

Structural Abstraction Experiments in Reinforcement Learning

Robert Fitch¹, Bernhard Hengst¹, Dorian Šuc¹, Greg Calbert², and Jason Scholz²

¹ National ICT Australia, University of NSW, Australia

{`robert.fitch,bernhard.hengst,dorian.suc`}@nicta.com.au

² Defence Science and Technology Organization, Salisbury SA Australia

{`greg.calbert,jason.scholz`}@dsto.defence.gov.au

Abstract. A challenge in applying reinforcement learning to large problems is how to manage the explosive increase in storage and time complexity. This is especially problematic in multi-agent systems, where the state space grows exponentially in the number of agents. Function approximation based on simple supervised learning is unlikely to scale to complex domains on its own, but structural abstraction that exploits system properties and problem representations shows more promise. In this paper, we investigate several classes of known abstractions: 1) symmetry, 2) decomposition into multiple agents, 3) hierarchical decomposition, and 4) sequential execution. We compare memory requirements, learning time, and solution quality empirically in two problem variations. Our results indicate that the most effective solutions come from combinations of structural abstractions, and encourage development of methods for automatic discovery in novel problem formulations.

1 Introduction

When specifying a problem such as learning to walk, learning to manipulate objects or learning to play a game as a reinforcement learning (RL) problem, the number of states and actions is often too large for the learner to manage. It is straightforward to describe the state of a system using several variables to represent the various attributes of its components. Similarly, individual system component actions can be used to describe a joint action vector. However, this approach very quickly leads to an intractable specification of the RL problem. A tabular state-action representation requires storage proportional to the product of the size of all the state and action variables, leading to intractable storage and time complexity.

Function approximation can often help by generalizing the value function across many states. Function approximation is based on supervised learning. Gradient descent methods, such as artificial neural networks and linear function approximation are frequently used in RL for this purpose [1]. However, there are reasons to believe that simple function approximation will not scale to larger, more complex, problems.

Some problems may have structure and regularities that are not readily apparent and difficult to learn in one step [2]. Learning may be made tractable by the right problem decomposition and by learning in stages, reusing and combining previously learnt concepts. Learning in complex environments proceeds at the “frontier of receptivity” where new tasks can only be learnt once component child tasks have been learnt [3].

Our objective in this paper is to explore several structural abstraction methods and to study their effect on storage requirements, learning time and the quality of the solution for one problem with two different levels of inter-dependency. The contribution of this paper is to give a systematic description and experimental evaluation of a collection of known structural abstractions that can be used in a wide variety of RL problems. Our motivation comes in part from the desire to develop tractable decision systems for complex multi-agent games, and secondly to gain insight for automating the modelling process.

Our experiments in this paper are based on a task that requires two taxis acting together to pickup and deliver two passengers on the 5×5 grid. This task is chosen because it is intuitive and small enough so that we can still compute the optimal solution, yet it is complex enough to make it challenging to demonstrate various abstractions.

The formalisms underpinning the experiments in this paper are those of Markov decision problems (MDPs) and model reductions expressed as homomorphic mappings. The latter are important in establishing whether one system is a model of another and which properties of the original system the model retains [4]. The abstractions we will explore are symmetry reductions, multi-agent decompositions, hierarchical decompositions, sequential execution and some combinations.

In the rest of this paper we will first introduce the two-taxi task as our working example and summarize the formalisms. We then in turn describe the various structural abstractions of the problem including related work. In each case we discuss the computing requirements in relation to solution quality for the taxi task and comment on the scaling potential.

2 The Two-Taxi Problem

Taxi problems have been used by several researchers in related work. They have analogues in other problems, such as logistic problems involving transporters and cargo. We have taken two taxis [5] working jointly to pick up and deliver two passengers on the 5×5 grid shown in Figure 1.

