Towards a Learning Agent Architecture for Cross-Map Transfer

Cédric Herpson[1] and Vincent Corruble

Laboratoire d'Informatique de Paris 6, Université Pierre et Marie Curie,
104 avenue du Président Kennedy 75016 Paris, France
{cedric.herpson,vincent.corruble}@lip6.fr

Abstract. The capacity to apply knowledge in a context different than the one in which it was learned is still an open research question within the area of learning agents. This paper specifically addresses the issue of transfer of knowledge acquired through online learning in a partially observable environment characterized by its 2D geographical configuration. We propose an autonomous agent architecture combining an agent-centered representation and the supervised and unsupervised learning of discriminating concepts to achieve efficient cross-map transfer. Our preliminary experiments on a grid-world environment where two agents duel each other show that the agent's performances are improved through learning, including when it is tested on a map it has not yet seen.

Keywords: Generalization, Abstraction, Agent, Transfer Learning

1 Introduction

Learning and transfer of knowledge is a cross-discipline issue for those interested in understanding or simulating intelligent behavior. Knowledge transfer opens up the possibility of improving both learning speed and global performance of a system on a problem close to a known one. It has been addressed in particular by research in cognitive psychology and neuroscience [1]. In Artificial Intelligence, the ability to generalize has been an important focus of study, but mainly in the field of classification [2], i.e. for the identification of new instances of a given concept (e.g. spam recognition). Within the agent community, though a significant amount of research addresses the exploration of unknown environments, it mostly focuses on the exploration optimality, regardless of whether it is carried out in a mono- [18] or multi-agent setting [17]. Thus, with the notable exception of Case Base Reasoning [5], until recently, little effort had been put into the transfer of learned knowledge. Indeed, the need for an agent architecture which integrates learning and transfer capacities has become crucial in recent years with the development of new

[1] Corresponding author

application domains using, or open to the use of, autonomous agents, such as video games, military simulations or general public robotics.
Work in the area of strategy games is one concrete example; it has led to techniques which let agents learn strategies through playing [3,4]. Yet, learned strategies are only relevant to the game context in which they have been learned, that is to say a scenario, and more pointedly a specific "game map". Hence, each new scenario requires a new phase of learning since previous experience on previous game maps is not put to use. Recent work try to go beyond these limitations. However, they cannot be easily applied in open and/or complex environments as they require the full description of the state-action space.
The goal of this paper is to present a robust learning agent architecture able to realize efficient cross-map transfer in open or complex environments. To this aim, its main feature is to discover from an agent-centered point of view abstract concepts which allow the transfer of knowledge learned on a given topology to a different one, yet unknown. In the following we briefly introduce relevant literature. Next, we describe the proposed architecture and a preliminary evaluation thereof in a simplified game environment where the relevance of a learned concept is evaluated from the standpoint of the agent’s performances. Finally, we discuss experimental results and current limitations of our approach before concluding with some perspectives.

2 Related Work

An important part of recent work tackles the learning and transfer of knowledge problem by means of the Reinforcement Learning (RL) framework [6, 7, 8, 16]. To be able to generalize and to use RL in complex worlds, [6] assumes that states of the modeled Markovian Decision Process (MDP) are of different types (in a restricted number). This assumption simplifies the representation of the environment enough to allow the use of RL methods. However, the different types of states are determined *a priori*, which reduces the use of this technique to known and closed worlds. With a similar idea, i.e. to generalize across states, Dzeroski & Driessen [4] proposed the Relational Reinforcement Learning approach (RRL). This attractive combination of RL and Inductive Logic Programming is based upon the relational structure of the problem to abstract from the current goal. However, if RRL showed generalization abilities which are absent from traditional RL techniques, the use of first-order logic generates a sharp increase in the number of components usable for the complete description of each state. This increase led to the emergence of the same drawbacks as the tabular representation initially used in RL and does not allow its use in complex environments.
In the last few years, research on learning and transfer has found excellent test beds in the area of games. Indeed, games, and especially Real-Time Strategy (RTS) ones, provide research environments which are rich and complex, often stochastic, and favor experimental work [9]. In this context, a first approach of transfer models game states at a high level of abstraction meant to describe the state globally. Alternatively, another approach is to consider each unit or character of the game as an autonomous

agent interacting with the environment and processing only the information it can perceive locally.

