Plan-based Reward Shaping for Reinforcement Learning

Marek Grzes and Daniel Kudenko

***Abstract*—Reinforcement learning, while being a highly popular learning technique for agents and multi-agent systems, has so far encountered difficulties when applying it to more complex domains due to scaling-up problems. This paper focuses on the use of domain knowledge to improve the convergence speed and optimality of various RL techniques. Specifically, we propose the use of high-level STRIPS operator knowledge in reward shaping to focus the search for the optimal policy. Empirical results show that the plan-based reward shaping approach outperforms other RL techniques, including alternative manual and MDP-based reward shaping when it is used in its basic form. We show that MDP-based reward shaping may fail and successful experiments with STRIPS-based shaping suggest modifications which can overcome encountered problems. The STRIPS-based method we propose allows expressing the same domain knowledge in a different way and the domain expert can choose whether to define an MDP or STRIPS planning task. We also evaluate the robustness of the proposed STRIPS-based technique to errors in the plan knowledge.**

***Index Terms*—Reinforcement learning, reward shaping, symbolic planning, STRIPS**

I. Introduction

Reinforcement learning (RL) is a popular method to design autonomous agents that learn from interactions with the environment. In contrast to supervised learning, RL methods do not rely on instructive feedback, i.e., the agent is not informed what the best action in a given situation is. Instead, the agent is guided by the numerical reward which defines the optimal behaviour for solving the task. The problem with this kind of numeric guidance in goal-based tasks is that the reward from the environment is given only upon reaching the goal state. Non-goal states are not rewarded which leads to two kinds of problems:

1) The *temporal credit assignment problem*, i.e., the problem of determining which part of the behaviour deserves the reward.
2) Slower convergence: conventional RL algorithms employ a delayed approach propagating the final goal reward in a discounted way or assigning a cost to non-goal states. However the back-propagation of the goal reward over the state space is time consuming.

To speed up the learning process, and to tackle the temporal credit assignment problem, the concept of *shaping reward* has been considered in the field [1], [2]. The idea of reward shaping is to give an additional (numerical) feedback to the agent in some intermediate states that helps to guide it towards the goal state in a more controlled fashion.

Even though reward shaping has been powerful in many experiments it quickly turned out that, used improperly, it can be also misleading [2]. To deal with such problems Ng et al. [1] proposed potential-based reward shaping $F(s, s')$ as the difference of some potential function Φ defined over a source s and a destination state s':

$$F(s, s') = \gamma\Phi(s') - \Phi(s). \quad (1)$$

They proved that reward shaping defined in this way is necessary and sufficient to learn a policy which is equivalent to the one learned without reward shaping.

One problem with reward shaping is that often detailed knowledge of the potential of states is not available, or very difficult to represent directly in the form of a shaped reward. Rather, some high level knowledge of the problem domain exists, that does not lend itself easily to explicit reward shaping.

In this paper we focus on the use of high-level STRIPS operators to automatically create a potential-based reward function, that improves the ability and speed of the agent to converge towards the optimal policy. The only interface between the basic RL algorithm and the planner is the shaping reward and information about the current state. In related works where planning operators were also used [3], [4] a RL agent learns an explicit policy for these operators. In our approach symbolic planning provides additional knowledge to a classical RL agent in a principled way through reward shaping. As a result, our approach does not require frequent re-planning as is for example the case in [3].

We evaluate the proposed method in a flag-collection domain, where there is a goal state (necessary for applying STRIPS) and a number of locally optimal ways to reach the goal. Specifically, we demonstrate the success of our method by comparing it to RL without any reward shaping, RL with manual reward shaping, and an alternative technique for automatic reward shaping based on abstract MDPs [5] when it is used in its basic form. Thus, the contribution of the paper is the following: 1) we propose and evaluate a novel method to use the STRIPS-based planning as an alternative to MDP-based planning for reward shaping; 2) we show that MDP-based reward shaping may fail and successful experiments with STRIPS-based shaping suggest modifications which can overcome encountered problems. The STRIPS-based method

This research was sponsored by the United Kingdom Ministry of Defence Research Programme.

M. Grzes and D. Kudenko are with the Department of Computer Science, University of York, Heslington, YO10 5DD York, United Kingdom, (e-mail: {grzes, kudenko}@cs.york.ac.uk).

we propose allows expressing the same domain knowledge in a different way and the domain expert can choose whether to define an MDP or STRIPS planning task. The STRIPS-based approach brings new merits to reward shaping from abstract/high level planning in domains with the intensional representation [6] which allows for symbolic reasoning.

High-level domain knowledge is often of a heuristic nature and may contain errors. We address this issue by evaluating the robustness of plan-based reward shaping when faced with incorrect high-level state definitions or plans.