The objective is to deliver both passengers in the minimum number of steps. At each time step a taxi can take a navigation action and attempt a move one position North, South, East or West. These actions are stochastic. With 92.5% probability they move the taxi in the intended direction and with 7.5% probability they move the taxi randomly in one of the other three directions. Other actions available are to pickup passenger one, pickup passenger two, put down its passenger or stay. Taxis can only carry one passenger at a time. A move

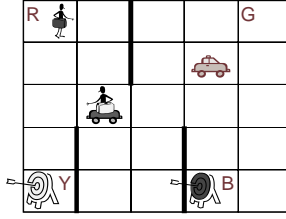


Fig. 1. The two-taxi Problem.

into a barriers or the grid boundary is counted as a step and results in the taxi staying where it is. There are four special locations marked R, G, Y, B that represent the possible source and destination locations of the passengers. At the commencement of every episode each passenger is located at random on one of the special locations and each taxi is located at random anywhere on the grid. Each passenger must be picked up by one of the taxis and put down at his or her destination location, even if the source and destination locations are the same. If a passenger is waiting to be picked up and both taxis are at this location then a joint action by both taxis to pick up the passenger results in only taxi one being successful.

We consider two versions of the problem, with and without taxi collisions. Without collisions the taxis are able to occupy the same cell and pass each other without impediment. With collisions the taxis cannot occupy the same cell nor can they pass each other in the sense that they cannot occupy each other's former cell following a joint action. A move resulting in a collision leaves the taxi locations unchanged. Initial taxi positions on the same cell are disallowed with collisions, although passengers can have the same pickup and destinations locations.

3 Modelling Formalisms

We now introduce RL, its application to the two-taxi task and abstractions formalized as homomorphisms.

3.1 RL Problem Formulation

Underlying RL is the Markov decision problem framework. We consider a system modelled at a base level by a set of states. At each time-step the system transitions from state to state depending on an input action to the system and generating a real-valued reward. We can describe a Markov decision problem (MDP) as a tuple $\langle S, A, T, R, S_0 \rangle$ where S is the finite set of states, A is the finite set of actions, $T : S \times A \times S \rightarrow [0, 1]$ is the transition function giving the probability of moving from one state to another after taking a particular action, $R : S \times A \times R \rightarrow [0, 1]$ is the probability of receiving a reward when taking an action, and $S_0 : S \rightarrow [0, 1]$ is the starting state distribution giving

the probability of starting in each state. The Markov property requires that the states and actions can be defined so that the probabilities are independent of the history of transitions and rewards. In addition we assume the probabilities are independent of time, i.e. stationary. An MDP includes an optimality criterion, usually to maximize a function of the sum of future rewards. The objective is to find an optimal policy $\pi : S \rightarrow A$ providing the action to take in any state to optimize the value function. A semi-MDP is a generalization of an MDP where actions can be *extended*, taking several time steps to complete.

Specifying the two-taxi problem above as an MDP we proceed to define the state $s \in S$ by the variables $(t_1, t_2, p_1, p_2, d_1, d_2)$, where t_1 and t_2 each specify the respective taxi location as one of the 25 grid values, p_1 and p_2 indicate the location of the passengers at either one of the four pickup locations, riding in either taxi or delivered, and d_1 and d_2 each specify one of the four destinations of the respective passengers. The total number of possible collision-free states is therefore $25 \times 25 \times 7 \times 7 \times 4 \times 4 = 490,000$. The actions $a \in A$ are defined by the variables (a_1, a_2) representing the simultaneous joint actions of the two taxis. There are eight actions available per taxi giving a joint action set of 64. At each time step we specify a joint reward of -1 until both passengers are delivered. Maximizing the sum of future rewards achieves our objective to minimize the number of steps to complete the task.