Following this last approach, [10] tackles knowledge transfer within the SOAR cognitive architecture [11] and follows an agent-centered approach. The test bed is a first-person-shooter map where the agent's goal is to move from one point to another, and where some topological changes occur between the learning and test phases. It uses learning mechanisms based upon creation/modification[2] rules and on the storage of all spatial knowledge to reach this goal. However, storing all the topological information requires the use of huge amounts of memory which does not allow applying it on a larger scale.

With the same application domain, [16] addresses the problem at a high level of abstraction. It represents the structure of the problem as a relational MDP to tackle the planning problem on different maps in a RTS. However, the required full description of the state-action space does not allow the use of this approach in open or partially observable environments. More recently, [8] has proposed an architecture where an agent controls from a central point of view all the units from its side using a CBR learner defined as a combination of CBR and RL. This proposal was tested on a strategy game simulator, on a scenario where each side aims to destroy enemy forces. Their system builds a high-level description of the game state using only global information such as percentage of units "alive" on each side, percentage of territory controlled, etc. This radical abstraction makes the representation independent from any given map or game context. The architecture then selects an action solely as a function of this high-level description. Results obtained with this approach indicate its ability to reuse learned knowledge when initial positions and/or number of units vary. However, the fact that the game state description on which decisions are made is completely unrelated to the context (including its topology) constitutes a major obstacle to more ambitious transfer. Thus, a map of higher complexity, or a complete change of environment will not impact the state description and lead therefore to only one high level description for two distinct situations. As a result, only one action will be chosen where two different actions have to be selected.

In summary, the research described above make strong assumptions such as being totally independent of the context or knowing the structure of the problem in advance. In the next section, our proposal attempts to go beyond these limitations.

3 An Architecture for Cross-Map Transfer

3.1 Abstraction and Situated Representation

We consider a representation using the notion of a situated agent. It lets one change from a central, globalized point of view, often used in work on strategy games to an agent-centered perspective.

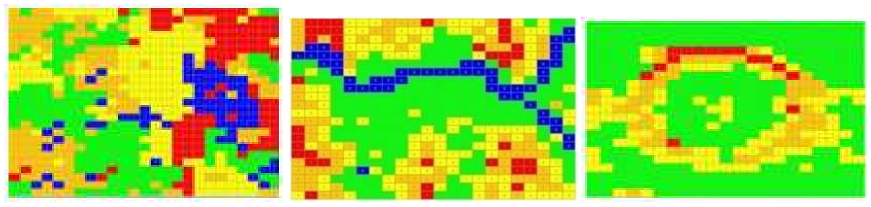
[2] In Soar, agent's knowledge is represented as rules.

Definition 1 (Situation). *A situation is the world view as perceived by the agent from its sensors at a given moment.*

A situation is the basic level of information; data obtained from sensors are expressed as a set of attribute-value couples. Our first assumption is that the elimination of information outside the sensors' field of view, considered as a "passive" abstraction [12] and usually seen as a constraint in domains such as situated robotics [13], brings significant advantages here. It offers the possibility for the agent's reasoning to be independent of map-specific (x,y) coordinates and of the environment's complexity, in terms of size as well as richness. However, by using agent-centered limited perceptions and representations, we lose some possibly relevant state information and enter the realm of partially observable environments. Thus, if the environment contains multiple agents, they become unobserved and unpredictable; which means that the environment is not stationary anymore. These properties are usually working hypotheses in most learning agent frameworks, as they are necessary to guarantee convergence with typical learning algorithms. Doing without them requires special attention.

3.2 The Duel Example

Before presenting in detail our architecture, we introduce an example that will be used both to illustrate our following explanations as well as to evaluate it in the experimental section. Consider a grid-world type of environment where each location on the grid is characterized by its altitude besides its (x,y) coordinates.



(a) (b) (c)

Fig. 1. Map (a) corresponds to the learning environment. Maps (b) and (c) respectively correspond to the Mountain and Cirque test environments. Altitude varies from blue (level 0, the lowest) to red (highest).

The effective field of view of an agent evolving on this map takes into account the obstacles present. As a consequence, the field of view of an agent located on a high position is better than the one of an agent located downhill.

The chosen scenario consists of a duel between two agents evolving within the above grid-world. Each agent's goal is to become the last survivor. They have available a range weapon and have to hit their competitor twice to win. The probability of hitting when shooting depends on the shooter-target distance as well as the angle of incidence of the shooting. A horizontal shoot has a very high hit probability, whereas when two agents are located at different altitudes, the one positioned higher will have a hit probability much higher than the lower one.