The remainder of this paper is organised as follows. Section 2 introduces reinforcement learning. The proposed method to define the potential function for reward shaping is introduced in section 3. The experimental domain is described in section 4 and the chosen RL algorithms are presented in section 5. Section 6 shows how the proposed method can be used in the experimental domain, and a range of empirical experiments and results are presented in section 7. Section 8 concludes the paper with plans for further research.

II. Markov Decision Processes and Reinforcement Learning

A Markov Decision Process (MDP) is a tuple (S, A, T, R), where S is the state space, A is the action space, $T(s, a, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t + 1$, $R(s, a, s')$ is the immediate reward received when action a taken in state s results in a transition to state s'. The problem of solving an MDP is to find a policy (i.e., mapping from states to actions) which maximises the accumulated reward. When the environment dynamics (transition probabilities and a reward function) are available, this task becomes a planning problem which can be solved using iterative approaches like policy and value iteration [7]. Value iteration which is used in this work applies the following update rule:

$$V_{k+1}(s) = \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V_k(s')]. \quad (2)$$

The value of state s is updated according to the best action after one sweep of policy evaluation.

MDPs represent a modelling framework for RL agents whose goal is to learn an optimal policy when the environment dynamics are not available. Thus value iteration in the form presented in Equation 2 can not be used. However the concept of an iterative approach in itself is the backbone of the majority of RL algorithms. These algorithms apply so called temporal-difference updates to propagate information about values of states $V(s)$ or state-action $Q(s, a)$ pairs. These updates are based on the difference of the two temporally different estimates of a particular state or state-action value. Model-free SARSA is such a method [7]. It updates state-action values by the formula:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]. \quad (3)$$

It modifies the value of taking action a in state s, when after executing this action the environment returned reward r, moved to a new state s', and action a' was chosen in state s'. Model-based RL algorithms (e.g., DynaQ) learn additionally how the world responds to its actions (transition probabilities) and what reward is given (reward function) and use this model for simulated backups made in addition to real experience.

Immediate reward r which is in the update rule given by Equation 3 represents the feedback from the environment. The idea of reward shaping is to provide an additional reward which will improve the performance of the agent. This concept can be represented by the following formula for the SARSA algorithm:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + F(s, a, s') + \gamma Q(s', a') - Q(s, a)],$$

where $F(s, a, s')$ is the general form of the shaping reward which in our analysis is a function $F : S \times S \rightarrow \mathbb{R}$, with $F(s, s')$. The main focus of this paper is how to compute this value in the particular case when it is defined as the difference of potentials of consecutive states s and s' (see Equation 1). This reduces to the problem of how to compute the potential $\Phi(s)$.

III. Plan-based Reward Shaping

The class of RL problems is investigated in which background knowledge allows defining state and temporal abstractions using intensional representation [6]. Abstract states are defined in terms of propositions and first order predicates, and temporally extended actions (or *options* [8]) can be treated as primitive actions at the abstract level. The function $f_{abs}(S) = Z$ maps states $s \in S$ onto their corresponding abstract states $z \in Z$.

A. Potential Based on STRIPS Plan

The intensional representation allows for symbolic reasoning at an abstract level when options can be defined in terms of changes to the symbolic representation of the state space, e.g., they can be expressed as STRIPS operators. For such problems STRIPS planning can be used to reason at this abstract level. When the RL problem is to learn a policy which moves the agent from start state s_0 to goal state s_g it can be translated to the high level problem of moving from state $z_0 = f_{abs}(s_0)$ to state $z_g = f_{abs}(s_g)$. Because of the intensional representation at the abstract level, symbolic reasoning can be used to solve the planning problem of moving form state z_0 to goal state z_g. It is a classical planning task which can be solved using standard STRIPS planners (Graphplan [9] is used in our experiments). The trajectory $\omega = (z_0, z_1, ..., z_g)$ of abstract states (obtained from plan execution at the abstract level) can be used to define the potential for low level states as:

$$\Phi(s) = step(f_{abs}(s)),$$

where the function $step(z)$ returns the time step at which given abstract state z appears during plan execution. In other words, the potential is incremented after the RL agent has successfully completed an abstract action in the plan, and reached a (low-level) state that is subsumed by the corresponding abstract state in the trajectory.

The question remains what potential to assign to those abstract states that do not occur in the plan. One option is to ignore such states and assign a default value of zero. This approach can strongly bias the agent to follow the given path. The agent would be discouraged from moving away from the plan. As it will be discussed later, this leads to problems when the plan is wrong and in particular when there is no transition from state z_i to state z_{i+1} in the environment. The agent may not be able to get out of state z_i, because of the negative reward for going to any state other than z_{i+1}.

We propose a more flexible approach that will allow the agent to abandon the plan and look for a better solution when the plan is wrong. Figure 1 shows the algorithm. States which are in trajectory ω (plan states) have their potential set to the time step of their occurrence in the plan. Non-plan states that are reachable from any state $z \in \omega$ have their potential set to the potential of the last visited plan state (variable $last$). In this way the agent is not discouraged from diverging from the plan (it is also not rewarded for doing so).