We assume the learner can fully observe the state, but that the transition and reward functions are not known beforehand. We use simple *Q-learning* [6] to learn the optimal policy as our base case. Q-learning involves updating an action-value function stored as a *Q-table* with one entry for each state action pair. The update performed after each step is called a Bellman backup and given by $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a'))$ where the system has transitioned from s to s' after taking action a and receiving reward r . The discount rate for future rewards is γ and set to 1.0 in our experiments. The learning rate is α . It is well known that given sufficient exploration of all states and actions and suitable reduction in the learning rate the action-value function will converge to its optimal value, Q^* , and the optimal policies are $\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$. While there are many speedup techniques such as prioritized sweeping and eligibility traces, we have used this simple form of RL with $\alpha = 0.1$ in all our experiments for consistency. Our exploration policy is ϵ -greedy with ϵ generally set to 10%. All performance is measured using greedy policies. We are primarily interested in comparing learning efficiency with various model abstractions and have found that similar relationships hold for any setting of α and adequate exploration.

3.2 Abstractions as Homomorphisms

The idea of an abstraction is to simplify the representation of a system in such a way that the problem of interest can be solved more efficiently. Ideally we would like a model to perfectly preserve the system characteristics of interest so that we only need to solve the simplified model and the solution is guaranteed to be a

solution for the original system. A useful algebraic formalism for modelling abstractions is a homomorphism. We are particularly interested in homomorphisms in the framework of MDPs and semi-MDPs. A homomorphic mapping from a system to a model means that if we perform an action or temporally extended action in a system state then the resultant model state is the same whether we map the resultant system state to the model or whether we first map the system state and (extended) action to the model and perform the model action. State transition homomorphisms by themselves do not guarantee that the reduced model is relevant to the problem at hand. An MDP homomorphism ensures relevancy by also mapping the reward function and thus preserving the implicit goal specification. A useful exposition of MDP and semi-MDPs is given by Ravindran [7].

4 Abstraction Classes

We study several exact and approximate model abstractions (homomorphisms) and apply them to the two-taxi problem. The results are shown in Table 1 and Figure 2 and explained in this section.

The base case Q-table size for the collision-free two-taxi problem is $490,000$ states \times 64 actions = $31,360,000$ entries. With collisions the base case Q-table size is $30,105,600$ as taxis cannot occupy the same state. We have calculated the optimal solution using dynamic programming for both base cases. Q-learning on the base case takes over four billion time steps to converge. We have defined convergence to be the number of time-steps required for the mean of the results to be within 5% of its final value. Q-learning oscillates slightly short of optimal performance with the learning rate α fixed at 0.1 (see Table 1, “Base Case”). For the base case and other abstractions we have found that the mean converged average steps per episode over multiple runs has a standard deviation of less than 0.06.

4.1 Symmetry

Symmetries in MDPs arise when states, actions or a combination of both can be mapped to an equivalent reduced MDP that has less states and actions. For example, if two actions have identical transition and reward probability functions between any two states then one action can be eliminated. An example of a problem with state symmetry is learning to exit similar rooms that differ only in color. More general symmetries often arise where state-action pairs are mapped together both with different states and actions from the original in the reduced model [8]. We call these forms of symmetry simple state-action symmetries. Another class of symmetry that overlaps the class of simple state-action symmetry is bisimulation homogeneity [9]. In bisimulation homogeneity the states of a problem can be partitioned into blocks such that the inter-block transition probabilities and reward probabilities are constant for all actions. An

