

Case Adaptation by Segment Replanning for Case-Based Planning Systems

Flavio Tonidandel and Marcio Rillo

Centro Universitário da FEI - UniFEI
Av. Humberto de A. Castelo Branco, 3972
09850-901 - São Bernardo do Campo - SP - Brazil
flaviot@fei.edu.br, rillo@fei.edu.br

Abstract. An adaptation phase is crucial for a good and reasonable Case-Based Planning (CBP) system. The adaptation phase is responsible for finding a solution in order to solve a new problem. If the phase is not well designed, the CBP system may not solve the desirable range of problems or the solutions will not have appropriate quality. In this paper, a method called CASER – Case Adaptation by Segment Replanning – is presented as an adaptation rule for case-based planning system. The method has two phases: the first one completes a retrieved case as an easy-to-generate solution method. The second phase improves the quality of the solution by using a generic heuristic in a recursive algorithm to determine segments of the plan to be replanned. The CASER method does not use any additional knowledge, and it can find as good solutions as those found by the best generative planners.

1 Introduction

The adaptation phase of a CBR (Case-Based Reasoning) system is responsible for finding a solution from a retrieved case. Specifically for Case-Based Planning systems, the adaptation phase is important for finding good quality plans, where the quality refers to the solution plan length and the rational use of resources and time.

However, a high quality solution is not easy to find. In fact, searching for an optimal solution is *NP-Hard* [12] even for generative planning or case-based planning systems. Actually, many solutions usually obtained by case-based planning systems [14] are longer than necessary. Although there is no advantage for adapting plans over a complete regeneration of a new solution in the worst case, as shown by Nebel and Koehler [12], the adaptation of plans can be the best choice in most situations [6] and it has great potential to be better than planning from scratch [2].

Gerevini and Serina [6] propose the ADJUST-PLAN algorithm that adapts an existing plan in order to find a solution to a new problem. Their algorithm, although not designed for Case-Based Planning (CBP) systems, adapts an existing plan instead of finding an entire plan when the problem is modified. The main problem of the algorithm is that its technique requires constructing an entire planning graph, which can be computationally expensive for complex domains [6].

Another approach is the Planning by Rewriting paradigm [1]. It addresses the problem of adapting a solution plan through rewriting rules in order to find a better quality plan. However, rewriting rules are domain-dependent rules that can be determined by hand or by a learning process [1], which requires some extra knowledge about the domain.

In contrast of the previous approaches, this paper presents an domain-independent adaptation process. This method, called CASER (*Case Adaptation by Segment Replanning*), has two phases. First it finds a low quality solution by a simple completion of the retrieved case, and then the second phase uses the FF-heuristic [9] to detect sub-plans (or segments) in the solution plan that can be replanned in order to improve the quality of the solution.

This paper focuses in a STRIPS-like version of the CASER method, where just the number of steps can define a solution plan's quality. Its improving to deal with metrical and temporal domains is discussed in the discussion section.

2 Solution Quality and Plan Adaptation

The challenge of most plan adaptation processes is to determine which part of a plan must be adapted in order to achieve a correct solution, guaranteeing its high quality.

In a case-based planning domain, a case is a plan and the improvement of the case quality in the STRIPS-version is the reduction of the number of actions that guides the plan from the initial state to the final state. The purpose of an adaptation process applied to a Case-Based planning system is to change, add or even delete appropriate actions of the case in order to find a better solution.

There are in the literature dedicated efforts on adaptation processes. One adaptation process is the ADJUST-PLAN method [6], which refines a plan until it becomes a new solution. In fact, it tries to find a complete solution from a given plan by refining sub-plans on the graph created by the Graphplan system [4]. It is well specified to work in a domain where problems are partially modified and, consequently, some sub-graphs used to solve previous problems can be re-used to find and refine a solution to a new problem. However, in general, when a case is retrieved from a case base, the case does not contain any previous planning graph, forcing the ADJUST-PLAN method to create the entire graph. The process to create an entire graph can be computationally expensive and the method can be unable to be used efficiently in case-based systems. Similar to ADJUST-PLAN is the replanning process called SHERPA [10]. Although not designed to improve solution quality, it finds replanned solutions whose qualities are as good as those achieved from scratch [10].

The adaptation of a plan can also be useful for generative planning systems. A new planning paradigm, called Planning by Rewriting (PbR), uses an adaptation phase, called rewriting, that turns a low quality plan into a high quality plan by using some domain dependent rules. PbR is a planning process that finds an easy-to-generate solution plan and then adapts it to yield a better solution by some domain-dependent rules. These rules can be designed by hand or through an automated learning process [1]. In both cases, PbR needs additional domain specific knowledge and a complete specification of the rules or some training instances for the learning process.

In fact, there is no adaptation method which finds and improves the solution quality from a retrieved case without any additional knowledge besides standard planning knowledge, such as an operator's specification and initial and goal states.

