Solving a Goal-planning task in the MASH project

Jean-Baptiste Hoock, Jacques Bibai

Solving a Goal-planning task in the MASH project

Jean-Baptiste Hoock
TAO (Inria)
LRI, UMR 8623(CNRS - Univ. Paris-Sud), Bat 490 Univ. Paris-Sud
91405 Orsay, France
Email: hoock@lri.fr

Jacques Bibai
Independent researcher
Yaounde, Cameroon
Email: jacques.bibai@gmail.com

***Abstract*—The MASH project is a collaborative platform with the aim to experiment different methods in an unknown environment of large size. The application is a goal-planning task in a 3D video game where runs are expensive. Moreover, there is no prior knowledge, the decisions have unknown semantics, observations on the environment are partial and of big size and accomplishing the task by taking random decisions always requires a very long run. So, solving this task is a big challenge. In this paper, we extend Monte-Carlo Tree Search, which has been proved very effective for applications in which simulating is easy and fast, to contexts in which there are only "real" expensive runs. This generic approach combines Clustering and Monte-Carlo Tree Search.**

***Keywords*-MASH; goal-planning; clustering; Monte-Carlo Tree Search;**

I. Introduction

Many specialized methods (e.g. in Data Mining [8] [3]) have proven the efficiency in a lot of specific domains. Ensemble methods such as Adaboost [4] try to take advantage of each specific method by building more collaborative approaches. In this context, the MASH project [1] was created. The aim of this project is to create new tools for the collaborative development of large families of feature extractors in order to start a new generation of learning software with great prior model complexity. Targeted applications of the project are (i) classical vision problems and (ii) goal-planning in a 3D video game and with a real robotic arm.
For those applications, features are the result of one or several methods used to extract relevant information. The goal-planning application starts with a given initial state, a set of decisions and a set of methods. The state is the environment in which the avatar (for the 3D video-game) or the robot-arm are located. A decision is applied in order to go from a given state to another state. When methods are applied to a state, observations are generated. From these observations, the best decision is selected and then applied. A new state, a reward and an information about the final state are generated. A final state is the situation where the state doesn't change anymore whatever the applied decision. These processes are repeated from the initial state until one final state is reached. The goal is to maximize the sum of all rewards given from an initial state to a final state.
The problem is to find best decisions in order to reach the goal. Some algorithms such as Monte-Carlo Tree Search [6], Nested Monte-Carlo Search [1], Rollout Classification Policy Iteration [5] have been proposed. However, those algorithms can not be applied because either prior knowledge about decisions, environment and/or methods are crucial, either the complexity of the problem should be smaller (the number of features in the observation is not too large or the reward is informative...) either a model of the problem is required in order to simulate it. The MASH goal-planning problem is hard and there is no prior knowledge or model.
This paper proposes a new generic algorithm in order to solve the goal-planning problem in the 3D video-game. The first part presents the MASH problem, the second part presents the algorithm and the third part presents experiments and discussions on 2 different problems and the last part is the conclusion.

[1]http://mash-project.eu/wiki/index.php/About_the_MASH_project. MASH means MAssive Sets of Heuristics

II. the MASH application

We propose to solve a goal-planning problem in the 3D video-game. The environment is first described, then methods, observations, decisions and rewards. Finally, the best solution of this problem is given.

1) Environment: The environment is a square various size room with globally grey textures. The square room enclosed by 4 walls, contains an avatar and 2 flags. One of the two flags is red and the other flag is blue (See Fig. 1).

2) Methods: Methods are heuristics. More specifically, a heuristic is an image processing operator. The heuristic is applied on the view of the 3D avatar. 3 heuristics are used :

- *red* which indicates if a pixel (an element of the view of 3D avatar) is red or not
- *blue* which indicates if a pixel is blue or not
- *identity* which transforms the value of the pixel to a gray-scale value.

Each heuristic gives around 100,000 features. In the observation, the solver ignores what heuristic provides the feature and has no notion of heuristics.

3) Building an observation: The cumulative number of features given by the 3 heuristics *red*, *blue* and *identity* are $3 \times 100,000$ features but it's too huge. In order to reduce the size of observation space, 10,000 features are randomly selected over the 300,000 features. The observation given to the solver will be always made with these 10,000 features.

