set is a superset of the original goal-set(a set S is always a superset of itself), and according to lemma A.10 it is impossible that a state in the closed list satisfies the goal set. *DRA** repairs fully each state of the open list by re-computing its g-value and its h-value that correspond to the new goal-set, which results in an updated f-value for all the valid states, while all the invalid states are removed from the open list. Consequently, the open list is sorted according to the new f-values of its states. According to theorem A.3, full informing finds the optimal path for any state to which there is a path from s_{newinit} and, therefore, since *DRA** utilizes admissible heuristic, the f-value of every state s_{M} cannot be greater than the cost of the optimal path. Also according to the same theorem, informing marks as invalid and, hence, does not expand the states to which no past exist from s_{newinit}.

Let's assume that there is state s_{A} with $\mathrm{fval}_{\mathrm{A}}$, which is expanded before a state s_{B}, which has a smaller $\mathrm{fval}_{\mathrm{B}}$. In order for this to be possible, there must be another state s_{K} in the open list which generates s_{B} and has not been expanded before s_{A}, which will result to $fval_{\mathrm{B}} < fval_{\mathrm{A}}(1)$. Since all actions costs are positive it holds that $gval_{\mathrm{K}} < gval_{\mathrm{B}}(2)$. Also since h-values are admissible, it holds that $hval_{\mathrm{K}} \leq hval_{\mathrm{B}} + c_{\mathrm{action_{KB}}}(3)$. From (2) and (3), it derives that $hval_{\mathrm{K}} + gval_{\mathrm{K}} < hval_{\mathrm{B}} + gval_{\mathrm{B}} \Rightarrow fval_{\mathrm{K}} < fval_{\mathrm{B}}$ (4). From (1) and (4), it follows that $fval_{\mathrm{K}} < fval_{\mathrm{A}}$ and therefore, s_{K} will be expanded before s_{A}. But this is a contradiction, since we assumed that s_{K} has not been expanded before s_{A}. From the previous, it is derived that the expanding of the states in the open list is carried out in a non-decreasing way with respect to to the f-values. Therefore, according to theorem A.4, *DRA** is sound and complete.

□

Theorem A.10. *Let s_{init} be the initial state and gs_{init} the goal set of a DRA* search that has been completed. Supposing a new initial state, the same goal-set and a number of actions costs' increases, DRA* is sound and complete if the h-values that are used are admissible.*

Proof. According to lemma A.5, *DRA** corresponds in this case to an A^* search from s_{newinit} which instead of using a new open and closed list, utilizes the final closed and open lists from the previous search and which utilizes an informing procedure in order to compute the f-values of the states with respect to to s_{newinit} and determine which states are valid. In this case, the closed list is not traversed since the new goal-set is a superset of the original goal-set(a set S is always a superset of itself), and according to lemma A.10 it is impossible that a state in the closed list satisfies the goal set. *DRA**, repairs lazily each state of the open list that is selected for expansion by re-computing a new g-value, which results in an updated f-value for it. According to theorem A.4, lazy informing finds the optimal path for any state to which there is a path from s_{newinit} and, therefore, since *DRA** utilizes admissible heuristic, the f-value of every state s_{M} cannot be greater than the cost of the optimal path. Also according to the same theorem, lazy informing marks as invalid and, hence, does not expand the states to which no past exist from s_{newinit}.

According to lemma A.9 , since no actions costs changes are decreases, the g-value of a state s_{M} can only decrease, if it is generated by another state that lies in the open list. Also, since the goal-set does not change then the initial h-value which was admissible remains smaller than the cost of the path from s_{M} to a goal-state. From the previous, it is derived the f-value of a state can only decrease, if it is generated by a state that lies in the open list. Let's assume that there is state s_{A} with $\mathrm{fval}_{\mathrm{A}}$, which is expanded before a state s_{B}, which has a smaller $\mathrm{fval}_{\mathrm{B}}$. In order for this to be possible, there must be another state s_{K} in the open list which generates s_{B} and has not been expanded before s_{A}, which will result to $fval_{\mathrm{B}} < fval_{\mathrm{A}}(1)$. Since all actions costs are positive it holds $gval_{\mathrm{K}} < gval_{\mathrm{B}}(2)$. Also since h-values are admissible, it holds that $hval_{\mathrm{K}} \leq hval_{\mathrm{B}} + c_{\mathrm{action_{KB}}}(3)$. From (2) and

(3), it derives that $hval_{\mathrm{K}} + gval_{\mathrm{K}} < hval_{\mathrm{B}} + gval_{\mathrm{B}} \to fval_{\mathrm{K}} < fval_{\mathrm{B}}$ (4). From (1) and (4), it follows that $fval_{\mathrm{K}} < fval_{\mathrm{A}}$ and therefore, $\mathrm{s_K}$ will be expanded before $\mathrm{s_A}$. But this is a contradiction, since we assumed that $\mathrm{s_K}$ has not been expanded before $\mathrm{s_A}$. From the previous, it is derived that the expanding of the states in the open list is carried out in a non-decreasing way with respect to to the f-values. Therefore, according to theorem A.4 , *DRA** is sound and complete.

□

Theorem 1. *DRA* is sound and complete for repairing scenarios of goal-set modifications or actions costs changes if the h-values that are used are admissible.*

Proof. *DRA** is sound and complete for scenarios when the new goal-set has been modified according to theorem A.7. *DRA** is sound and complete for scenarios when the new goal-set is a superset of the original goal-set according to theorem A.8. *DRA** is sound and complete for scenarios with actions' costs than can be increased or decreased according to theorem A.9. *DRA** is sound and complete for scenarios with actions' costs than can be increased according to theorem A.10. Therefore, *DRA** is sound and complete for every re-planning scenario it addresses.

□