This paper introduces a domain-independent and anytime behavior adaptation process, called CASER (*Case Adaptation by Segment Replanning*), that finds an easy-to-generate solution with low quality and adapts this solution to improve it. It follows the same idea as PbR. However, this method does not require any additional domain knowledge apart from an operator's specification, an initial state and a goal state. As we discuss later, the CASER method is very useful for case-based planners that uses Action Distance-Guided (ADG) similarity [13] to retrieve cases, as used in the FAR-OFF system [14], or even for those CBP systems where the solutions are usually longer than necessary.

3 The CASER Method

The CASER method has two main phases, namely:

1. The completion of a retrieved case in order to find a new solution (easy-to-generate solution); It uses a modified version of the FF planning system [9] to complete the case.
2. A recursive algorithm that replans solution segments in order to improve the final solution quality by using a modified version of the FF-heuristic and of the FF planning system [9].

The two processes do not use any additional knowledge and use a generative planning system, which is a modified version of the original FF planning system [9] in this STRIPS-like version, described below.

3.1 A modified version of the FF planner

The original FF planning system, as presented in [9], is designed to plan in a fast way by extracting an useful heuristic, called FF-heuristic, from a relaxed graph similar to GraphPlan [4] graph but without considering delete lists of actions. The FF-heuristic is the number of actions of the relaxed plan extracted from the relaxed graph. The FF planner uses the FF-heuristic to guide the Enforced Hill-climbing search to the goal.

In order to improve the efficiency of the FF planner, Hoffmann and Nebel [9] introduces some additional heuristics that prunes states, such as the *added-goal deletion* heuristic. In this modified version of the FF planner, the FF-heuristic is used in its regular form with a small modification: it permits that the relaxed graph expands further until a fixpoint is reached.

As defined by Hoffmann and Nebel [9], the relaxed graph is created by ignoring the delete list of the actions. It is constituted by layers that comprise alternative facts and actions. The first fact layer is the initial state (*initst*). The first action layer contains all actions whose preconditions are satisfied in *initst*. Then, the add lists of these actions are inserted in the next fact layer together with all facts from the previous fact layer, which leads to the next action layer, and so on. The regular

specification of the FF-heuristic is that the relaxed graph is expanded until all goals are in the last fact layer. With the modification discussed above, the relaxed graph must be created until a fixpoint is found, i.e., when there are no more fact layers that are different from the previous ones.

With the relaxed graph created until the fixpoint, a relaxed solution can be found for any state that can be reached from *initst*. The process of determining the relaxed solution, following Hoffmann and Nebel [9], is performed from the last layer to the first layer, finding and selecting actions in layer *i-1* if it is the case that their add-list contains one or more of goals initialized in layer *i*. After that, the preconditions of the selected actions are initialized as new goals in their previous and corresponding layer. The process stops when all unsatisfied goals are in the first layer, which is exactly the initial state. The relaxed solution is the selected actions in the graph and the estimate distance is the number of actions in this relaxed solution.

The FF-heuristic returns, therefore, the number of estimated actions between *initst* and any possible state reached from *initst*. We will use this heuristic in the modified version of the FF planner.

However, the modification of the FF-heuristic to expand the graph until fixpoint is not enough for the modified version of the FF planner. In fact, we must also change the *added-goal deletion* heuristic. In its regular specification, the *added-goal deletion* heuristic does not consider a state on which a goal that has been just reached can be deleted by the next steps of the plan in the search tree. The heuristic analyzes the delete lists of all actions in the relaxed solution provided by the FF-heuristic. If a reached goal is in the delete list of any action in the relaxed solution, the FF planner does not consider this state in the search tree.

The heuristic works fine when the goal contains no predicates that are easy to change, like *handempty* in the Blocks World domain or the *at(airplane,airport)* in the logistic domain. If one of these easy-to-change predicates is in the goal, the FF planner can not find a solution.

For example, consider that the initial state of a planning problem in Blocks World domain is *on(A,B)*, *clear(A)*, *ontable(B)* and *holding(C)*; and the goal is *on(A,C)* and *handempty*. The regular solution plan must have many actions that delete the *handempty* predicate. The FF planning system, therefore, is unable to find a solution because the *added-goal deletion* will try to avoid that the *handempty* be deleted and, consequently, it will prune the next states.

Since the CASER method will consider complete and consistent states as goals, called *extended goals*, it would experience some problems by using the original FF planning system to find an alternative plan. Therefore, in order to avoid this problem, we implemented a modified version of the FF planning system. In this version, a *relaxed added-goal deletion* heuristic is implemented.

The *relaxed added-goal deletion* heuristic does not prune a state if the predicate goal reached has one of the following conditions:

1. It is presented in the add-list of more than one action;
2. It is the unique predicate in a precondition of at least one action.

The first condition excludes predicates like *holding(_)* and *handempty* in the Blocks World domain. The second condition allows that a specific action with a unique predicate in preconditions can be used.