4) Decisions: There are 4 decisions :

- 0 (which means "go forward")
- 1 (which means "go backward")
- 2 (which means "turn right")
- 3 (which means "turn left")

Note that the solver doesn't know the meaning of decisions.

5) Rewards:

- Hit a wall: -1
- Touch the first time the blue flag: +5
- Touch (or hit) another time the blue flag: 0
- Hit the red flag without touched the blue flag: -5
- Hit the red flag after having touched the blue flag: +10
- Else: 0.

This definition of rewards is not informative, because most often, the avatar moves without touching anything and receives a reward of 0.

6) Goal: A final state is reached if and only if once having touched the blue flag, the red flag is touched. The best cumulative reward is 15, by first touching the blue flag and then touching the red flag without hitting wall. We can distinguish 2 subgoals (i) touch the blue flag and then (ii) touch the red flag.

III. The Algorithm

As it has been seen in the section II, there is no information that can help us to infer knowledge for solving more efficiently the problem. But we can simulate several scenarios in order to build a knowledge. After having defined Notations, 2 pieces of knowledge built by unsupervised methods are presented: (i) Categorization of Decisions and (ii) Clusterization of features. Then the Policy and the Algorithm, which uses built knowledge and learns how to reach the goal, are shown.

A. Notations

An observation (or a state) is denoted by s. From a given state s_t, the state reached after the application of one decision on the state s_t is denoted by s_{t+1}. The set of states is denoted by S, the set of features by F, the set of decisions by U, a feature by f and a decision by u. Let r_t the reward and $finished_t$ the Boolean variable which states if the state s_{t+1} is terminal (or final), $MakeTransition$ denotes the transition function (i.e. the application of the decision u_t on the given state s_t) and returns the new state s_{t+1}, the boolean $finished_t$ and the reward r_t : $(s_{t+1}, finished_t, r_t) = MakeTransition(s_t, u_t)$.
When a transition is applied on a state s_t, features of the state s_t are activated or deactivated. Given a state s_t, $F_a(t)$ denotes the set of active features of this state.

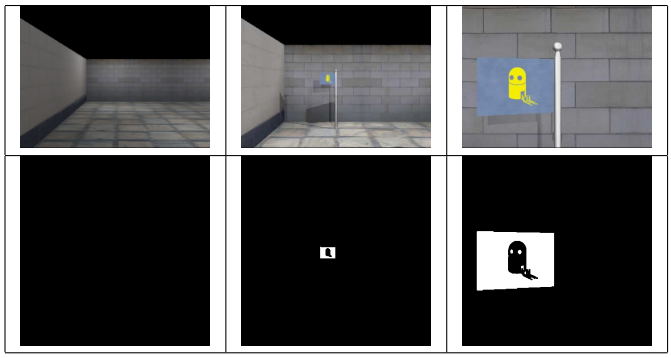

Figure 1. **Top:** 3 different views of the avatar. **Bottom:** Images resulting of the application of the *blue* heuristic. **Left:** The blue flag is not seen, no features are activated. **Center:** The blue flag is far, few features are activated. **Right:** The blue flag is closer, more features are activated

B. Learning a Categorization of the Decisions

At the beginning, the algorithm (or solver) has no idea of decisions. But by doing simulations, the impact on the state can be studied.

Formally, a decision u is :

- periodic if $\exists k \in \mathbb{N}^*, \forall s_t \in S \; s_{t+k} = s_t$
- stationary if $\forall s_t \in S \;\; \exists K \in \mathbb{N}^*, \forall j \in \mathbb{N}^* \;\; s_{t+K} = s_{t+K+j}$
- final if $\forall s_t \in S \; s_{t+1}$ is a final state.

with s_{t+x} the state s_t obtained after applying x times the decision u.

A decision u^- is the inverse of a decision u^+ if $s_t = s_{t+2}$ with $(s_{t+1}, \_, \_) = MakeTransition(s_t, u^+)$ and $(s_{t+2}, \_, \_) = MakeTransition(s_{t+1}, u^-)$ (_ denotes an ignored information.).

First, from a given state s_t, we categorize a decision u by applying it several times. Let k the number of times the decision u has been applied.

1) A first case is to reach a final state by applying the first time ($k = 1$) the decision u. The decision is categorized as a final decision.
2) A second case is the decision has no more impact on the state ($s_{t+k-1} \approx s_{t+k}$). The decision u is categorized as a stationary decision.
3) A third case is the initial state s_t is seen a new time ($s_{t+k} \approx s_t$). The decision u is categorized as a periodic decision.
4) A fourth case is the initial state s_t is seen a new time after applying the decision u one time ($k = 1$) and another decision $u^* \neq u$ one time, too. Decisions u and u^* are categorized as inverse decisions.