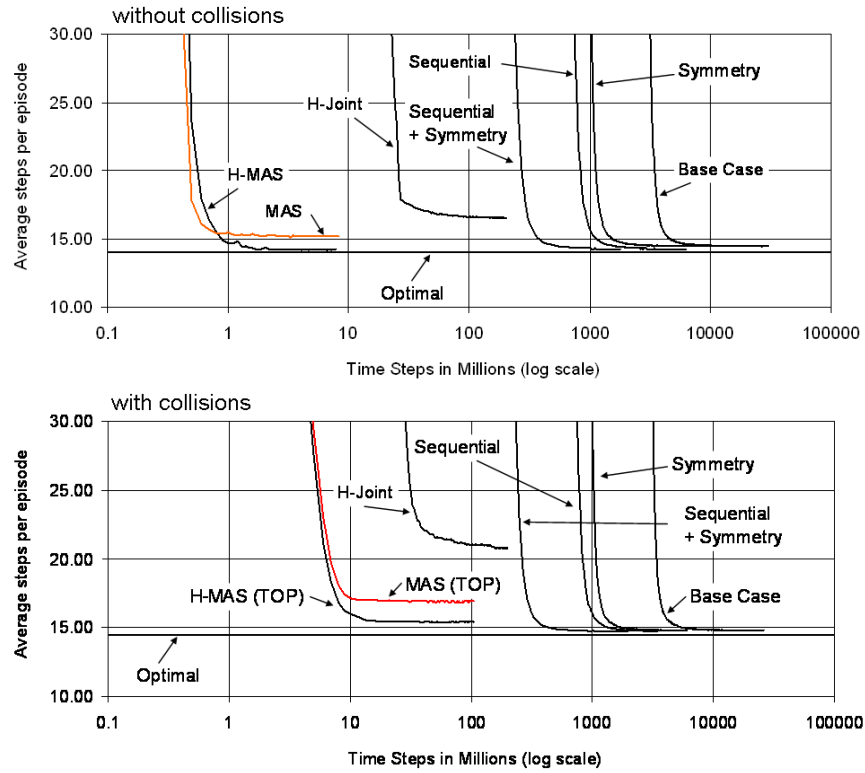


Fig. 2. Two-taxi task results: without collisions (top) and with collisions (bottom).

example of model reduction using this type of symmetry is the elimination of an irrelevant random variable in a state description.

The taxi problem exhibits simple state-action symmetry assuming homogeneity of taxis and passengers. In one form of symmetry two different navigation actions by the taxis can be interchanged without affecting the solution whenever both taxis occupy the same location. If the taxis are at different locations and they are interchanged the two situations are also symmetrical and can be mapped to the same abstract state. The second mapping will not reduce the states if both taxis are on the same cell, a situation that can only arise in the collision-free problem. Similarly, swapping passenger-destination pairs can be exploited as a state-action symmetry abstraction, as can swapping both taxis and passengers. Specifically, state $(t_1, t_2, p_1, p_2, d_1, d_2)$ with action (a_1, a_2) is symmetrically equivalent to $(t_2, t_1, p_1, p_2, d_1, d_2), (a_2, a_1)$ where p_1 and p_2 need to be adjusted whenever the passengers are in the taxis. Similar homomorphic state-action mappings can be defined for a passenger swap and for both a taxi and passenger swap providing a 4-to-1 state reduction mapping when taxi and passenger-destination locations differ.

The above abstraction reduces the two-taxi problem without collisions to 131,950 states compared to 490,000 for the base case. The approximately 3.7-fold

Table 1. Memory, learning times and performance of various abstractions on the two-taxi tasks for *collision free* (left) and *with collisions* (right). Both cases use the same abstractions, except with MAS and H-MAS, where in the case *with collisions* also the position of the other taxi (TOP) is included in the state.

	COLLISION FREE			WITH COLLISIONS		
	MEMORY ($ Q $)	STEPS TO CONVERGE	CONVERGED AVE. STEPS PER EPISODE	MEMORY ($ Q $)	STEPS TO CONVERGE	CONVERGED AVE. STEPS PER EPISODE
OPTIMAL	3.1×10^7	DP	14.06	3.0×10^7	DP	14.46
BASE CASE	3.1×10^7	4.4×10^9	14.46	3.0×10^7	4.2×10^9	14.80
SYMMETRY	8.4×10^6	1.5×10^9	14.45	7.2×10^6	1.3×10^9	14.82
MAS	4.2×10^3	7×10^5	15.18	1.0×10^5	9×10^6	16.90
H-JOINT	2.7×10^5	3.4×10^7	16.56	2.0×10^5	4.6×10^7	20.78
H-MAS	5.7×10^3	9×10^5	14.24	1.1×10^5	1×10^7	15.41
SEQUENTIAL	7.8×10^6	1.2×10^9	14.26	7.5×10^6	1.1×10^9	14.76
SEQ.+SYM.	2.1×10^6	3.9×10^8	14.27	1.8×10^6	3.5×10^8	14.75