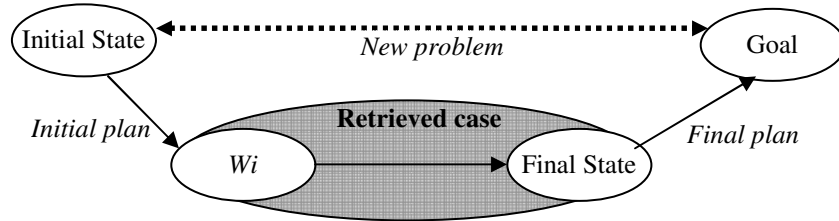


Fig. 1 – The Completion Process – it shows the retrieved case and its Wi and Final State. Given a new problem, the retrieved case becomes a sub-plan of the solution composed by the Initial Plan + Retrieved Case + Final Plan.

Besides the *relaxed added-goal deletion* heuristic, the modified version of the FF planner has also the following features:

- It does not switch to the complete Best First Search if a solution was not found by using Enforced Hill-Climbing search;
- It avoids an specific action as the first action in the solution plan;
- It allows to specify the maximal length of a solution;
- It allows to specify an upper-time limit time to find a solution.

The first constraint described above is imposed because it is not important to find a solution at any time cost; the remaining three items are imposed to permit the CASER method to focus the FF planning to its purpose.

The aim of these modifications is not to improve the FF planning system capacity, but only adapts the system to be used in the CASER method. Since these modifications relax some FF planner features, there is also a great probability that the original system outperforms this modified version.

3.2 The Completion Process - the first phase

The easy-to-generate solution may be obtained by completing a retrieved case. Each retrieved case must be a plan (a complete solution of an old problem) and it must have its correct sequence of actions.

In fact, the CASER method in this first phase receives an initial state, a goal state and a totally ordered plan from the retrieved case that does not match on either of two states necessarily. In other words, the retrieved case is a sub-plan of the solution.

The completion of the case must extract the precondition of the retrieved case. This precondition is called Wi [13]. Informally, Wi is a set of those literals that are deleted by the plan and that must be in the initial state necessarily. It is equal to the result of the foot printing method used by PRODIGY/ANALOGY system [17].

The completion phase of the CASER process finds a simple way to transform the retrieved case into a solution. It just tries to expand the retrieved case backward in order to match the initial state and expand it forward in order to satisfy the goal.

The backward expansion is a plan, called *initial plan*, that bridges the initial state and the retrieved case. This plan is found by a generative planner applied from the initial state to Wi of the case, where Wi is the goal to be achieved. Since the Wi may be an extended goal, the CASER method uses the modified version of FF planning system as its generative planner. Figure 1 shows the completion process.

```

procedure CASER_completion(retrieved_case, initst, goalst)
  Wi ← find_Wi (retrieved_case);
  FS ← find_final_state(retrieved_case);
  Initial_plan ← modified_FF(initst, Wi);
  Final_plan ← modified_FF(Final_state, goalst);
  return Initial_plan + retrieved_case + Final_plan;
end;

```

Fig. 2. The completion algorithm. The *initst* is the initial state of a problem and *goalst* is the goal state. The *modified_FF* finds a plan from its first argument to its second argument.

The forward expansion is similar to the backward expansion, but it finds a plan, called *final plan*, that fits the final state of the case and the goal of the new problem. The final state of the case can be easily found by executing the actions effects of the new adapted case (*initial plan* joined with the retrieved case) from the initial state.

The final state of the case becomes a new initial state for the modified FF-planner that finds the *final plan* from this final state direct to the new goal.

At this stage of the CASER method, a complete solution of the new problem is stated by the joint of *initial plan*, retrieved case and *final plan*. Figure 2 summarizes the algorithm of this first phase.

However, as stated before, the solution obtained by the completion phase is just an easy-to-generate plan and it may not have appropriate quality since it can have more actions than necessary. The second phase of the CASER method is designed to reduce the length of the plan and, consequently, increases its quality.

3.3 The Recursive Replanning Process - the second phase

The planning search tree is usually a network (e.g. a graph) of actions composed of all states and all actions in a specific domain. For total-order planners, a solution plan is necessarily a sequence of actions, and it can be represented as a path in a directed graph where each node is a consistent state and each edge is an action.

Considering a distance function, h^d , it is possible to estimate the number of actions between two planning states. If this function is applied to estimate the distance from an initial state to all other states in a directed graph, it is possible to determine the costs of reaching each state from the initial state in number of actions.

However, there is no accurate distance function to determine optimal costs for each state without domain specific rules or that takes reasonable time to do it. An approximation of these optimal costs can be obtained by using a heuristic function used by the heuristic search planners, such as FF-heuristic [9] and HSP-heuristic [5]. Considering the recent results of the FF system, the FF-heuristic is one of the best choices to be used in this cost estimation. It was also successfully used in other methods, such as the ADG similarity [13] and the FAR-OFF system [14].