Pieces of information will be very useful for simulating more efficiently.

C. Learning a Clustering of the Features

The algorithm ignores what heuristic provides the feature. The idea is to clusterize relevant features that can help the

Function $CluVo()$
Let $clusters$ a static variable containing all clusters of features.
Initialization: each cluster is composed of one feature.
for $iteration$ in $1.. + \infty$ **do**
  $ldb = GenerateNewDatabase(iteration)$
  $nbClus = size(clusters)$
  $clusters = Clusterize(ldb, clusters)$
  if $nbClus \neq size(clusters)$ **then**
    $nbClus = size(clusters)$
  else
    break
  end if
end for

Figure 2. the complete Clustering (generation of the collection + clustering).

algorithm to find goal. To solve this issue, we propose an algorithm of clustering (Fig. 2) called CluVo for finding correlated features. CluVo makes no assumption about the number of clusters and is completely unsupervised (Features are not labelled.). In the MASH problem, it corresponds to find, optimally, 2 clusters : "Features which come from the *blue* heuristic" and "Features which come from the *red* heuristic".

In the clustering, we distinguish 3 parts : (i) The generation of collection (or database) of lists of features by doing simulations - (ii) Clustering features - (iii) The Metric which is the correlation between 2 features.

1) Simulations for the generation of the collection: For the generation of the collection, following the number of iterations, simulations are done differently : For the first 3 iterations, decisions are chosen randomly; but from the 4^{th} iteration, decisions are chosen randomly or by vote (See Section III-D4). Given a cluster, the vote is very efficient for finding new features but needs several updates of (4) before to be used efficiently.

2) Clustering features: The approach of our algorithm is hierarchical. There exist 2 variants : (i) The divisive (or top-down) clustering [8] starts from one cluster and gradually splits clusters. (ii) The agglomerative (or bottom-up) clustering [3] starts from a set of small clusters and then aggregates them. Because of the system of vote (4), a bottom-up clustering is surely better. The shown algorithm is so agglomerative. Therefore, each feature corresponds to a cluster at the starting of the CluVo Algorithm.

3) Metric: Let ldb the collection of the list of features. Let $f1$ and $f2$ two features of the set of features obtained thanks to a uniform draw.

Let $Db(f) = \{list \in ldb; f \in list\}$ with f a feature and $Db(f1, f2) = Db(f1) \cap Db(f2)$

Let $Card(X) = \#X$ the number of elements of the set X.

The features $f1$ and $f2$ are significantly present (or correlated) if $\#Db(f1, f2)) > 5$ and $3 \times \#Db(f1, f2) >= max(\#Db(f1), \#Db(f2))$.

The numbers 5 and 3 have been fixed after experiments.

Function $Clusterize(ldb, lclus)$
Let P_C1 a static parameter fixed to 10^4
Let P_C2 a static parameter fixed to 10^7
Let $nbClus$ the number of clusters initialized to the size of ldb
$nbTry = \min(\max(P\_C1, nbClus \times nbClus), P\_C2)$
while $nbTry > 0 \wedge nbClus > 1$ **do**
  Decrement $nbTry$
  $(f1, f2) = SelectTwoFeatures(lclus, ldb)$
  Let $clus1$ the cluster of features to which the feature $f1$ belongs
  Let $clus2$ the cluster of features to which the feature $f2$ belongs
  if $clus1 \neq clus2 \wedge f1$ and $f2$ are correlated (See Section III-C3) **then**
    $Merge(clus1, clus2, lclus)$
    $nbClus = size(lclus)$
    $nbTry = \min(\max(P\_C1, nbClus \times nbClus), P\_C2)$
  end if
end while
Return $lclus$

Figure 3. The Clustering phase. $SelectTwoFeatures(lclus, ldb)$ selects 2 features by using either the list of clusters $lclus$ or the collection of the list of features ldb. $Merge(clus1, clus2, lclus)$ creates a new cluster into the set $lclus$ by merging $clus1$ and $clus2$ and then removes them from the set $lclus$.