A problem with this approach is that some non-plan states can be reached from different levels of potential. For this reason, for each non-plan state the highest value of the last potential is stored in the array Max. The main aim of using this array is to prevent continuous changes in the potential of non-plan states which may be disadvantageous for the convergence of the value function.

initialise $last \leftarrow 0$

if $f_{abs}(s) \in \omega$ **then**
  $last = step(f_{abs}(s))$
  return $step(f_{abs}(s))$
else
  if $last > Max(f_{abs}(s))$ **then**
    $Max(f_{abs}(s)) = last$
    return $last$
  else
    $last = Max(f_{abs}(s))$
    return $Max(f_{abs}(s))$
  end if
end if

Fig. 1. Assigning potential $\Phi(s)$ to low level states through corresponding abstract states.

The abstract goal state in the considered class of RL tasks needs to be defined as a conjunction of propositions. The most straightforward way to define potential for such goals manually is to raise it with each goal proposition which appears in a given state. This kind of potential, even though it gives some hints to the agent which propositions bring it closer to the goal, does not take into account how the environment is regulated (there may be a certain sequence of achieving goal conditions, that leads to higher rewards). One example is the travelling salesmen problem. Potential raised just for each visited town will strongly bias the nearest neighbour strategy. An admissible heuristic based on, e.g., minimum spanning trees can be used to give correct (optimistic) potential [10]. In our approach instead of encouraging the agent to obtain just goal propositions, a more informed solution is proposed that takes into account how the environment behaves.

B. Potential Based on Abstract MDP

Marthi [5] proposed a general framework to learn the potential function by solving an abstract MDP. In this section we show how this idea can be applied with the same kind of knowledge that is given to the STRIPS-based approach. The automatic shaping algorithm obtains potential by firstly learning dynamics for options (i.e., actions at the abstract level) and secondly solving an abstract MDP. Options can be defined as policies over low level actions. Because in our class of problems options are assumed to be primitive and deterministic actions at an abstract level, computation of their dynamics can be omitted. An abstract MDP (e.g., value iteration from Equation 2 can be applied) can be solved before target RL learning and the obtained value function is used directly as the potential. The following equation describes this fact:

$$\Phi(s) = \hat{V}(f_{abs}(s)),$$

where $\hat{V}(z)$ is the value function over state space Z and it represents an optimal solution to the corresponding MDP-based planning problem. Because the high level model is deterministic and options make transitions between abstract states, this planning task can be solved using the following formula:

$$V_{k+1}(z) = \max_{z'}[R_{zz'} + \gamma V_k(z')]. \quad (4)$$

Knowledge equivalent to STRIPS operators can be used to determine the next possible states z' for given state z. The reward given upon entering the abstract goal state and discount factor γ can be chosen to make the difference in the value function between neighbouring states equal to one, thus enabling us to perform easier comparisons with the STRIPS-based reward shaping approach.

IV. Experimental domain

The proposed algorithms are evaluated on an extended version of the navigation maze problem. This problem has been used in many RL investigations, and is representative of RL problems with the following properties:

- There exists an abstract goal state. This can stand for a number of actual states. A well-defined goal state is necessary for applying STRIPS planning.
- There are many ways to reach the goal, with varying associated rewards. In other words, there are local policy optima that the RL agent can get stuck in.

We use the artificial domain to evaluate our algorithm, therefore, it will be suitable for any real-world problem with these properties (approach adopted also in [4]).

In the basic navigation maze problem an agent moves in a maze and has to learn how to navigate to a given goal position. In the extended version of this problem domain, the agent additionally has to collect flags (i.e., visit certain points in the maze) and bring them to the goal position. The reward at the goal is proportional to the number of flags collected. In order to introduce abstraction and demonstrate the use of high-level planning, the maze is additionally partitioned into areas (rooms).

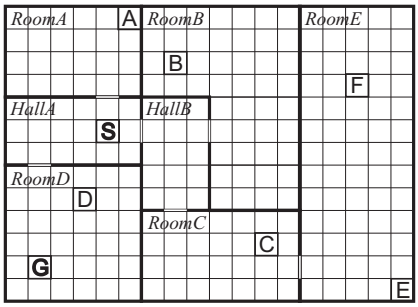

Fig. 2. The map of the maze problem. S is the start position and G the goal position. Capital letters represent flags which need to be collected.

Because an episode ends when the agent reaches the goal position regardless of the number of collected flags, this problem has been used in the past to evaluate sophisticated exploration strategies (e.g., [11], [12]). The learning agent can easily get stuck in a local optimum, bringing only a reduced number of flags to the goal position.