Final goal : alone(x) $\wedge$ alive(x)
State : ammo(x) $\in$ {true,false} and Status $\in$ {hurt,dead,ok}
Rules : ok(x) $\vee$ hurt(x) $\rightarrow$ alive(x) ; dead(x) $\rightarrow$ alone(y) [with x $\neq$ y] ;
hurt(x) $\wedge$ hit(x) $\rightarrow$ dead(x) ; ok(x) $\wedge$ hit(x) $\rightarrow$ hurt(x)
Actions : ammo(x) $\wedge$ see(x,y) $\rightarrow$ Shoot(x,y) ;
move and rotation towards the 4 cardinal directions.

Fig. 2. A simplified version of available rules for a duel

3.3 Architecture

Our architecture is based on a perception-action loop involving several components.

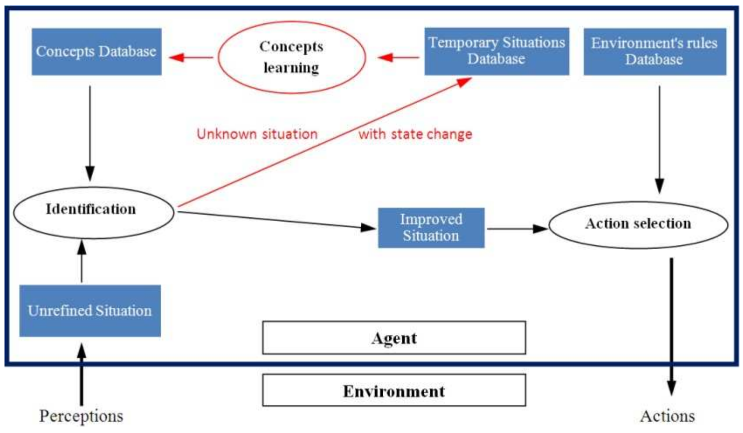


Fig. 3. General architecture of a learning agent for cross-map transfer

As shown in Figure 3, an agent has a memory to save facts, learned concepts and rules about the environment (*Concepts* and *Environment rules' Databases*).

Definition 2 (Concept) *Let* ***pre*** $\longrightarrow$ ***Action*** *be a rule.*
*A concept is a couple <**pre**, **descr**> where **descr** is the agent's representation of the precondition **pre**, learned from its world view (the situations).*

Thus, the learning mechanism (*Concept Learning*), using both supervised and unsupervised methods, continuously extracts patterns from perceived situations and associates them to the premise of the agent's actions/decision rules (red process in Fig 3). An inference engine (*Action Selection*) uses the relations between these different elements in order to select the actions that can lead the agent toward its current goal. The automatic recognition in the current situation of a prior learned pattern (*Identification*) helps reaching better decisions. Thus, learned concepts are used as an interface between the physical sphere and the representation sphere.

3.3.1 Learning Phase

As shown in the duel's rules example, Figure 2, the agent's internal state is defined by one or more variables. An agent's change of internal state corresponds to a change in one of its attributes. A change occurs as a consequence of the agent's last action or of another agent's one. Without prior knowledge, the learning or discovery of a new concept proceeds in three stages:

1. Save in the *Temporary Situation Base* (TSB) all situations newly encountered when a change in the agent's internal state occurs. Each saved situation is associated to the pre-condition part of the last action rule used.
2. Apply a learning algorithm to infer a pattern when the number of situations associated with the same precondition is above a threshold N.
3. Add the resulting couple < pre, pattern> into the *Concept Base (*CB).

In our duel example, if we consider that the weapon cartridge clip is not empty, then the *Shoot* action can only be used when an agent perceives its enemy. After a successful shot has killed its enemy, the shooter-agent's final goal is reached. This internal state change triggers the learning algorithm. Thus, the current situation is associated with the pre-condition *see(x,y)*. After N duels, the agent discovers that being located, for example, at a high point, improves the chances of seeing the enemy and therefore of winning.

In stage (2), two types of learning mechanism are used depending on whether the target concept is related to the final goal (case A) or to a sub-goal (case B).

Definition 3 (Sub-goal*).* *A sub-goal is an intermediary objective considered by the agent as a necessary condition to reach its final goal.*

The reason for this distinction is that we hypothesized that concepts linked to win-lose situations can be defined within a single representation. Assuming that it is true, winning and losing situations can be opposed and form two classes which can be discriminated more easily using classical learning algorithms. However, if the targeted concept is a sub-goal - e.g. to find a cartridge clip at the beginning of a duel - the only situations available to store will be the ones obtained when this sub-goal is reached.