Function $\pi(s)$
Let ga a static variable containing a tree of goals
Let gn the current node and $root$ the root node of ga
Let u a static variable containing the decision
Let l a static variable containing the remaining number of times that the decision u will be repeated
if gn is $root \vee IsReached(gn, s)$ **then**
  $gn = ChooseNextNode(gn)$
  $l = 0$
end if
if $l == 0$ **then**
  Build a new Macro-Decision (u, l)
- $u = GetDecision(s, gn)$
- $l = GetMacroDecisionLength(s, u, gn)$

end if
Decrement l
Return u

Figure 4. A simplified version of our Policy. The primitive $GetDecision$ is a function which returns a decision and $GetMacroDecisionLength$ a function which returns the number of times a decision should be repeated. The function $GetDecision(s, gn)$ returns either a final decision if the rule of gn is $GOAL\_FINISHED$, either a decision by vote if some features given by the node gn are activated in the state s, or else a decision for exploration.

D. Policy

The Policy $\pi(s)$ is the function which gives the best decision to apply from the state s.

The best Policy should be able to go to the blue flag, first and then to go to the red flag. In order to realize this task, the Policy must be able to switch to the first subgoal (reach the blue flag) to the second subgoal (reach the red flag).

1) Memory for switching subgoals: A memory is necessary so as to memorize if the blue flag has been touched or not.
The memory is modeled by a tree [2] of subgoals called ga. A node of the tree corresponds to a subgoal, an edge the decision of reaching a subgoal, the root means the top of the tree and a leaf is a node which has no next subgoal. A node contains (i) a rule of goal (ii) a list of features to activate (given by a cluster) with optionally, a number of features to activate (iii) the number of simulation, (iv) the

[2]a definition can be found in http://en.wikipedia.org/wiki/Tree_(graph_theory)

cumulative reward of its last simulation and (iv) the average of cumulative rewards. 2 rules of subgoals are defined :

- $GOAL\_APPROACH$ reach a state whose an amount quantity of features are activated
- $GOAL\_FINISHED$ the next decision should lead to a final state.

To go from a subgoal gn_1 to another subgoal gn_2, a first step (called $IsReached(gn_1, s_t)$ in Fig. 4) is to estimate if the state s_t accomplishes the subgoal or not. If the answer is positive, a second step (called $gn_2 = ChooseNextNode(gn_1)$ in Fig. 4) is to choose from the current subgoal gn_1 the next subgoal to realize. The next chosen node is the child node of gn_1 which maximizes the number of simulations.

2) Useful Macro-Decisions: A Macro-Decision [7] is a decision repeated several times. A Macro-Decision is modeled by the couple (u, l).

- u is the decision
- l is the number of times that the decision is repeated.

A decision can be useless because the application of the decision on the state s_t has no more impact (stationary decision), or the decision is the inverse of the last decision, or when the avatar has made a complete turn, it doesn't need to continue to turn (periodic decision). The set of useful decisions for a given state s_t is denoted U_{ok}.

3) Exploration for finding a subgoal: In the MASH application, the environment is partially observable. The environment must be explored. But because of the size of the environment, classical Random Search algorithm can not be applied. Macro-decisions is a tool that can help to explore more rapidly the environment[7]. To touch a flag, a first step consists to find it.

To find the flag, some decisions are more useful than others (e.g. the decision to turn is good). During the exploration, when there is no activated features in the state s_t and then a given decision activates features in the state s_{t+1}, this decision can be considered as a good one. These decisions can be found statistically after a lot of explorations.

In order to choose good decisions for finding a subgoal, 2 variables are defined. Let $Fclus$ a set of features

$$disc(Fclus, u) = \#\{s_t, F_a(t) == \emptyset \wedge F_a(t+1) \cap Fclus \neq \emptyset\} \quad (1)$$

$$disc(u) = \#\{s_t, F_a(t) == \emptyset \wedge F_a(t+1) \neq \emptyset\}. \quad (2)$$

The formulas can be updated at each simulation after a $MakeTransition$.

Then for a given node gn, the decision for exploration is given by

- Let obj the set of features given by the node gn.
- one time over $1 + \sum_{u \in U} disc(u)$, return a random decision.
- with a probability of $disc(obj, u)/disc(u)$, if $u \in U_{ok}$ then return the decision u
- else return a random decision

When features have been activated, decisions can be now chosen by using these activated features.