An example maze is shown in Figure 2. The agent starts in state S and has to reach goal position G after collecting as many flags (labelled A, B, C, D, E, F) as possible. The episode ends when the goal position has been reached and the reward proportional to the number of collected flags is given. Thus the reward is zero in all states except the goal state. The agent can choose from eight movement actions which deterministically lead to one of eight adjacent cells when there are no walls. The move action has no effect when the target cell is separated by a wall.

V. Evaluated algorithms and parameters

To conduct the evaluation two RL algorithms are used: SARSA and DynaQ. The usage of SARSA aims at investigating the influence of potential-based reward shaping on model-free reinforcement learning. Model-based methods are represented by DynaQ. All these RL algorithms were used in its basic form as they are presented in [7]. The following common values for parameters were used: $\alpha = 0.1$, $\gamma = 0.99$, the number of episodes per experiment 10^5. In all experiments an ϵ-greedy exploration strategy was used where epsilon was decreased linearly from 0.3 in the first episode to 0.01 in the last episode.

Reward shaping was applied to all the above RL algorithms. Plan-based reward shaping was compared with a non-shaping approach and with three other shaping solutions. This results in five reward shaping options: 1) no reward shaping, 2) STRIPS-based reward shaping, 3) abstract MDP-based reward shaping, 4) flag-based reward shaping, 5) composed reward shaping. STRIPS-based and abstract MDP-based reward shaping appear in the form as they were introduced. In the above no-shaping case no shaping reward is given. The flag-based shaping reward is determined by the number of collected flags, and the potential is the function $\Phi(s) = flags(s)$, where $flags(s)$ is the number of collected flags in state s. It is an instance of the manual shaping approach (discussed in Section III-A) which raises the potential for each goal proposition achieved in the current state. This kind of reward shaping thus represents the "nearest flag" heuristic. In composed reward shaping the potential is a sum $\Phi(s) = plan(s) + flags(s)$ of STRIPS-based potential ($plan(s)$) and the number of collected flags in state s ($flags(s)$). Flag-based reward shaping when combined in this way with STRIPS-based shaping may hurt the performance of pure STRIPS-based approach. However the "nearest flag" bias added by flag-based information can help in the case of incorrect planning knowledge. For this reason such composition of flag- and STRIPS-based shaping named composed is also evaluated.

If not explicitly mentioned otherwise, all experiments were repeated ten times and the average performance is shown in the result graphs.

VI. Potential for Experimental Domain

This section shows how the proposed RL and reward shaping approaches were applied to the flag collection domain.

A. Low Level Model

In our experiments, reinforcement learning is carried out at the low level which is defined by the target MDP (S,A,R,T), where S is the state space defined by the position of the agent in the 13x18 maze and by the collected flags, and A is the set of eight primitive actions corresponding to eight movement directions. The reward function R and transition probabilities T are not known to the agent in advance.

B. High Level Knowledge

Plan-based reward shaping assumes that there exists a high level structure in the modelled world. The access to two types of knowledge is required:

1) **State mapping** The mapping from low level to abstract states. The function which maps low level states into abstract states identifies each abstract state as the area in the maze in which the given low level position is located. Hence, the abstract state is determined by the room location of the agent and the collection of collected flags. Such a state can be symbolically expressed as:

$$robot\_in(roomB) \wedge taken(flagE) \wedge taken(flagF).$$

2) **Transitions** Possible transitions between high level states. In this case there are two types of knowledge which allow defining transitions at the abstract level:
   a) Possible transitions between areas in the maze (i.e., which adjacent rooms have doors between them).
   b) Location of flags: in which room a given flag is located.

C. High Level Planning Problems

Knowledge about the high level structure of the world is used to define high level planning problems. The abstract state representation attributes are used to define state representation for both classical and MDP-based planners. In the case of the

STRIPS representation the location of the robot and symbolic names of collected flags are used for an intensional description of the world. For the MDP-based planner the state space is enumerated and all possible states are collected in the tabular representation which has 448 entries. In both cases the state encoding preserves the Markov property.

Both investigated planning approaches require action models. In this case knowledge about transitions and intensional state representation is used to define high level actions. The following STRIPS operators were used:

```
(TAKE ((<flag> FLAG) (<area> AREA))
    (preconds (flag-in <flag> <area>) (robot-in <area>))
    (effects  (del flag-in <flag> <area>) (taken <flag>)))
(MOVE ((<from> AREA) (<to> AREA))
    (preconds (robot-in <from>) (next-to <from> <to>))
    (effects  (del robot-in <from>) (robot-in <to>)))
```

These operators together with the knowledge about the possible transitions between areas and the location of flags allow reasoning about the changes in the environment. The same knowledge is used to define possible transitions between abstract states in the abstract MDP, strictly to find for each state the set of reachable states. According to the description of the algorithm, deterministic options are assumed which allow for deterministic transitions between abstract states.