A) Concept related to the final goal, learned under supervision

Environments in which agents evolve are, just as the real world, stochastic. There is therefore a significant amount of noise in the data from which we wish to extract a relevant situation's pattern. To tackle this, our algorithm (Algorithm 1 below) firstly removes some noise from data using a majority vote process (line 1 to 9). After this step, it then infers a pattern among remaining situations using a probabilistic classification tree[3] based on the C4.5 algorithm [14].

[3] The classification tree was selected after having tested a number of learning algorithms including Clustering, Boosting on MLP, KNN, Kmoy and SVM.

```
Let VS: VictorySituations, DS: DefeatSituations and pre: precondition
VS ← {vs ∈ TSB | vs ∈ <pre,descr>}
DS ← {ds ∈ TSB | ds ∈ <pre,descr>}
$\forall$ x ∈ VS $\bigcup$ DS
    $\forall$ y ∈ VS $\setminus\{x\}$
        If $(dist(x,y) < \eta)$ then x.nbPos ← x.nbPos+1
    $\forall$ y ∈ DS $\setminus\{x\}$
        If $(dist(x,y) < \theta)$ then x.nbNeg ← x.nbNeg+1
    If $((\frac{x.nbPos}{|VS|} < \frac{x.nbNeg}{|DS|}) \wedge x \in DS)$ then nDS←nDS $\bigcup$ {x}
    If $((\frac{x.nbPos}{|VS|} > \frac{x.nbNeg}{|DS|}) \wedge x \in VS)$ then nVS←nVS $\bigcup$ {x}
pattern ← C4.5(nVS,nDS)
Concept Base ← Concept Base $\bigcup$ <pre,pattern>
TSB ← TSB $\setminus$(VS $\bigcup$ DS)
```

Algorithm 1. Learning a concept related to the final goal

Once these two stages have been realized, the new association between a pattern and a precondition is added to the *Concept Base* (CB). Finally, the situations associated to the precondition of the action having triggered the learning phase are removed from the *Temporary Situation Base* (TSB).

B) Concept related to a sub-goal : process of characterization

Characterizing the concept consists of identifying attributes whose values show a pattern within the subset of situations associated with one pre-condition. Let us consider that attribute L is representative of a given concept whenever more than α of its instances lie within the interval defined by a variation of γ around its mean value. The resulting situation constitutes a prototype for the concept which originated the N situations under consideration. In a manner similar to what was presented in Case A above, situations used from the TSB are deleted, and the new concept is added to the CB.

In order to distinguish the various processes at work in our architecture, we have so far considered that the CB was initially empty during the recording of <pre, new situation> couples in the TSB. However, our architecture needs to be able to use a concept already learned. As a consequence, when the CB is not empty during the recording of new situations in the TSB, a similarity measure is applied between new incoming situations and the descriptions of learned concepts. If a correspondence is detected (similarity above a threshold), the couple <pre, pattern> in the CB is updated with an additional precondition and the incoming situation is deleted. The same similarity measure will be used in the decision phase presented in the following.

3.3.2 Decision Phase

The inference engine, prolog, is interfaced with the agent’s learning and action mechanisms (themselves implemented in Java). At each time step, the situation perceived by the agent updates the set of facts available to the inference engine. Prolog then uses its model of the environment and known facts so as to ‘prove’ the desired goal and to select the best action.

```
 1 : While (!goalReached)
 2 :     currentSit ← agent.updateCurrentSit();
 3 :     agent.updateFactToProlog(currentSit);
 4 :     action ← agent.prolog.prove();
 5 :     If (action!=null) then
 6 :          agent.doAction(action);
 7 :     else
 8 :         descr ← agent.CB.search(agent.prolog.getMissingPre());
 9 :         If (descr!=null) then
10 :             location ← agent.searchInFov(descr);
11 :             x ← randomInt();
12 :             If ($x > \epsilon \wedge$ location!=null) then
13 :             agent.doMove(location);
14 :         else
15 :              agent.randomMove();
16 : EndWhile
```

Algorithm 2. Decision-Action loop

During the identification stage, at line 10, the agent considers virtually all the positions it can perceive in its field of view and evaluates their quality relative to the targeted concept. The identification carried out at each location works in two different manners depending on whether the description obtained from the Concept Base (line 8) is an identification function (Case A: concept related to the end goal) or a prototype (Case B: concept related to a sub-goal).