The CASER method uses the FF-heuristic in its second phase in order to calculate the cost of each state and to determine the segments for replanning. This second phase was first published in [15] as the SQUIRE method (Solution Quality Improvement by Replanning) for refining solution plans from generative planning system. This paper improves this method and uses it to adapt plan reused from cases.

```

procedure det_costs(plan, initst)
  fixpoint  $\leftarrow$  create_relaxed_graph(initst)
   $S_0 \leftarrow$  initst;
   $v_0 \leftarrow 0$ ;
   $i \leftarrow$  from_state;
  while  $i <$  number of actions of the plan do
     $i \leftarrow i+1$ ;
     $S_i \leftarrow$  execute_action ( $A_i$  of the plan on the  $S_{i-1}$  state)
     $v_i \leftarrow$  determine_relaxed_solution( $S_i$ );
  endwhile
  return array of values  $\langle v_0, v_1, v_2, v_3 \dots, v_n \rangle$ 
end;

```

Fig. 3. Algorithm of a function that determines the distances of each intermediate state from the initial state *initst* of a plan. The *from_state* variable is the number of the first state that must be considered in the plan.

The second phase has two steps. The first step determines the cost of each intermediate state of a solution plan provided by the completion phase. The algorithm in Figure 3 calculates, given an initial state and a solution plan, the estimated distance of each intermediate state from the initial state by using the FF-heuristic.

In order to determine the final and the intermediate state, the algorithm executes each action of the plan from the initial state (function *execute_action* in Figure 3).

The functions *create_relaxed_graph* and *determine_relaxed_solution* in Figure 3 are algorithms extracted from the FF-heuristic [9]. The function *create_relaxed_graph* is modified to expand the graph until a fixpoint is found.

The algorithm of Figure 3 returns an array of costs that contains the distance estimation value of each intermediate state from the initial state. In an optimal or optimized plan, these values must increase constantly from the beginning to the end. Any value that is less than the value before it can indicate a hot point of replanning, because it indicates a possible return in the directed graph of search. This value and its respective state are called **returned**.

The main idea of the CASER method is to find a misplaced sub-plan by detecting three kind of potential states: a Returned State (**Sr**) which is a potential state of a misplaced sub-plan, a Back State (**Sb**) that would be the final part of the misplaced sub-plan; and a State with Misplaced action (**Sm**) that would be the initial part of the sub-plan. Therefore, the misplaced sub-plan would be formed by the actions between **Sm** and **Sb** states. Figure 4 summarizes the main idea of the CASER method.

Definition 1: (Returned State) For a plan with $\langle S_0, S_1, S_2, S_3, \dots, S_n \rangle$ intermediate states, a **Sr** is an intermediary state where its value, obtained by a heuristic function, h^d , is less than the value of S_{i-1} .

This **returned** value indicates that its respective intermediate state is nearer to the initial state than the intermediate state immediately before it. The **returned** state and its **returned** value are indicated as S_r and v_r , respectively. The CASER method detects the first occurrence of a **returned** state in the solution plan. The algorithm that determines the **Sr** and v_r is presented in Figure 5. The symbol **Sr** just indicates that this point can be a part of a misplaced segment.

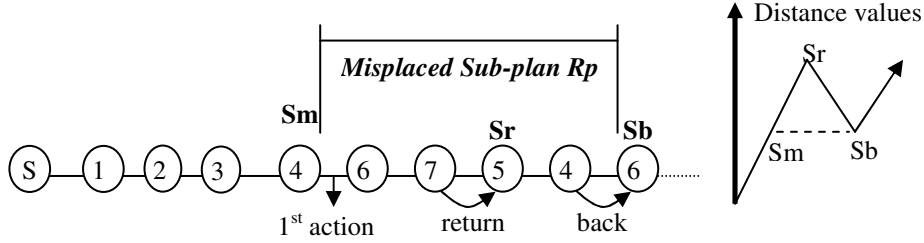


Fig. 4. A solution plan with the intermediate state represented as circles with their respective values and a misplaced sub-plan.

```

function determine_Sr (< $v_0, v_1, v_2, v_3, \dots, v_n$ >, from_state)
    retv  $\leftarrow$  0;
    i  $\leftarrow$  from_state;
    while (retv=0) and (i<n) do
        i  $\leftarrow$  i+1;
        if  $v_i < v_{i-1}$  then retv  $\leftarrow$  i;
    endwhile
    return retv;
end;

```

Fig. 5. Algorithm to detect the first occurrence of Sr and vr in a plan from the State from_state. The variable retv stores the position of the Sr state in the solution plan.

After detecting a Sr , the CASER method tries to detect the next intermediate state that continues to pursue the goal. This intermediate state, called Sb , and its respective value vb , indicate for the CASER method the point where the plan back to converge directly to the goal after the Sr state. The back State (Sb) and its value (vb) is determined as follows:

Definition 2: (Back State) Given a plan with $\langle S_0, S_1, S_2, S_3, \dots, S_n \rangle$ intermediate states, and a state Sr , a Sb state is a state S_i , with $i > r$, where its value vb , obtained by a heuristic function, h^d , is more than or equal to the S_{i-1} value.

The idea of the CASER method is to create an alternative plan that bridges the correct segments of the solution with the segment of the plan after the state Sb . This alternative plan will substitute a misplaced sub-plan, called Rp , which the last state is the Sb State. Figure 6 presents the algorithm that detects the Back State and its value.