Introduced STRIPS actions allow reasoning symbolically about the changes in the world. The planner has to find a sequence of MOVE and TAKE actions which can transform the system from the start state in which $robot\_in(hallA)$ to the goal state:

$$robot\_in(roomD) \wedge taken(flagA) \wedge ... \wedge taken(flagF).$$

Because of the closed-world assumption (everything not mentioned explicitly in the description of the state is assumed to be false) the start state has to define all initial facts, like locations of flags (e.g., $flag - in(flagA, roomA)$ and connections between rooms (e.g., $next - to(hallB, roomC)$). The last group of facts is called rigid facts because they do not change over time (the fact whether rooms are connected or not remains unchanged).

Both the MDP and STRIPS planning problems can be solved in advance before the learning takes place. Once these problems have been solved they can be used to assign potential to high level states directly and to low level RL states indirectly via the mapping function, which translates low level states to high level abstract states. The potential is assigned to abstract states in the manner presented earlier.

VII. Empirical Results

In this section the empirical results are presented and discussed.

Even though the high level plans used for reward shaping are optimal according to the provided high level knowledge, this knowledge may contain errors. Therefore, the plan may not be optimal at the low level where the RL agent operates. For this reason the presentation of experimental results is divided into two sections. First, the results on different RL algorithms are analysed when the high level plan is optimal. Afterwards, various possible plan deficiencies are defined and their impact is empirically evaluated.

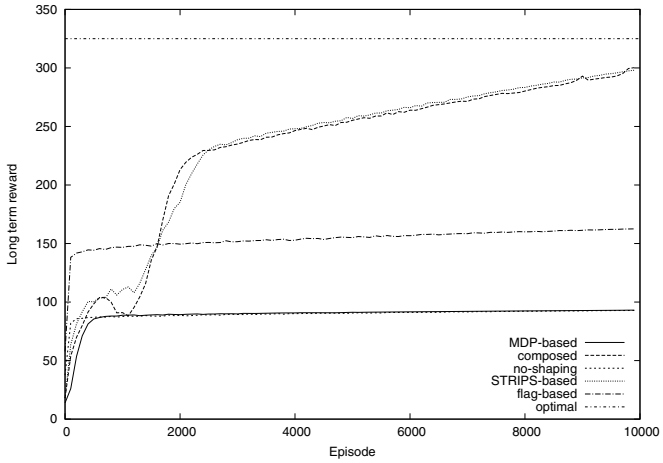

Fig. 3. SARSA results with all reward configurations.

A. Results with Optimal Plan

Results presented in this section are for the test domain as shown in Figure 2. High level plans generated by STRIPS planning and the abstract MDP are both optimal at the lower RL level. The STRIPS plan is shown in Figure 5. The MDP-based plan leads to the same sequence of visited abstract states as in the STRIPS plan when the policy determined by the value function is followed from the start to the goal state.

The discussion of experimental results is done separately for model-free and model-based RL algorithms.

1) Model-Free Methods: The first set of experiments looks at the performance of the different reward shaping approaches when used with model-free RL. In Figure 3 results with SARSA are presented. They show the difficulty of the investigated maze problem in terms of exploration. In all 10 runs the no-shaping RL version was not able to learn to collect more than one flag. It quickly converged to a sub-optimal solution which takes only flag D and directly moves to position G (the goal position). The only approaches that were able to learn to collect all flags (though not in all runs) are using STRIPS-based and composed reward shaping.

The experimental results show that this problem poses a challenge to model-free methods and is difficult to solve without properly used background knowledge.

In the above results the MDP-based reward shaping displayed a particularly worse performance than not only STRIPS-based but also less informed methods. A more detailed analysis was undertaken to look for the reason of this low performance.

Some conclusions can be drawn from the analysis of the histogram (see Figure 4) which shows how many times each abstract state was entered in both STRIPS-based and MDP-based approaches. The presented graph is for a single run of SARSA.

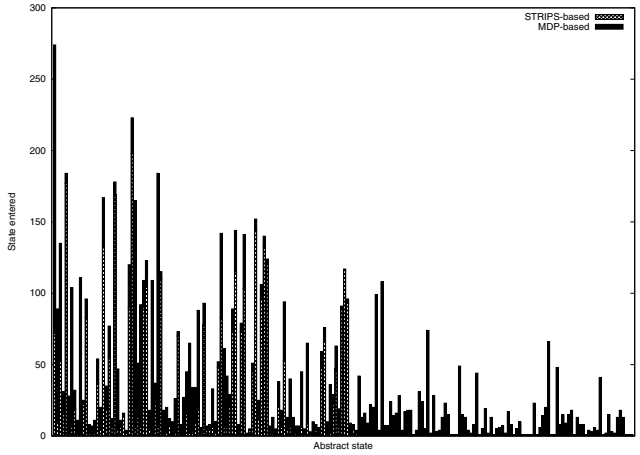

Fig. 4. Histogram presents how many times abstract states were entered during first 50 iterations of the single run of the SARSA algorithm.