4) Vote for reaching a subgoal: Given a set of features, a set of decisions and from an initial state with some active features, a sequence of decisions which leads to the goal is searched. A decision can activate new features or deactivate features. Here, the goal is to activate a maximal number of features of the set. In order to do this, the idea is that each of active features votes for a decision which increases the number of active features of the set.

For each couple $(f, u) \in F \times U$ is associated a score $score(f, u)$. A vote is defined by

$$Vote(f) = Argmax_{u \in U} score(f, u). \quad (3)$$

Let $F_v(u) = \{f \in F_a(t) \cap FClus, u == Vote(f)\}$ the set of active features of the state s_t belonging to a cluster (or set of correlated features) $FClus$ and which vote for the decision u.

Then, the decision by vote is given by $Argmax_{u \in U_{ok}} \#(F_v(u))$.

We can see like this : For each decision u, we count the number of active features which vote for the decision u. The decision which has received the most of votes is selected.

The function $score$ gives a cumulative number of features of one cluster which have been activated from a sample of states. More precisely, Let (s_t, u, s_{t+1}) the state s, the decision u and the state s_{t+1} obtained immediately after applying the decision u on the state s.

Let $F_{a|Fclus}(t)$ the set of active features of the state s_t belonging to the cluster $Fclus$ ($F_{a|Fclus} = F_a(t) \cap Fclus$). Let $f_a \in Fclus$ an active feature of the state s. Then

$$score(f_a, u) = \sum_{s_t} (\#F_{a|Fclus}(t+1) - \#F_{a|Fclus}(t)). \quad (4)$$

The signification of the formula is to activate more features in the state s_{t+1} than activated features of the state s_t.

Note that $f_a \in F_a(t)$ but *maybe*, $f_a \notin F_a(t+1)$. In this case, the feature f_a has been deactivated when the decision u has been applied on the state s_t.

The function $score$ can be updated at any moment when a decision is applied.

E. Learning a Sequence of Goals

The categorization of decisions and the Clustering of features are unsupervised methods, the information of the reward has not been used. Now, for finding the goal, the reward is used and we build the tree ga. The proposed algorithm (Fig. 5) is a Monte-Carlo Tree Search [6] [2] on subgoals. The algorithm is called GMCTS (Goal Monte-Carlo Tree Search).

Function $Gmcts(maxSimulation)$
Let ga a static variable containing a tree of subgoals
Let $root$ the root of the tree ga.
for sim in $[\![1; maxSimulation]\!]$ **do**
  Let s initialized to the initial state.
  $(r_{cum}, finished, gn) = (0, false, root)$
  while $finished \neq true$ **do**
    $gn = ChooseNextGoal(gn)$
    $reached = false$
    while $reached \neq true \wedge finished \neq true$ **do**
      $u = GetDecision(s, gn)$
      $l = GetMacroDecisionLength(s, u, gn)$
      for j in $1..l$ **do**
        $(s, finished, r) = MakeTransition(s, u)$
        $r_{cum} = r + r_{cum}$
        $reached = IsReached(gn, s)$
        if $reached \vee finished$ **then**
          break
        end if
      end for{this loop is the application of the Macro-decision (u, l) on the state s}
    end while
  end while
  $UpdateTree(r_{cum}, gn)$ {Pieces of information in a node which are to update are : (i) the number of simulations (ii) the average of the cumulative rewards and (iii) the cumulative reward. All nodes visited during the simulation (going from the node gn to $root$) are updated.}
end for

Figure 5. Monte-Carlo Tree Search for searching goal. The argument $maxSimulation$ is a parameter of the GMCTS algorithm and denotes the number of simulations which will be done to build the tree ga. Macro-decisions are used in the GMCTS algorithm.

Function $ChooseNextGoal(gn)$
Let P_CG1 a static parameter fixed to 3
if $IsLeaf(gn)$ **then**
  Return $ChooseNewGoal(gn)$
end if
$mode = 1$
if $rand()\mod P\_CG1 \neq 0$ **then**
  $mode = 2$
end if
if $rand()\mod Size(Children(gn)) \neq 0$ **then**
  Return $ChooseBestGoal(gn, mode)$
else
  Return $ChooseNewGoal(gn)$
end if

Figure 6. The Policy of GMCTS.

The policy of the GMCTS is given by the Fig. 6

Given the node gn, the function $ChooseNewGoal$ creates a new child node of gn and returns it, whereas the function $ChooseBestGoal$ returns the child node of gn which maximizes either the average of cumulative rewards (mode 1) or the cumulative reward of its last simulation (mode 2).