In order to determine the beginning of the misplaced sub-plan Rp , the CASER method finds a state with the highest value among all states before Sr but less than the vb value. This state, called Sm , becomes the initial state of a possible misplaced sub-plan, and its value is indicated as vm . Formally, Sm can be defined as follows:

Definition 3: (State with misplaced action) Given a plan with $\langle S_0, S_1, S_2, S_3, \dots, S_n \rangle$ intermediate states, and a state Sr and the vb value, a Sm state is a state S_i , with $i < r$, where its value vm is the highest value among all other intermediate states $\langle S_0, S_1, S_2, S_3, \dots, S_{r-1} \rangle$ and less than vb value.

```

function determine_Sb (<v0, v1, v2, v3..., vn>, r)
backv ← 0;
i ← r;
while (backv=0) and (i<n) do
    i ← i+1;
    if vi≥vi-1 then backv ← i;
endwhile
return backv;
end;

```

Fig. 6. Algorithm to detect the first occurrence of **Sb** and **vb** in a plan from the position **r**. The variable backv stores the position of the **Sb** state in the solution plan.

```

function determine_Sm (<v0, v1, v2, v3..., vn>, r, b)
misv, i ← 0;
for i ← 0 to r-1 do
    if (vi>misv) and (vi<vb) then misv ← vi;
endfor;
return misv;
end;

```

Fig. 7. Algorithm to detect the first occurrence of **Sm** and **vm** in a plan from the first state to the position **r-1**. The **Sm** is that state that has the highest value less than **vb** value. The variable misv stores the position of the **Sm** state in the solution plan.

The algorithm that determines the position of **Sm** and **vm** in the solution plan is given in Figure 7. In fact, the CASER method considers the **Sm** as a state immediately before a possible sub-plan with misplaced actions and that must be replanned.

The **Rp** sub-plan, then, is formed by the actions between **Sm** and **Sb**. The **Sm** can be considered the initial state and the state **Sb** as the final state of the **Rp** sub-plan.

However, the actions of the sub-plan **Rp** can not be misplaced by themselves, and the **returned** value can be caused by any other action before **Sm**. To determine if the **Rp** sub-plan really contains misplaced actions, the algorithm *det_costs*, presented in Figure 3, is applied for the **Rp** sub-plan. If there is any returned value in the **Rp** sub-plan, it becomes a potential misplaced sub-plan that must be replaced. If the **Rp** does not contain any returned value by itself, then the initial state of the **Rp** sub-plan is *backtracked*. The new initial state becomes the state immediately before the state **Sm**. The algorithm *det_costs* is applied to this new **Rp** sub-plan in order to determine if it contains a returned state or not. The state backtracking continues until the initial state is reached or a misplaced sub-plan is detected.

When a misplaced sub-plan **Rp** is determined, the CASER method starts the replanning process by calling the modified version of the FF planning system to create an alternative plan to the **Rp** sub-plan. This alternative plan cannot have the same first action and must have fewer actions than the **Rp** sub-plan. The objective of changing the first action is to force the planning system to find an alternative solution. If no other alternative plan is found, the CASER backtracks the initial state of the **Rp** sub-plan and tries again. The backtracking continues until the initial state of the plan is found or until a specific number, *maxbktk*, of backtrackings are performed.

```

function CASER_replan(plan, initst, from_state, maxtime, maxbktk)
<v0, v1, ..., vn> ← det_costs(plan, initst);
r ← determine_Sr(<v0, v1, ..., vn>, from_state);
if r < 0 then
    b ← determine_Sb(<v0, v1, v2, ..., vn>, r);
    m ← determine_Sm(<v0, v1, v2, ..., vn>, r, b);
    Rp ← the sub-plan between Sm and Sb;
    bktk ← 0; // controls the number of backtrackings
    i ← m;
    isRp ← false;
    while (time < maxtime) and (i > 0) and (bktk < maxbktk) and not(isRp) do
        <vp0, vp1, ..., vpn> ← det_costs(Rp, Si);
        if determine_Sr(<vp0, vp1, vp2, ..., vpn>, 0) = 0 then
            isRp ← true;
        else
            i ← i-1;
            Rp ← plan between Si and Sb;
        endif;
    endwhile
    if isRp then Rp ← call modified_FF (Si, Sb) with
        - changing the 1st action of Rp;
        - maximal_length ← b-i-1;
        - time < maxtime;
    if Rp = null then isRp ← false;
    endif;
if not(isRp) then
    from_state ← b;
    else
    substitute Rp in plan between Si and Sb;
    endif
if time < maxtime then plan ← CASER_replan(plan, initst, from_state);
return plan;
end;

```

Fig. 8. The complete CASER second phase algorithm. It needs a solution plan, an initial state (*initst*) and the first position of the plan that will be considered (*from_state*). It also needs the maximal time and the maximal number of backtracking (*maxbktk*).

If there is no backtracking and no alternative solution is found, the CASER method continues to detect points of replanning after the **Sb** State.

On the other hand, if an alternative plan is found, it substitutes the **Rp** sub-plan and the CASER method continues to detect points of replanning after the **Sb** state until the final state is reached. The complete CASER algorithm is presented in Figure 8. Figure 9 shows the behavior of the backtracking for an example in the Blocks World domain.