The first observation from this experiment is that the algorithm with MDP-based plan tried many different paths, especially in the first episodes of learning. In the STRIPS-based case there is only one path along which potential increases. In the MDP-based case many different paths can be tried because the potential increases along many paths when moving towards the goal. When the agent moves away form the plan it can still find a rewarded path to the goal because the MDP-based policy defines an optimal path to the goal, not only from the start but from all states. This led to a rather "undecided" behaviour of the algorithm in the early stages of learning. The agent tries many different and advantageous paths, but because different paths are tried, they do not converge quickly enough (compare the number of steps made by SARSA with MDP-based shaping shown in Figure 6).

```
MOVE(hallA,hallB)
MOVE(hallB,roomC)
TAKE(flagC,roomC)
MOVE(roomC,roomE)
TAKE(flagE,roomE)
TAKE(flagF,roomE)
MOVE(roomE,roomC)
MOVE(roomC,hallB)
MOVE(hallB,roomB)
TAKE(flagB,roomB)
MOVE(roomB,hallB)
MOVE(hallB,roomA)
TAKE(flagA,roomA)
MOVE(roomA,hallA)
MOVE(hallA,roomD)
TAKE(flagD,roomD)
```

Fig. 5. The optimal STRIPS plan.

In effect, short and sub-optimal paths, like, e.g., the one that goes from start state S directly to goal G after taking flag D, quickly dominate because they lead to better performance than very long paths that collect more flags, because they have not converged yet. The histogram shown in Figure 4 provides more evidence for this hypothesis. First of all, it can be noticed that the number of visited abstract states is almost twice bigger in the MDP-based case. The agent considers a higher number of paths to be "interesting" in this case. In this particular run the number of visited abstract states was 106 in the STRIPS-based and 202 in the MDP-based case (there were 3141 and 6876 visited low level states respectively). Specifically, in the STRIPS-based case, the states that are visited when the optimal plan is followed, are those with the highest number of visits in the histogram. Other abstract states which also have high values in the histogram are adjacent to those which follow the optimal path. It is worth noting that states which follow the optimal path are not visited very often in the MDP-based case.

The main conclusion from this empirical analysis is that in the case of model-free RL algorithms and a difficult problem domain (in terms of exploration), it may be better to assign potential according to one particular path which can converge quickly rather than to supply many paths with increasing potential. The latter raises the probability of converging to a sub-optimal solution.

This observation suggests one potential improvement to MDP-based reward shaping when problems discussed here may arise. Instead of using the value function for the entire state space as potential, the best path which corresponds to the STRIPS plan can be extracted. When this path (in the same way as the STRIPS plan) is used with our algorithm to define potential, it can direct the agent in a more focused way toward the goal when it can be easily misled by a suboptimal result.

2) Model-based Methods: In our experiments, DynaQ represents the category of model-based reinforcement learning algorithms. Figure 7 shows the results with the different reward shaping techniques. The first observation is that DynaQ can deal much better with the problem domain than model-free SARSA. Even in the no-shaping case on average 5.4 flags were collected. The informed reward shaping methods (composed, MDP-based, STIRPS-based) performed better, showing the fastest increase in the obtained reward during almost the entire period of learning. With STRIPS-based and composed schemas to assign potential to abstract states, all six flags were collected in all ten test runs.

Results with model-free SARSA showed that reward shaping is essential in solving problems where it is easy to get stuck in a sub-optimal solution. Model-based methods like DynaQ make better use of what has been experienced during the learning process and make additional simulated backups over the state space. In this way it is possible to propagate information about highly rewarded areas even without visiting these areas many times (in a deterministic environment it is enough to make each transition once).

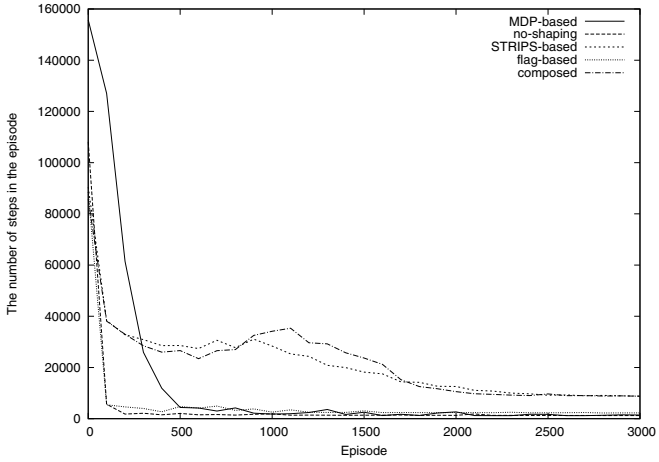

Fig. 6. The number of steps made by SARSA with different reward settings.