reduction in the number of states in the reduced model leads to a commensurate speedup in learning as illustrated in Figure 2. The symmetries also apply to the taxi task with collisions with similar convergence speedup results. These state and action abstractions are an exact homomorphism of the original MDP and therefore converge to an optimal policy of the original problem.

For the general collision-free task involving n taxis on a size k grid with m passengers and l special locations the number of states is $k^n \cdot (l + n + 1)^m \cdot l^m$. This can be shown to abstract to $C_n^{k+n-1} \cdot C_m^{(l+n+1)l+m-1}$ states³, when taxis and passengers are indistinguishable.

An approach to generalizing exact symmetries is to consider approximate symmetries. A bounded parameter Markov decision process [10] can be used to bound the error when inter-block transitions and rewards probabilities are not constant but constrained to a range of values. Symmetries will only take us so far in model reduction. We will now turn to another type of structural abstraction.

4.2 Multiagency

Modelling a system as a simultaneous interaction of multiple autonomous agents can be a significant abstraction [11]. When a problem is specified using a multi-dimensional action description, it is natural to attempt this type of decomposition.

In multi-agency the overall homomorphism is approximated by combining independent homomorphic mappings for each agent. One of the major issues is the abstraction of the global reward signal for each agent so that their combined behavior is in the common good [12]. If the actions of all the agents are independent and the global reward is the sum of the rewards for each agent then the multi-agent homomorphism is exact and will produce an optimal solution.

³ C_k^n (n choose k) = $n!/(k!(n-k)!)$

This total decomposition is rare for interesting problems and does not hold for either version of the two-taxi task. For optimal performance the taxis need a coordinated strategy for picking up passengers and need to avoid each other when collision is possible.

One homomorphic approximation arbitrarily pre-allocates one passenger to each taxi. One mapping is:

$$\begin{aligned} h((t_1, t_2, p_1, p_2, d_1, d_2) \xrightarrow{(a_1, a_2)}) &= (t_1, p_1, d_1) \xrightarrow{a_1}, (t_2, p_2, d_2) \xrightarrow{a_2} \\ &= (t, p, d) \xrightarrow{a}, (t, p, d) \xrightarrow{a} \quad (\text{using symmetry}) \end{aligned}$$

Each agent receives the global reward. Effectively we have defined two reduced MDPs (one for each agent) that together comprise the two-taxi problem. When the two agents are identical we can additionally apply symmetry from Section 4.1 and use only one MDP for both agents. This not only saves storage space, but transfers learning between the two agents. The actions per taxi can be reduced by one as they now do not need to specify which passenger is to be picked up. This means that only $25 \times 6 \times 4$ states and 7 actions are required to be modelled reducing the number of table entries to 4,200 compared to the original 31,360,000. The reduced MDP in effect solves Dietterich’s original single taxi problem.

The joint action policy required for the original MDP is formed by combining the actions from the reduced MDP, primed respectively with the two sets of agent states. Each taxi agent models the world by focussing on its passenger and ignores the other agent and passenger. It is rewarded for delivering its passenger.

Figure 2 shows the results for a multi-agent system (MAS) learner with the above abstraction using the fixed allocation policy. Fixing the allocation of the taxis to passengers is clearly suboptimal. We can do better if we learn a joint dispatch strategy as a higher level decision task as described in the next section.