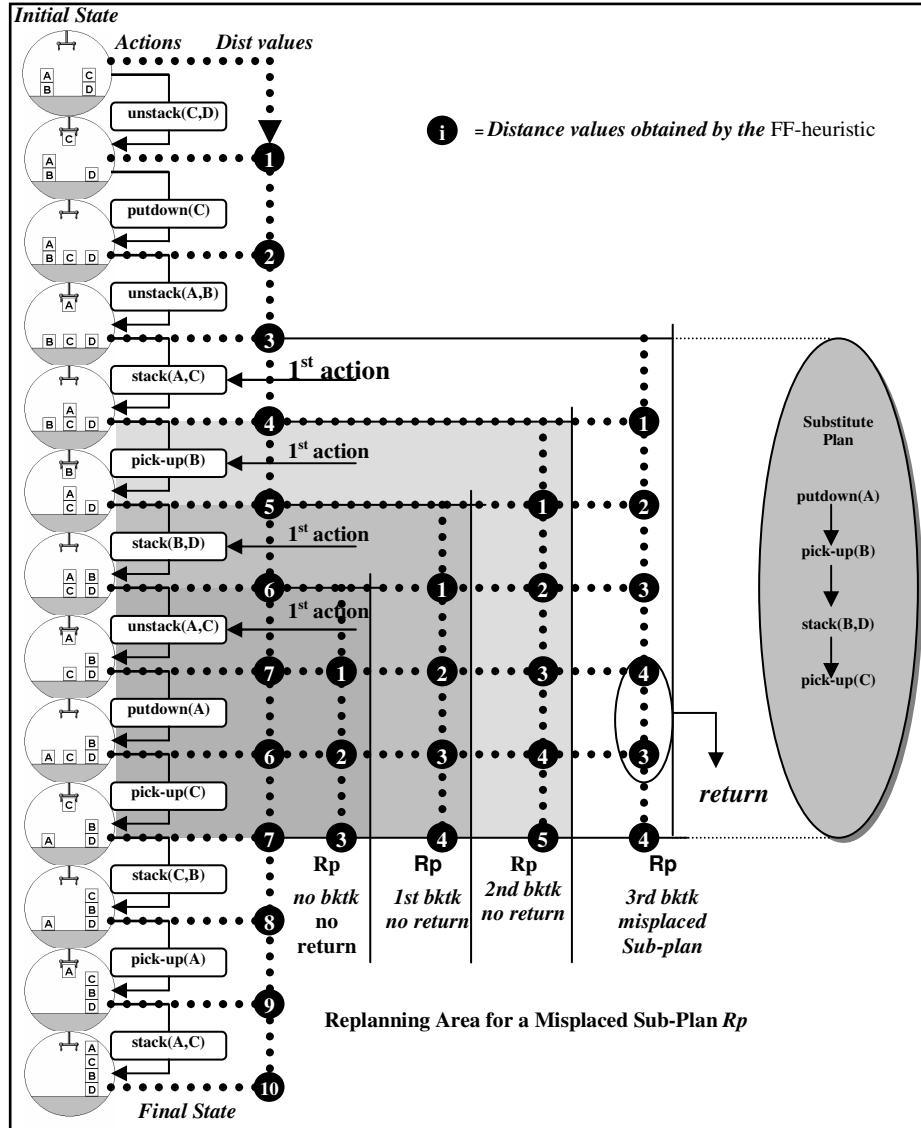


Fig. 9 - Example of the 2nd phase of the CASER applied in a plan of the Blocks World domain.

The replanning process can be fast due to two factors. One is the use of a modified FF planning that permits the system to work with a complete state as goal. The other is that the modified FF must find a plan with a limited length what can reduce the time to find long solutions. However, the numbers of backtrackings and the complexity of a solution plan can turn the CASER method into a time consuming process.

Because of this time consuming risk, the CASER method allows the user to define the maximal time that the method can try to improve the solution quality and also the maximal number of backtrackings that the process can do.

Table 1 – Results of the CASER method applied to some STRIPS domains. The time is in milliseconds and #act means number of the actions in the plan solution