A) Identification of a concept related to the end goal.

Using the disambiguation process previously introduced in Algorithm 1, virtual situations detected within the horizon of the agent are filtered. Then, the classification tree computes for each situation its relevance level to the concept the agent is looking for (where 1 means certainty and 0 the opposite). To limit the risk of misclassification, only a virtual situation with a confidence value above a threshold β (β in [0, 1]) can be selected.

B) Identification of a concept related to a sub-goal

In this case, the element extracted from the Concept Base is a situation prototypical of the target concept. To estimate the similarity over attributes describing the various situations, a non-informed similarity measure based on the Euclidian distances is applied between the prototypical situations and perceived ones[4]. As for a concept related to the end goal, only a virtual situation with a similarity value above the threshold β is selected.

4 Experimental Evaluation

In this section, we describe an empirical evaluation of the learning and transfer capacity of our architecture. As introduced in 3.2, we chose to test our architecture in a grid-word type environment where two agents duel each other. To this aim, we set

[4] The Euclidian distances were selected after initial experiments with Manhattan, Euclidian and Mahalanobis distances.

up three maps of varying topologies, sizes and maximum altitudes. Each action's availability depends on the agent's current situation. The shoot action is automatically executed when an agent perceives its enemy in its field of view. This was decided so as to reduce the simulation time. The result of the shoot action (target hit or not) is stochastic and depends on a number of characteristics unknown to the agent.

4.1 Experimental Setup

To test the impact of learning a single concept on the performances, changes of internal states leading to the storing of situations and the learning of a concept are only to be related to end-game situations. We therefore consider that ammunitions are unlimited. N, the threshold that triggers the learning algorithm is set to 20. The β parameter is set to 0,8. Parameters α and γ are set respectively to 0,75 (overqualified majority) and 0,1. η, θ and ε, the different parameters in algorithms 1 and 2, are set to 0,1. Performances are evaluated based on three criteria : the percentage of victories obtained by each agent, the average number of time steps needed to end an episode or game (i.e. when a agent wins) and the evolution of the proportion of the environment explored by each agent over time.

The first experiment aims to evaluate the ability of an agent to transfer knowledge to a new environment. To obtain a baseline, 1000 duels between two random agents are run for the three considered environments. Next, one learning agent runs a series of episodes against a random one on the learning environment until it learns one concept related to the final-goal. Then, the learning step is deactivated and we run a series of 1000 episodes opposing a random agent and the trained agent on the three environments.

The second experiment aims to evaluate the capacity of our architecture to produce a behavior of a higher efficiency by adding a memory component. Indeed, in the first experiments, stimuli from the agent's sensors do not provide any temporal information. An agent is not able to know if it is currently in an area it just left a few steps ago. Consequently, it can move for a long time in an area where there is no enemy. If information from the new sensor, which is added to the description of each situation, is recognized as relevant during the concept learning, it may improve the results because of a better spatial exploration. Following the protocol previously introduced for the first experiments, an agent thus upgraded is therefore evaluated against a random one.

In the presentation of all results, we refer to the three types of agents as follows:

- *Random*: baseline agent, with no knowledge or learning mechanism.
- *Intel_1*: agent having discovered/learned a concept related to the end goal.
- *Intel_2*: agent with a short-term memory having learned a concept related to the end goal.

4.2 Results

The tree learned in the first experiment (fig. 4 below) shows that the agent will favor situations with a good field of view, or, if the field of view is considered average, situations where average altitude around the agent location is lower than its own. Thus the learned concept drives the agent to search and follow the map's ridge paths.

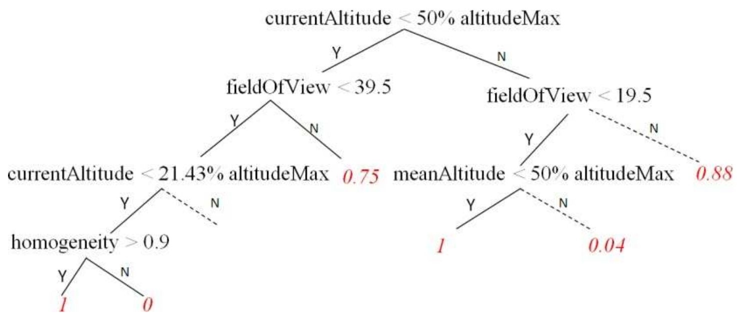

currentAltitude < 50% altitudeMax
Y
N
fieldOfView < 39.5
fieldOfView < 19.5
Y
N
Y
N
currentAltitude < 21.43% altitudeMax
0.75
meanAltitude < 50% altitudeMax
0.88
Y
N
Y
N
homogeneity > 0.9
1
0.04
Y
N
1
0

Fig. 4. Tree automatically generated and associated to the *see(x,y)* precondition. A value above 0.75 indicates a favorable situation. Dotted lines indicate sub-trees that were manually pruned from the figure for better clarity.