For the case with collisions the previous model of the partially observable environment from the point of view of one taxi is now inadequate. Collisions depend on the other taxi that cannot be observed. The model is no longer Markov and the homomorphism fails. When the outcome for one agent depends on the actions of the others then it is necessary to include extra state information to try to retain the individual agent homomorphic mappings⁴. To address this situation we have included the other taxi’s position (TOP) as an integral part of the state description for each agent. This is an approximation, modelling the other agent’s movements as stochastic. The resulting reduced model requires the least amount of storage in our experiments with collision and produces reasonably good results as shown in Figure 2 MAS (TOP). The abstraction also requires us to dispatch the taxis to a fictitious passenger pickup location after they complete their mission to avoid each taxi inadvertently blocking the other.

⁴ However, even when all information is taken into account and the individual agent models are exact, if agents behave greedily the overall abstraction may only be approximate and produce suboptimal solutions. This effect is well known as Voter’s Paradox or the Tragedy of the Commons and has consequences such as Braess Paradox [13] where more options can reduce overall performance.

The requirement for a better dispatching policy and Dietterich’s original decomposition of the single taxi problem suggest we can do better by turning to another form of structural abstraction – hierarchical decomposition.

4.3 Task Hierarchies

A problem may be able to be decomposed into a task hierarchy in which each parent task can choose to execute any of its child subtasks. Dietterich introduced the MAXQ decomposition to compactly represent a value function over a task hierarchy with reusable subtasks [5]. The whole task hierarchy is a semi-MDP homomorphism that maps child task policies to temporally extended actions at the root node. The homomorphism is in general approximate in that a task hierarchy may constrain the policies available in the original problem. Indeed, each parent task in the hierarchy can be interpreted as a sub-model of the problem using a semi-MDP homomorphism where child policies are mapped as the parent’s abstract actions. Learnt from the bottom up, a task hierarchy implements the “frontiers of receptivity” [3] in a multi-layered learning paradigm.

One such task hierarchy decomposes the two-taxi problem into two levels of subtasks as shown in Figure 3. The lower level navigation subtasks with states (t_1, t_2) and actions (a_1, a_2) has the objective of jointly moving the taxis to the four special locations. As the actions are joint they can handle collisions. The joint navigation subtasks terminate when both taxis have reached their designated special locations. If the taxis are interpreted as separate agents then this model implicitly implements the concurrent action model T_{all} termination scheme [14]. The termination scheme requires agents to terminate simultaneously and is clearly suboptimal. However, forced synchrony reduces the number of sub-MDPs and abstract actions required in the hierarchy. Sixteen sub-MDPs are needed at level one for the collision-free problem and 12 with collisions, representing the various termination combinations at the four special locations for the two taxis.

The level two dispatch subtask uses states (p_1, p_2, d_1, d_2) and policies from level one generating 108 and 144 abstract actions on the task with and without collisions respectively. These represent the combination of pickup and putdown actions upon termination of the sub-MDPs. We have used a semi-automatic version of the HEXQ algorithm [15] for our implementation. Results are labelled “H-Joint” in Table 1 and Figure 2 and show about two orders of magnitude savings in both memory and learning time but are clearly suboptimal as expected. The HEXQ algorithm constructs the task hierarchy while learning but its variable-wise decomposition does not produce a multi-agent decomposition. The large number of abstract actions makes execution expensive. This type of joint action hierarchy could be abstracted further by using symmetry as discussed in Section 4.1 but this was not implemented.

Our second hierarchical abstraction is shown in Figure 3 and builds on the multi-agent abstraction from the previous section. In this three level task hierarchy the level one subtasks are the individual taxi agents performing the whole delivery task. We only use one Q-table for both taxis as discussed in Section