DriverLog Dmain Problems IPC'02	CASER METHOD					LPG system reference results in quality track	
	1st phase		2nd phase	Final results		#act	
	time	#act	time	time	#act		
dlog-01	12	11	70	82	9	7	
dlog-02	24	19	65	89	19	20	
dlog-03	13	33	231	244	20	12	
dlog-04	27	31	119	146	28	16	
dlog-05	31	22	105	136	21	18	
dlog-06	300	22	1835	2135	16	17	
dlog-07	4846	30	125	4971	26	13	
dlog-08	340	32	776	1116	30	22	
dlog-09	21	31	120	141	31	23	
dlog-10	10978	28	90	11068	23	17	
Logistic domain problems IPC'00	CASER METHOD					FF system reference results	
	1st phase		2nd phase	Final Results		#act	
	time	#act	time	time	#act		
LOGISTICS-04-0	14	26	68	82	26	49	
LOGISTICS-05-0	23	45	175	198	41	49	
LOGISTICS-06-0	19	33	70	89	33	25	
LOGISTICS-07-0	45	63	851	896	53	36	
LOGISTICS-08-0	66	52	170	236	43	31	
LOGISTICS-09-0	65	56	205	270	49	36	
LOGISTICS-10-0	200	86	425	625	74	46	
Blocks World Domain Problems IPC'00	CASER METHOD					Reference Results	
	1st phase		2nd phase	Final Results		FF	HSP
	time	#act	time	Time	#act	#act	#act
BLOCKS-04-0	11	6	70	81	6	6	6
BLOCKS-05-0	2	12	105	107	12	12	12
BLOCKS-06-0	19	26	255	274	12	16	12
BLOCKS-07-0	22	40	1640	1662	22	20	26
BLOCKS-08-0	52	26	1665	1717	26	18	18
BLOCKS-09-0	30	42	1511	1541	30	30	40
BLOCKS-10-0	22	34	9285	9307	34	34	98
BLOCKS-11-0	44	38	3296	3340	34	32	44
BLOCKS-12-0	35	42	5600	5635	42	36	36

4 Empirical Tests

The complete CASER method was tested in some STRIPS-model domains for some retrieved cases obtained by the FAR-OFF Case-Based Planning system [14]. The cases retrieved by the FAR-OFF system are from a case base with random cases generated by a case base seeder [15].

The STRIPS domains used in the tests are retrieved from the IPC'00 (International Planning Competition) [3] and IPC'02 [11]. The problems considered in the tests are the first one proposed in those competitions for each domain.

The tests, in Table 1, show the performance and results of the CASER method when applied to a retrieved case in Blocks World, Logistic and DriverLog domains. The solutions of the LPG [7], FF [9] and HSP [5] planning systems in IPC'02 [11] and IPC'00 [3] planning competitions are used as a comparative result of the quality of the solution. They had one of the best results in such domains and provide a good validation for CASER results.

The tests were performed in a Windows® environment on a Pentium® III 450 MHz computer with 512 Mbytes of RAM memory and considering a number of backtrackings limited to 15. As stated by Tonidandel and Rillo [15], in a general frame, the use of unlimited backtracking is not relevant for the quality solutions, and it still spends much more time than when a limited of 15 backtracking is imposed. In addition, there is no imposition of time limit, even though the CASER method allows that the user specifies a maximal time that can be used to improve the solution. The definition of a limited time for replanning does not affect the planning problem solution; it only reduces the time for the improvement of the solution quality.

The second phase of the CASER method reduces the easy-to-generate solution in most of the situations and for some of them it reduces up to the optimal plan length. There is some quality improvement from the easy-to-generate solution in the tests, as can be seen in BLOCKS 6.0 and BLOCKS 7.0 problems where the replanning phase reduces the easy-to-generate solution in about 50%.

The tests results show that the CASER method is effective for a case-based planning system (e.g. the FAR-OFF system) in all domains considered in the tests. Considering the solution quality provided by the generative planning systems, the CASER returns the best solution in about 35% of the results. These results do not empirically prove that the CASER is better than generative planners; they just show that the CASER method works suitably. The CASER method is just a part of an entire case-based planning system and, therefore, it can not be compared with an entire generative planner in terms of solution quality and time performance.

The time spent by the second phase of the CASER method depends on the number of returned values and misplaced sub-plans. Since the more actions a plan have, the higher is the possible number of returned values and misplaced sub-plans. Therefore, the CASER method takes more time to adapt a solution plan with more actions.

5 Discussion

Considering the conditions of the tests, the CASER method is a very promising tool of case adaptation. The tests were performed in hard conditions. The cases retrieved by the FAR-OFF system are not very similar to the possible solution because they are retrieved from a case base constituted by random cases, from where retrieved cases are not necessarily good similar cases. A random case is created by a Case-Based Seeding process that finds random initial and goal states and applies a generative planning system to find a plan between the states [16].

In fact, even with a retrieved case from a case base filled of random cases and with the same knowledge used by generative planners, the CASER can return some best solutions, about 35% of the solutions in the tests. The results only show that the

CASER method can work well with the same knowledge provided to a generative planner.

However, there are some efficiency bottlenecks that must be solved. One of these bottlenecks is the behavior of the CASER method that tries to replace an optimal sub-plan in some situations (e.g. BLOCKS-5.0 test). It is caused by the FF-heuristic, which is not appropriate to detect whether a plan is optimal or not. Because of this, the time of replanning applications is higher than necessary occasionally. Other heuristics or some new verification methods will be considered in the future.

The CASER method does not guarantee to find an optimal solution of a specific problem because it only analyses the intermediate states and the final state that were provided by the solution plan. The method is restricted to reduce the plan found by an easy-to-generate phase. For instance, if an optimal solution of a specific problem is to consider another final state than that provided by the easy-to-generate plan, the CASER method is unable to find this optimal solution. This limitation of the CASER method is responsible for the difficulty that this method presented in the DriverLog domain. This limitation will be analyzed in our future research.

