Proceedings of the Tenth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2014)

Learning Micro-Management Skills in RTS Games by Imitating Experts

Jay Young and Nick Hawes

Abstract

We investigate the problem of learning the control of small groups of units in combat situations in Real Time Strategy (RTS) games. AI systems may acquire such skills by observing and learning from expert players, or other AI systems performing those tasks. However, access to training data may be limited, and representations based on metric information – position, velocity, orientation etc. – may be brittle, difficult for learning mechanisms to work with, and generalise poorly to new situations. In this work we apply *qualitative spatial relations* to compress such continuous, metric state-spaces into symbolic states, and show that this makes the learning problem easier, and allows for more general models of behaviour. Models learnt from this representation are used to control situated agents, and imitate the observed behaviour of both synthetic (pre-programmed) agents, as well as the behaviour of human-controlled agents on a number of canonical micro-management tasks. We show how a Monte-Carlo method can be used to decompress qualitative data back in to quantitative data for practical use in our control system. We present our work applied to the popular RTS game Starcraft.

Introduction

In Real-Time Strategy games, the overall skill of a player is typically thought of as being based on their competency in two broad areas: micro-management and macro-management. Macro-management concerns training units, building structures, selecting upgrades, scouting and other high-level issues relating to a strategy. Micro-management on the other hand concerns the task of managing single units, or groups of units, and issuing tactical commands during combat – unit movement, targeting commands, use of special abilities etc. and requires continuous input. It is essential that a player (either AI or human) has strong skills in both areas in order to be successful. Good micro-management skills are those which best exploit the characteristics of the units available to the player, as well as the environment, in order to overcome an opponent in combat situations.

One common approach to implementing micro-management in AI systems is to apply distributed reactive control mechanisms. Co-operation between units is then achieved through the emergent properties of a set of rules followed by each unit, rather than being explicitly planned or pre-scripted, which can result in complex behaviour (an example of one of our own systems can be seen at: https://vimeo.com/45580695). Human players also utilise similar reactive approaches when micro-managing units in combat, due to the need for continuous input to deal with highly dynamic, ever-changing conditions.

In our work, we are interested in how an AI player can *learn* to play Real-Time Strategy games by observing either humans or other AI systems play. In this paper, we focus on the problem of learning unit micro-management tactics by observation of experts. We are interested in doing this using *qualitative spatial representations* – abstraction methods which we introduce shortly, allowing us to address several problems associated with the learning task.

We make the following contributions:

- A novel integration of qualitative spatial representations for learning by observation in Starcraft.
- Construction of benchmark agents which perform well on tasks characteristic of the micromanagement domain, for which we make full source code available for future researchers.
- A study of the problem of learning by observation from both the above synthetic controllers and *human* controllers on three micro-management tasks.

Starcraft

Starcraft is a Real-Time Strategy game released by Blizzard Entertainment in 1998[1]. The game requires a human player to engage in goal-directed planning, reasoning under uncertainty, creative adaptation, and other tasks (specifically our micro-management) requiring continuous input. This must be accomplished in real-time, which exasperates many of the already difficult AI challenges present. In recent years the game has become the focus of interest from the AI community, supported by the release of the Brood-War API (BWAPI[2]), a software API which allows for the injection of code into the Starcraft game engine, and facilitates the production of third-party AI players.

[1]http://us.blizzard.com/en-us/games/sc/

[2]http://code.google.com/p/bwapi/

Learning by observation

Engineering AI systems is a non-trivial task, which is why much research focuses on building systems that can *learn* solutions to problems. Learning-by-doing techniques (e.g. reinforcement learning) can be risky however, due to the need for trial-and-error, which can be dangerous and expensive in certain domains, such as Robotics (Argall et al. 2009), and often presume the existence of some global objective function which can be consulted an unlimited number of times. If expert agents already exist for the target domain, then an alternative approach is for a system to *learn by observation* (Montana and Gonzalez 2011), by observing and mimicing those expert agents. An expert agent can be considered to be an agent skilled enough to behave competently at the given task, though with no guarantees made as to whether the agent behaves optimally or not in reference to a particular task.