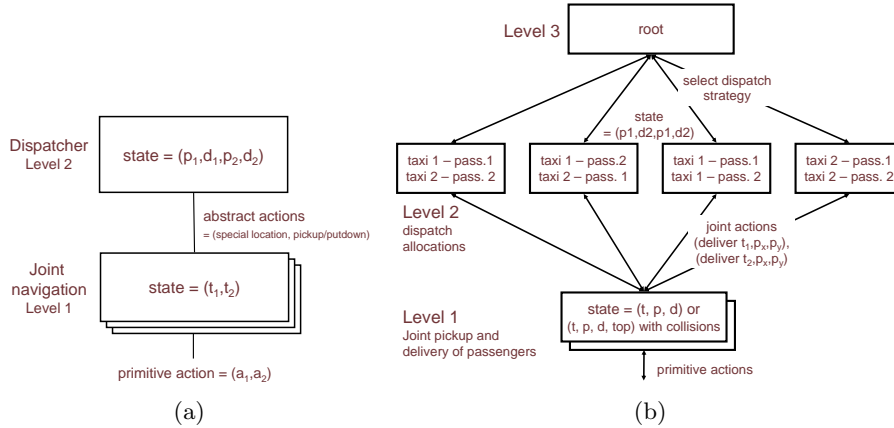


Fig. 3. Dispatcher and joint navigation task hierarchy (H-Joint) is shown in (a); (b) shows three-level hierarchy using multi-agent travelling salesman dispatcher and whole delivery task at lower level (H-MAS).

4.2. The second level tasks represent the various dispatching options for the two taxis. There are four options altogether allocating each taxi to deliver both passengers or one or the other. In the event that one taxi is to deliver both passengers, the level two tasks learn the best order of delivery. This problem is akin to a multi-agent travelling salesman problem (TSP) in which each city needs to be visited by one agent only. A city visit corresponds to a passenger pickup and delivery. The root task simply chooses one of the dispatch strategies.

The results are depicted in Figure 2 and Table 1 as H-MAS and H-MAS(TOP). This task hierarchy gives near optimal results for the collision-free version of the problem. It is suboptimal because, due to stochastic drift, it is possible that the optimal dispatch decision should be switched prior to pickup but is locked in by the second level controller in the hierarchy. We conjecture to be able to improve the results, achieving optimality in this case, by using hierarchical greedy execution (polling) [16, 5].

The H-MAS structural abstractions combine hierarchy, multi-agency and symmetry and achieve the best resource effectiveness with and without collisions. Interestingly, storage can be reduced further by more than a factor of four and learning time by at least a factor of two by decomposing the level one subtask in Figure 3 into three levels as for the original taxi task [5], creating a five level hierarchical decomposition for the two-taxi task.

4.4 Sequential Execution

The idea behind sequential execution is to “simulate” the concurrent action of the agents by timesharing their execution. This means that the agents act one at a time as far as the learner is concerned, but are assumed to be able to execute jointly in a real world.

This is a semi-MDP homomorphism in which one joint action is mapped to a two step temporally extend action. The advantage is that the number of actions is the sum rather than the product of the number of actions of the agents. In effect we are assuming that the problem can be approximated with a reduced action set with joint actions $(a_1, stay)$ and $(stay, a_2)$.

With and without collisions, this reduces the memory use and learning time by about a factor of four and achieves near optimal results. In combination with symmetry we achieve the multiplicative benefit of both abstractions. These results are labelled “Sequential” and “Seq.+Sym.” in Table 1 and Figure 2.

5 Automatic Discovery of Abstractions

One of the major lessons from implementing these abstractions, even on this relatively simple example, is the difficulty in manually specifying the various decompositions. For example, the symmetry abstractions require both states and actions to be mapped together, but it is not straightforward to specify the complete homomorphic mapping in Section 4.1. Whenever the user provides an abstraction we are effectively providing background information to the learner and this must be manually tailored for each problem. Both these issues motivate automating the discovery of structural abstractions.

Our approach to automation would try to exploit any structure in a problem by searching for symmetry, multi-agent and hierarchical abstractions. We may, for example, be able to find sub-models that are well defined homomorphisms covering parts of the whole problem. These sub-models may be able to be found by simultaneously exploring the projected spaces defined by subsets of state-action variables and testing for Markov transitions and rewards. The sub-models become the building blocks for a search for even more abstract models at a higher level. In this way it is envisaged that a task hierarchy could be shaped from the bottom up to solve the problem efficiently.