IV. Experiments

A. The optimization of a draw of letters

For the genericity, the Algorithm has been applied on another application : optimization of a draw of letters. The longest word is a game where a draw of 10 letters is made and the goal is to produce the longest correct word according to a dictionary of around 300,000 French words[3]. But, in order to have interesting draws, letters are not drawn equiprobably. The goal is to find the best distribution of letters in order to have draws with long correct words.

[3]http://www.pallier.org/ressources/dicofr/dicofr.html

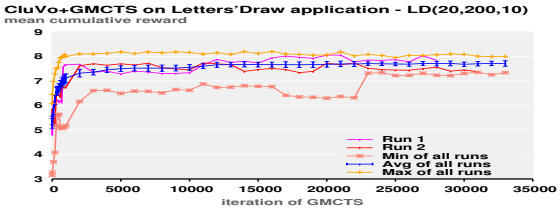

Figure 7. Results of GMCTS on LD(20,100,10). Run 1 and Run 2 correspond to 2 runs. The mean cumulative reward corresponds to the average length of longest words on draws of 10 letters.

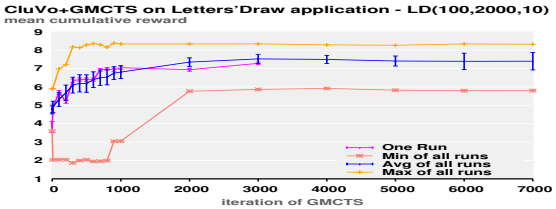

Figure 8. Results of GMCTS on LD(100,1000,10). The errorline of the average curve is the standard deviation computed on all remaining runs. The errorline of the curve "One Run" is the standard deviation computed through 100 evaluations.

State: The state is a vector of size $NbL \times 26$; each of the 26 groups contains NbL features. Each feature is 0 or 1. The state describes a distribution on words as follows: we randomly draw 10 letters, and we start by randomly draw a consonant - each consonant has probability proportional to the number of 1 in its features. Then, for each letter, we switch from vowel to consonant or from consonant to vowel with probability 95%; and letters are drawn among consonants or vowels with probability proportional to the number of features at 1. The initial distribution contains each letter one times.

Decisions: 27 decisions are possible. Each decision (except the decision "not add") consists in adding one letter; the feature which is activated among the NbL corresponding features is randomly drawn.

Reward: The reward is 0 if s is not terminal. When s is terminal, the reward is computed as the average length of longest words through $NbDraws$ draws.

Terminal State: Let Mal the maximal number of letters we can add in all. A state is terminal if either Mal letters have been added either the decision of "not add" has been taken.

For different variants of this problem, the letter's draw application will be denoted $LD(NbL, Mal, NbDraws)$. A difficulty of this testbed is that the reward is stochastic. Like in the Mash application, we work in a setting with no prior knowledge.

B. Results

For both applications, all tests have been made with 100 evaluations and all experiments have been exactly made with the same algorithm. No parameter has been modified.

1) Results on draw of letters: Results of clustering is given in Table I.

Table I
RESULTS OF CLUSTERING ON LETTER'S DRAW APPLICATION. THE ROW ITERATIONS IS THE NUMBER OF ITERATIONS OF CLUSTERING. THE ROW ERROR IS THE NUMBER OF RUNS WITH BAD CORRELATION(S). THE ROW INCOMPLETE IS THE NUMBER OF RUNS WHERE THE CORRECT NUMBER OF CLUSTERS HAS NOT BEEN FOUND. THE ROW SUCCESS IS THE NUMBER OF RUNS WITH COMPLETE CLUSTERING WITHOUT BAD CORRELATION.

Application	LD(20,200,10)	LD(100,2000,10)
Nb Runs	43	35
Iterations	between 3 and 4	3 (except one run with 4)
Error	2	0
Incomplete	1	0
Success	40/43	35/35

Table II
RESULTS OF GMCTS ON MASH APPLICATION. THE COLUMN ITERATION IS THE NUMBER OF SIMULATION OF GMCTS. AVGCR MEANS THE AVERAGE OF CUMULATIVE REWARD. THE OPTIMAL CUMULATIVE REWARD IS 15. THE LAST COLUMN SUCCESS IS THE PERCENTAGE OF EVALUATIONS WHERE THE GOAL "TOUCH THE BLUE FLAG AND THEN THE RED FLAG" HAS BEEN ACCOMPLISHED (THROUGH 100 EVALUATIONS).