The results obtained for the two first evaluation criteria on the three environments are presented in fig 5. They show that the concept learned is sufficiently relevant to improve performance on the training environment. Furthermore, results obtained on the two test environments are even better than those from the training environment.

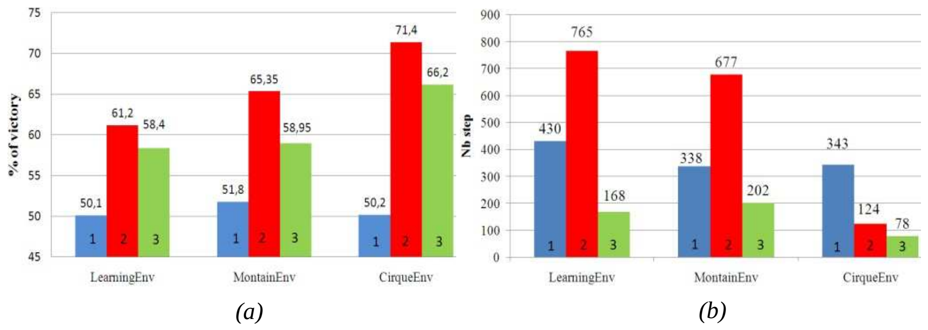


Fig. 5. *(a)* represents the percentage of wins for agents *random* (1), *Intel_1*(2) and *Intel_2* (3) respectively, all against a random agent - *(b)* shows the average duration of a duel for the three types of agents facing a random agent. In each case, results are given in the three different environments after 1000 duels.

Figure 5.a shows that *Intel_1* improves results over a random agent by 22% in the training environment and by 26 to 42% on test environments. *Intel_2* improves results by 15 to 32% depending on the environment. However, as it performs worse than *Intel_1*, it looks as if its memory is more of a hindrance than an advantage here. Results obtained in fig 5.b, with the second evaluation criterion (the length of a duel) modify this last result: *Intel_1* increases the average duel time for two of the three environments while *Intel_2* drastically reduces the time on all three of them. Results of *Intel_2* indicate that the temporal information is recognized as relevant during the concept learning and is profitable, considering the duration of a duel at least. For the third evaluation criterion, the general shape of graphs is similar for all 3 environments so we give below in figure 6 the results for the Mountain environment only.

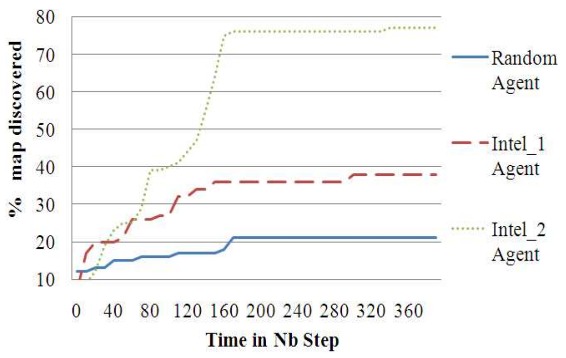


Fig. 6. Evolution over time of the proportion of the environment explored, for each agent in the Mountain environment.

Because of its memory the *Intel_2* agent goes over a larger part of the environment in a given time interval. It is therefore more likely to encounter its opponent than *Intel_1*. The latter, without short-term memory, repeatedly crosses the same zone. In this context, *Intel_1* has an advantage: as actions are simultaneous, the one moving can very well enter the field of view of its opponent and get shot immediately. Agent *Intel_1* takes "fewer risks", and therefore plays longer and wins more often.