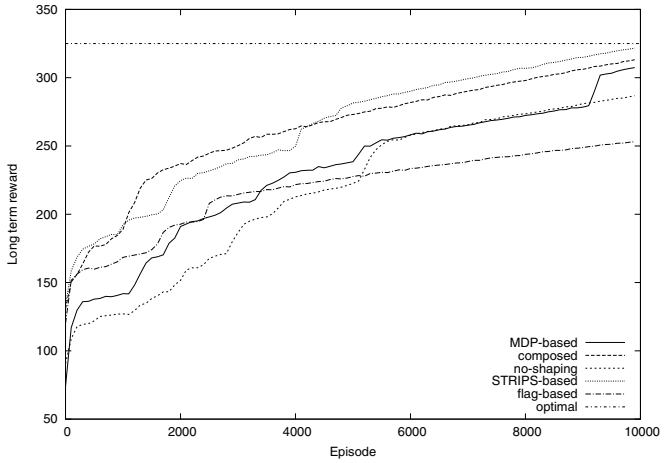

Fig. 7. DynaQ results with all reward configurations.

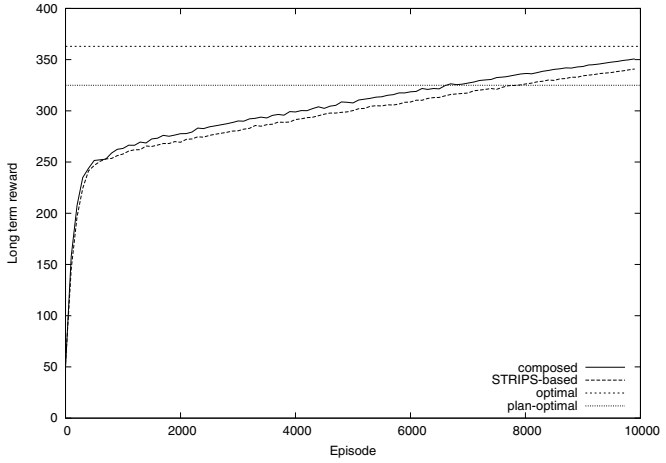

Fig. 8. SARSA learning when planner did not know about connection E and B.

B. Results with Sub-optimal Plans

In this section we take a closer look at various errors in the STRIPS-plans, that may be caused by incomplete or imprecise knowledge. Due to space restrictions, we do not show graphs for all experiments, but rather summarise the results in the text.

1) Plan Too Long: Incomplete knowledge about the environment can lead to the situation when the planner computes a longer plan than necessary. In the actual environment direct transitions from state z_i to state z_{i+k} where $k > 1$ may be possible, even though the planning knowledge did not include this fact.

In our experiments we created an additional transition from room E to room B, that has not been taken into account in the computation of the STRIPS-plan. After collecting two flags in room E the agent wants to collect flag B in room B. According to the plan it has to go through room C and hall B.

Empirical tests show that this kind of plan deficiency does not seem to cause problems for the RL agent (Figure 8 shows the results for SARSA). The transition from E to B when discovered is well rewarded because it has a higher difference in potential (6 in room E and 9 in room B).

The results for the other RL algorithms are similar and show the same trend.

2) Plan Assumes Impossible Transition: Incorrect knowledge can cause also the opposite effect: connections between two states that are assumed by the plan knowledge may not exist in the actual environment. In our experiments we created such a situation where the plan was computed assuming a connection between rooms E and B. The lack of this connection during learning is destructive for SARSA. However, model-based DynaQ finds a solution that is close to optimum and plan-based shaping performs better than no-shaping and flag-based shaping.

3) Missing Goal Conditions: This experiment evaluates the RL approaches when the plan was computed with a missing goal condition, thus potentially missing required actions, or including actions that are undoing part of the goal.

In our experiments we assumed that the information about flag B has not been given to the planner. The question is whether the learning agent is able to find the missing element through exploration. This is principally possible because the proposed schema to assign potential to non-plan states does not penalise moving away from the plan. The evaluation results show that the only configuration in our experiments that was able to perform better than the given (sub-optimal) plan was DynaQ with STRIPS-based shaping. Simulated backups led to the required propagation of the information about the discovered flag B.

4) Wrong Sequence: Even when high level knowledge about the domain is complete and the problem is specified correctly there is one more factor which may lead to sub-optimal policy at the low RL level. The main goal of classical planning algorithms is to find a plan which can transform the system from the start to the goal state. This achievement of the plan is usually satisfactory and the cost of actions is not taken into account in most STRIPS-based planners. In introduced in this paper application of classical planning this may lead to sub-optimal plans when high level actions can have different cost when implemented by low level primitive actions. To test our algorithm with these deficiency of plan, the experimental domain was modified in the following way. Halls A and B were merged into one hall and the high level plan was modified so flags were collected in the following order: B, A, C, E, F, D. This plan is clearly sub-optimal. Even though all flags are in the plan there is another plan that results in a shorter travelled distance. This setting was also difficult to tackle by most RL approaches. In this case, only model-based DynaQ with composed reward shaping was able to do better than the sub-optimal plan.