We adopt the following learning by observation formalism of Montana and Gonzalez (2011). Let B_C be the *behaviour* of an expert agent C, and let E be an environment. A behaviour may be a controller, policy or algorithm that the expert uses in order to determine which actions to execute. During operation, this behaviour is expressed as a Behaviour Trace BT representing the actions the expert performs over a period of operation. The behaviour trace of an expert is then:

$$BT_C = [(t_1, x_1), ..., (t_n, x_n)]$$

where t is a time index, and x is an action. The change in the environment E over time is represented by an input trace:

$$IT = [(t_1, y_1), ..., (t_n, y_n)]$$

The combination of a behaviour trace and an input trace produces a Learning Trace LT:

$$LT = [(t_1, x_1, y_1), ...(t_n, x_n, y_n)]$$

Consisting of, at any one time, the state of the environment E and the action the expert agent performs in that state. The learning by observation task is to learn a behaviour B which behaves the same way as the expert agents in the environment E, given the same inputs $LT_1, ..., LT_n$

This formalism applies to many scenarios in which learning by observation may be used over other methods, e.g. a sports robot that learns the behaviour of opponents by watching replays of past games, or a disaster relief robot in an ad-hoc team which requires training in the field. Learning by observation can be extremely challenging, particularly if the behaviour of a given expert agent is non-deterministic. A key challenge is to find the correct *knowledge representation* in which the environmental observations are delivered.

Related Work

Qualitative representations (QRs) have been used previously in learning tasks across of a variety of domains. For example, for learning and recognition of events in an aircraft loading domain (Sridhar and Cohn 2010) and desktop activities (Behera, Cohn, and Hogg 2012), and for reinforcement-learning for robot navigation (Frommberger 2008). The problem of behaviour modelling in RoboCup Soccer, another simulated domain, has been tackled in a number of ways including reinforcement learning (Vieira, Adeodato, and Gon 2010; Jung and Polani 2012; Shimada, Takahashi, and Asada 2010; Molineaux, Aha, and Sukthankar 2008), and case-based reasoning (Ros, Llu, and Mantaras ; Floyd, Esfandiari, and Lam 2008). However in general, little work investigates the application of qualitative, relational representations to the problem of learning by observation. In Starcraft, recent work by (Parra and Garrido 2013) makes use of Bayesian networks to model the decision process of humans engaged in micro-management tasks. As part of the evaluation of our work, we repeat their experiment and provide a direct comparison with the work.

Qualitative State Generation

Our system operates on qualitative states, each describing the state of the environment at a discrete time step. Below we describe the qualitative spatial relations which comprise the state. These relations are calculated exhaustively pairwise for all entities in the game, and so for each pair of entities we add one feature per calculi described below to the state representation, indicating the state of the particular relations between those entities. Our general system is capable of representing motion information using the Qualitative Trajectory Calculus (Van de Weghe et al. 2006), as shown in Figure 1, but we omit this for our Starcraft study, and consider only the calculi described below.

Region Connection Calculus

The Region Connection Calculus (RCC) (Randell, Cui, and Cohn 1992) provides a set of binary relations which describe the *degree of connectedness* between spatial regions. In Starcraft, we utilise RCC to represent just two spatial regions on the map – passable and impassable regions. We apply the RCC5 subset of RCC to provide the following relations:

- Equal (**EQ**) - Regions share all points in common.
- Disconnected (**DC**) - No points are shared between regions.
- Overlapping (**O**) - Some points are shared, however there also exist points that are not shared.
- Proper Part (**PP**) - One region entirely encloses the other.
- Proper Part Inverse (**PPi**) - One region is entirely enclosed by the other.

While our use of this calculi is limited to two spatial regions, we will see later that this is important for ensuring the agent makes valid moves during model implementation.

Star Calculus

The Star Calculus (SC) provide binary relations for describing the qualitative direction between points in space with respect to one-another (Renz and Mitra 2004). SC employs angular zoning based on either the parameter m (yielding zones of size $360/m$), or through the specification of an arbitrary sequence of angles. The result is a set of circle sectors emanating from a point, extending into infinity, discretising

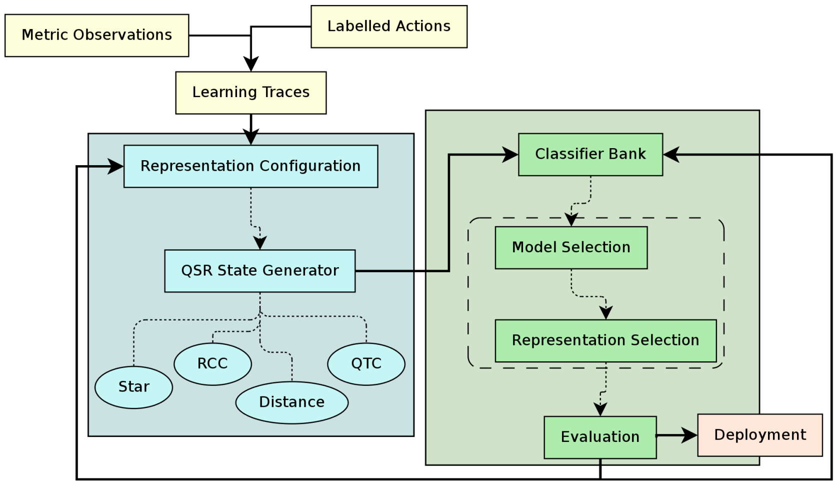

Figure 1: System Overview

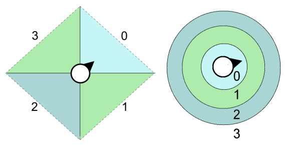

Figure 2: *Left*: A set of four angular zones. *Right*: A homogeneous distance system with four relations.

over a range of angles, with each composing a single binary relation. Between two points the current SC relation is determined by taking the angle between them and determining which discrete zone the result falls in to.