6 Discussion and Future Work

This paper presents classes of abstraction useful for scaling up RL and points towards promising future work in two directions – the manual use of structural abstractions, and their automatic discovery. Our experimental results show that the performance gains of single abstractions can be further improved through their combination. Our best results combine symmetry, task hierarchy and multi-agency. We have already indicated a clear opportunity for improving these results in Section 4.3 and believe there are further model reduction and scaling opportunities available. The model of the other agent (TOP) in our implementation uses a global position variable. We conjecture that it is possible to parsimoniously use only a “window of influence” for the TOP model, possibly a deictic representation, that would reduce memory requirements significantly. Further, given the correspondence to multi-agent TSP, this logistic problem should scale tractably with good results using many vehicles and cargo in much larger simulations.

The expected gains from abstractions depend on the degree of system coupling. In the experiments we found that increasing interdependency reduces the number of reachable states because of the added constraints but also creates less opportunity for abstraction.

The experiments and forgoing reasoning give some hope that we can scale RL to larger problems by using manually defined abstractions. The unifying framework of treating model reduction as a homomorphism suits and supports structural abstraction well. Our experiments also indicate the feasibility of developing methods for automatic discovery of abstractions as outlined in Section 5. We are currently exploring both of these approaches in complex domains such as battle-space simulations.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, Massachusetts (1998)
2. Clark, A., Thornton, C.: Trading spaces: Computation, representation, and the limits of uninformed learning. *Behavioral and Brain Sciences* **20** (1997) 57–66
3. Utgoff, P.E., Stracuzzi, D.J.: Many-layered learning. In: *Neural Computation*, MIT Press Journals (2002)
4. Ashby, R.: Introduction to Cybernetics. Chapman & Hall, London (1956)
5. Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* **13** (2000) 227–303
6. Watkins, C.J.C.H.: Learning from Delayed Rewards. PhD thesis, King’s College (1989)
7. Ravindran, B., Barto, A.G.: SMDP homomorphisms: An algebraic approach to abstraction in semi markov decision processes. In: *Proc. of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI 03)*. (2003) 1011–1018
8. Ravindran, B., Barto, A.G.: Model minimization in hierarchical reinforcement learning. In: *Fifth Symposium on Abstraction, Reformulation and Approximation (SARA 2002)*. LNCS, Springer Verlag (2002) 196–211
9. Dean, T., Givan, R.: Model minimization in markov decision processes. In: *AAAI/IAAI*. (1997) 106–111
10. Givan, R., Leach, S.M., Dean, T.: Bounded-parameter markov decision processes. *Artificial Intelligence* **122** (2000) 71–109
11. Crites, R.H., Barto, A.G.: Elevator group control using multiple reinforcement learning agents. *Machine Learning* **33** (1998) 235–262
12. Wolpert, D., Tumer, K.: An introduction to collective intelligence. Technical Report NASA-ARC-IC-99-63, NASA Ames Research Center, CA (1999)
13. Braess, D.: Über ein Paradoxon der Verkehrsplanung. *Unternehmensforschung* **12** (1968) 258–268
14. Rohanimanesh, K., Mahadevan, S.: Learning to take concurrent actions. In: *NIPS*. (2002) 1619–1626
15. Hengst, B.: Discovering hierarchy in reinforcement learning with HEXQ. In Sammut, C., Hoffmann, A., eds.: *Proceedings of the Nineteenth International Conference on Machine Learning*, Morgan-Kaufman (2002) 243–250
16. Kaelbling, L.P.: Hierarchical learning in stochastic domains: Preliminary results. In: *Machine Learning Proceedings of the Tenth International Conference*, San Mateo, CA, Morgan Kaufmann (1993) 167–173