In summary, our results show that even when plans are not optimal or contain errors, RL algorithms are performing best when STRIPS-based reward shaping is used, but are not always able to converge to the optimum. Nevertheless, this can be satisfactory because the goal is often not to find the optimal solution but an acceptable policy in a reasonable amount of time.

VIII. Conclusion and Future Work

In this paper we show a new method to define the potential function for potential-based reward shaping, using abstract plan knowledge represented in the form of STRIPS operators. We empirically compared the performance of our proposed approach to RL without any reward shaping, RL with a manually shaped reward, as well as a related automatic shaping approach based on abstract MDPs [5]. The results of the experiments demonstrate that the STRIPS-based reward shaping improves both the quality of the learned policy, and the speed of convergence over the alternative techniques.

Overall, the results can be summarised as follows:

1) RL problems that are difficult in terms of exploration can be successfully tackled with model-free methods with plan-based reward shaping.
2) Model-based methods can find solutions to these problems without reward shaping in some cases, but reward shaping always speeds up learning.
3) STRIPS-based shaping showed better results than the MDP-based approach, because the agent was strongly influenced by the plan that guides it towards a good policy. Thus, this observation suggests one potential improvement to MDP-based reward shaping. Instead of using the value function of the entire state space as potential the best path which corresponds to the STRIPS plan can be extracted and used with our algorithm to define the potential.

Additionally STRIPS-based approaches can deal with much bigger state spaces at an abstract level because states are not explicitly enumerated. Symbolic planners can solve large problems (with huge state spaces) through their compact and highly abstract representations of states. Such planning together with model-free RL (with which STRIPS-based planning works well) can therefore be used with large state spaces and with function approximation in particular. It is worth noting that function approximation has been up to now used mainly with model-free RL algorithms and SARSA in particular [13].

STRIPS-based reward shaping is easier to scale up than, e.g., MDP-based reward shaping. For MDP-based abstract planning the state space has to be enumerated, which may require stronger abstraction or function approximation when applied to RL domains with very large state spaces. However, a positive feature of MDP-based planning is that it can deal in a natural way with different costs of high level actions (something that is more difficult to achieve with STRIPS).

Overall, STRIPS-based reward shaping can be seen as an alternative to MDP-based reward shaping with the proposed extension as both these techniques are planning methods. It is up to the domain expert which method to choose, depending on the form of available knowledge.

In future research we intend to investigate the ability of RL to explicitly correct errors in high-level plan knowledge and revise it based on the learning experience. Another future challenge is to apply plan-based reward shaping to multi-agent learning, using techniques from multi-agent planning [10].

References

[1] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the 16th International Conference on Machine Learning*, 1999, pp. 278–287.

[2] J. Randlov and P. Alstrom, "Learning to drive a bicycle using reinforcement learning and shaping," in *Proceedings of the 15th International Conference on Machine Learning*, 1998, pp. 463–471.

[3] M. Grounds and D. Kudenko, "Combining reinforcement learning with symbolic planning." in *Fifth European Workshop on Adaptive Agents and Multi-Agent Systems*, 2005.

[4] M. R. K. Ryan, "Using abstract models of behaviours to automatically generate reinforcement learning hierarchies," in *Proceedings of the 19th International Conference on Machine Learning*, 2002, pp. 522–529.

[5] B. Marthi, "Automatic shaping and decomposition of reward functions," in *Proceedings of the 24th International Conference on Machine Learning*, 2007, pp. 601–608.

[6] C. Boutilier, T. Dean, and S. Hanks, "Decision-theoretic planning: Structural assumptions and computational leverage," *Journal of Artificial Intelligence Research*, vol. 11, pp. 1–94, 1999.

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, March 1998.

[8] R. S. Sutton, D. Precup, and S. P. Singh, "Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[9] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artificial Intelligence*, vol. 90, pp. 281–300, 1997.

[10] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, December 2002.

[11] R. Dearden, N. Friedman, and S. J. Russell, "Bayesian Q-learning," in *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. AAAI, 1998, pp. 761–768.

[12] M. J. A. Strens, "A bayesian framework for reinforcement learning." in *Proceedings of the 17th International Conference on Machine Learning*, 2000, pp. 943–950.

[13] P. Stone, R. S. Sutton, and G. Kuhlmann, "Reinforcement learning for RoboCup-soccer keepaway," *Adaptive Behavior*, vol. 13, no. 3, pp. 165–188, 2005.