This version of CASER method is designed to improve the solution quality by decreasing the number of steps of the solution plan. However, this method is not restricted to the solution length, but it can also be extended to deal with complex domains with resources and time.

In order to work in domains with numerical variables, a heuristic function that estimates the cost of a state by taking in consideration these numerical values must be available. The metric-FF planner system [8] provides a metric-heuristic that can be used to extend the CASER method to numerical domains. However, it will be left for further studies.

6 Conclusion

The CASER method, presented in this paper, is an domain-independent method to adapt retrieved cases. The method uses the FF-heuristic and a modified version of the FF planning system to determine and find a solution from a retrieved case by a recursive quality improvement algorithm with anytime behavior.

The first phase of the CASER method just finds an easy-to-generate solution whereas its second phase modifies some misplaced sub-plans and bad segments of the entire solution in order to improve the solution quality.

The empirical tests show that the CASER method is a promising adaptation process that can find good quality solutions for case-based planning systems. The most important feature of the CASER method is that it does not require any additional knowledge neither any other process like invariants extractions or learning algorithms. In fact, the CASER is a method that can be applied to any planning domain without any specification of adaptation rules or extra domain dependent information.

The method proposed in this paper focuses on the plan length and does not consider time or resources. This method will be extended in the future to support domains with such features.

References

1. Ambite, J. L.; Knoblock C. A. Planning by Rewriting. In: *Journal of Artificial Intelligence Research*, 15, (2001), 207-261.
2. Au, T.; Muñoz-Avila, H.; Nau, D. S. On the Complexity of Plan Adaptation by Derivational Analogy in a Universal Classical Planning Framework. In: *Craw, S.; Preece, A. (Eds.) Proceedings of the 6th European Conference on Case-Based Reasoning - ECCBR-2002*. Lecture Notes in Artificial Intelligence. Vol 2416. Springer-Verlag. (2002) 13-27.
3. Bacchus, F. AIPS-2000 Planning Competition Results. Available in: <http://www.cs.toronto.edu/aips2000/>. (2000)
4. Blum, A.; Furst M. Fast Planning through Planning Graphs Analysis, *Artificial Intelligence*, 90, (1997) 281-300.
5. Bonet, B; Geffner, H. Planning as Heuristic Search. *Artificial Intelligence*. 129 (2001) 5-33.
6. Gerevini A.; Serina, I. Fast Adaptation through Planning Graphs: Local and Systematic Search Techniques. In: *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling AIPS'00*. AAAI Press. (2000).112-121.
7. Gerevini A.; Serina, I. LPG: A Planner Based on Local Search for Planning Graphs with Actions Costs. In: *Preprints of the 6th International Conference on Artificial Intelligence Planning and Scheduling AIPS'02*. AAAI Press. (2002) 281-290.
8. Hoffmann J. Extending FF to Numerical State Variables, in: *Proceedings of the 15th European Conference on Artificial Intelligence*, Lyon, France (2002)
9. Hoffmann, J.; Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*. 14 (2001) 253 – 302.
10. Koenig, S. , Furcy, D., Bauer, C. Heuristic Search-Based Replanning. In: *6th Proceedings of the International Conference on Artificial Intelligence on Planning and Scheduling (AIPS-2002)*. Toulouse. (2002).
11. Long, D.; Fox, M. The 3rd International Planning Competition - IPC'2002. Available in <http://www.dur.ac.uk/d.p.long/competition.html>. (2002).
12. Nebel, B. ; Koehler, J. Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial Intelligence*, n. 76, p.427-454,. Special Issue on Planning and Scheduling (1995).
13. Tonidandel, F.; Rillo, M. An Accurate Adaptation-Guided Similarity Metric for Case-Based Planning In: *Aha, D., Watson, I. (Eds.) Proceedings of 4th International Conference on Case-Based Reasoning (ICCBR-2001)*. Lecture Notes in Artificial Intelligence. vol 2080. Springer-Verlag. (2001) 531-545.
14. Tonidandel, F.; Rillo, M. 2002. The FAR-OFF system: A Heuristic Search Case-Based Planning. In: *Proceedings of 6th International Conference on Artificial Intelligence on Planning and Scheduling (AIPS-2002)*. Toulouse. (2002).
15. Tonidandel, F.; Rillo, M. Improving the Planning Solution Quality by Replanning. In: *Anais do VI Simpósio Brasileiro de Automação Inteligente*. Bauru São Paulo (2003).
16. Tonidandel, F.; Rillo, M. A Case base Seeding for Case-Based Planning Systems. In: *Lemaitre, C., Reyes, C., Gonzales, J (Eds.) Proceedings of 9th Ibero-American Conference on AI (IBERAMIA-2004)*. Lecture Notes in Artificial Intelligence. vol 3315. Springer-Verlag. (2004) 104-113.
17. Veloso, M. Planning and Learning by Analogical Reasoning. Lecture Notes in Artificial Intelligence, Vol 886. Springer-Verlag. (1994).