Run	Iteration	AvgCR	Success
1	1	-13.65	
	10	4.82	
	100	9.67	
2	1	8.87	
	10	10.62	
	100	10.2	
3	1	12.09 ± 1.06	81%
	10	12.64 ± 1.03	86%
	100	11.38 ± 1.21	77%

Results of GMCTS are shown in Figures 7 and 8 in which the number of iterations is the number of simulations of the GMCTS Algorithm. The mean cumulative reward corresponds to the average length of longest words.

2) Results on Mash application: Results of the Clustering are : Over 12 runs, the number of runs with a bad correlation of features has been of 9. For the 3 other runs, 2 runs needed 5 iterations, one needed 8 iterations. At the end of the Clustering, 2 clusters have been well found.

Results of GMCTS are presented in the Table II.

3) Discussions:

Letter's draw optimization: The clustering works well. Although in $LD(100, 2000, 10)$ application, the size of the state space is 5 times larger and the maximal length of a simulation is 10 times bigger than in $LD(20, 100, 10)$, the clustering works always well and the GMCTS Algorithm doesn't need more simulations to converge in average. In a similar application, the french TV game "Des Chiffres et Des Lettres", the longest word has in average a length of 8.12 letters (Average calculated over 275 draws). In Figures 7 and 8, best runs have very good results; their score (given by the mean cumulative reward) reaches or exceeds a little 8.12. The score in average (around 7.5) is good, too.

Mash Application: The results are slow but stable. With 12 restarts on the clustering stage (2 hours each), we get 3 successes (which were detected as successes by heuristics without further testing); then, GMCTS could be successful on each of these 3 runs (8 hours each run). So on a parallel machine all this could be done in 10 hours, or 48 hours on a sequential machine. The success rate of CluVo is 25% (tested on 12 runs, and success can be detected on the fly) and the success rate of GMCTS is 100% (i.e. the goal "touch the blue flag then touch the red flag" is successfully found).

V. CONCLUSION

In this paper, we propose a generic algorithm CluVo+GMCTS which solves a very complex task in an unknown environment of large size. In this environment, the semantics of decisions are unknown, there is no prior knowledge and pure random search gives very rarely a good solution. Our approach combines Clustering and Monte-Carlo Tree Search. The Clustering method tries to learn some knowledge and from those results, GMCTS tries to find a strategy for reaching the goal. The genericity of the Algorithm has been shown on 2 applications. In further work, we will compare GMCTS with other algorithms such as Nested Monte-Carlo Search adapted to goal space.

VI. ACKNOWLEDGMENT:

The authors would like to thank the whole Mash team who has permitted us to make these experiments and who has contributed to develop the algorithm through many discussions and Olivier and Fabien Teytaud for the reading of this article. Experiments on the application of the draw of letters presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

REFERENCES

[1] T. Cazenave. Nested monte-carlo search. In C. Boutilier, editor, *IJCAI*, pages 456–461, 2009.

[2] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Proceedings of the 5th international conference on Computers and games*, CG'06, pages 72–83, 2007.

[3] D. Defays. An efficient algorithm for a complete link method. *Comput. J.*, 20(4):364–366, 1977.

[4] C. Dubout and F. Fleuret. Tasting families of features for image classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 929–936, 2011.

[5] G. Dulac-Arnold, L. Denoyer, P. Preux, and P. Gallinari. Fast reinforcement learning with large action sets using error-correcting output codes for mdp factorization. In *Machine Learning and Knowledge Discovery in Databases*, volume 7524, pages 180–194. 2012.

[6] L. Kocsis and C. Szepesvari. Bandit-based monte-carlo planning. In *ECML'06*, pages 282–293, 2006.

[7] A. Mcgovern, R. S. Sutton, and A. H. Fagg. Roles of macro-actions in accelerating reinforcement learning. In *In Grace Hopper Celebration of Women in Computing*, pages 13–18, 1997.

[8] S. M. Savaresi, D. L. Boley, S. Bittanti, and G. Gazzaniga. Cluster selection in divisive clustering algorithms. In *SIAM Internation Conference on Data Mining*, pages 299–314, 2002.