5 Discussion

Though our results provide evidence regarding the validity of the principles behind our learning architecture for the transfer of knowledge, a lot of work remains. First of all, the unsupervised learning of concepts related to sub-goals must be evaluated. For this purpose, experiments where agents begin each duel without ammunition are in progress. Next, the topology plays a large part in determining the ability of the agents. Thus, a thorough study in more various and more complex environments has to be done to prove our architecture's robustness when scaling up. Besides these points, our architecture suffers in its current form from several limitations. Thus, our architecture lets our agents identify desirable situations within their environment by making use of the game's rules.They are,however, not able to reason on what constitutes undesirable situations for them. Using a model of the environment to model the goals of the opponent would be relevant to avoid situations which are desirable to the enemy.

Irrespective of these points, it is interesting to note that our architecture does not seem to suffer from the agent-centric drawbacks highlighted in [15]. Indeed, results show that the impact of switching from a complete world representation to a partially observable one makes the learning performances fall drastically to the point of failing to reach the targeted goal. One possible explanation is that their work relies on RL algorithms. In this framework, properties of perceived objects are independent of their functional utility. Conversely, in the line of research we follow here, the essential role of perception is to give access to information organizing the action. Thus, perceived properties of objects must be dependent on the actions capacity of the agent. Our architecture relies on this principle and lets an agent learn to choose a state with the goal of executing an action resulting from a reasoning process.

6 Conclusion and Perspectives

We have proposed a new learning agent architecture for cross-map transfer. Initial results are encouraging. They show that our architecture does lead to a relevant knowledge transfer in stochastic environments regarding topological information, which proves efficient when the environment changes. Besides some necessary improvements discussed in section 5, one short-term perspective is to evaluate the efficiency of our learning architecture facing the Reinforcement Learning approach. This experiment will allow us to compare both the adaptability and efficiency of our approach to the RL-based ones in the transfer context. With a long-term view, an important perspective for this work is to extend the architecture to address the multi-agent framework. The choice of knowledge representation proposed here is compatible with the use of a command hierarchy where low-level units with local information can interact and share their knowledge at the strategic level. We aim to tackle the issue of communication and coordination between agents at this level.

References

1. S. Thrun, L.P.: Learning to learn. Norwell edn. Kluwer Academic (1998)
2. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer (2006)
3. Madeira, C.: Adaptive Agents for Modern Strategy Games: an Approach Based on Reinforcement Learning. PhD thesis, University of Paris VI (2007)
4. M. Ponsen, H. Muoz-Avila, P.S., Aha, D.: Automatically generating game tactics through evolutionary learning. AI Magazine (2006)
5. Watson, I., Marir, F.: Case-based reasoning: a review. In: KE. (1994) 327-354
6. Bethany R. Leffler, M.L.L., Edmunds Littman, T.: Efficient reinforcement learning with relocatable action models. In: Proc of AAAI. (2007)
7. K. Driessens, R. Givan, P.T.: Rrl : An overview. In: Proc of ICML. (2004)
8. M.Sharma, M.Holmes, Santamaria, J.C., A.Irani, Lee, C., Ram, I.J.A.: Transfer learning in real-time strategy games using hybrid cbr/rl. In: IJCAI. (2007)
9. D. Aha, M.M., Ponsen, M.: Learning to win: Case-based plan selection in a real-time strategy game. In: Proc of ICCBR. (2005)
10. Gorski, N., Laird., J.: Experiments in transfer across multiple learning mechanisms. In: Proc of ICML. (2006)
11. Lehman, J. Laird, R.: A gentle introduction to soar. (2006)
12. Agre, P.: The dynamic structure of everyday life. PhD thesis, MIT (1988)
13. N. Bredeche, S.Z., Zucker, J.: Perceptual learning and abstraction in machine learning. Systems, Man and Cybernetics, Part C, IEEE (2006) 172-181
14. Breiman, L.: Classification and Regression Trees. Boca Raton, London (1993)
15. S. Finney, P. Wakker, L.K., Oates, T.: The thing that we tried didn't work very well : Deictic representation in reinforcement learning. In: Proc of UAI, Morgan Kaufmann (2002)
16. C. Guestrin, D. Koller, C. Gearhart, N. Kanodia : Generalizing Plans to New Environments in Relational MDPs In: IJCAI.(2003).
17. Burgard, W., Moors, M., Stachniss, C., Schneider, F.: Coordinated multi-robot exploration. IEEE Transactions on Robotics 21 (2005) 376-386
18. Albers, S., Kursawe, K., Schuierer, S.: Exploring unknown environments with obstacles. In Proc. 10th ACM-SIAM Sympos. Discrete Algorithms. (1998) 842-843