Qualitative Distance

We employ the measure of qualitative distance proposed by (Clementini 1997) whereby the distance between two objects A and B is expressed in terms of the presence of B in one of a set of distinct, non-overlapping distance intervals relative to A. We centre uniformly-sized intervals at A, yielding doughnut-like spatial regions, as illustrated on the right in Figure 2. These relations are calculated and added to the state representation similarly to the Star relations described previously.

Learning

Recall that our learning by observation task is based on Learning Traces incorporating time T, states X and actions Y.

$$LT = [(t_1, x_1, y_1), ...(t_n, x_n, y_n)]$$

The function of our system is to take in such learning traces, and build a model which makes predictions about actions based on observations made about the current state of the world. The following sections describe the format of our state and action representations, before discussing the learning mechanisms we have evaluated and employed in order achieve our goal. We recall that a learning trace is given as a time series, where at each time step the state of the world is given, along with a labelled action. By default, states are described in terms of the metric positions of all observable entities. Our state representation is generated from taking this metric information, and producing a Boolean vector representing the *qualitative spatial relations* between each observable entity given by the calculi described previously.

Action Representation using QSRs

In Starcraft, each agent is capable of selecting from a finite set of discrete actions – attack, move, stop, patrol, as well as special abilities. In a movement command, an agent will move to a *point in space*. The set of possible points an agent can be commanded to move to may be large (4^{10} in the worst case). We would prefer not to represent movement actions in terms of absolute global co-ordinates, since this is brittle, especially when we consider differing map dimensions – some points that exist on 128x128 tile map (with potentially 4^9 possible movement points, in the worst case) may not exist on a 64x64 tile map (4^8 potential points). Describing points in *relative* terms is an improvement, though still movement actions such as $move(0, -10)$ and $move(0, -12)$ may be identical *qualitatively*, perhaps with either being selected by the agent in the same state. This is particularly prevalent with traces of human players – the lack of pixel-perfect accuracy in selecting movement points means that a human player may select points that are qualitatively identical, but are represented quantitatively as many separate and distinct points. To capture this, rather than representing movement points quantitatively, we represent points in terms of the relative *qualitative spatial relations* between the point and the agent.

In Figure 3, we give a trivial example of how this is accomplished. If we consider the red dots as the relative metric positions of points the agent has been commanded to move to, we can see that (under this particular configuration) they can be expressed in terms of their QSRs with the agent – $(star(0), ring(2))$ by the same representation as in Fig. 2.

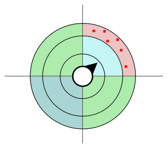

Figure 3: Compression of points.

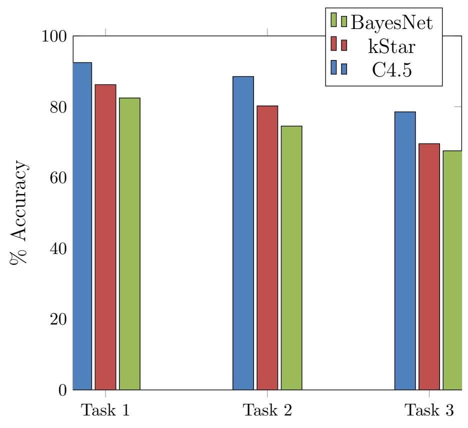

Figure 4: Per-task performance of classifiers, evaluated on data gathered from synthetic agents over 150 episodes per task.

Learning Mechanisms

We approach the problem as one of classification – given some training set of examples of expert behaviour, consisting of *states* and *actions* taken in those states, we wish to build a classifier to predict which action should be taken in a given state. To use the raw metric data that comes from Starcraft presents significant problems for this approach however. We take as an example a single learning trace for the Vulture kiting scenario, consisting of 8607 frames of data, 7072 of which are unique. This metric representation is brittle and exhibits little repetition that can be harnessed by learners to make reliable predictions, and does not generalise well. Another problem faced is that the data inherently features a class imbalance. For instance, the agent selects the *attack* action 4080 times, in comparison to the *move* action which it selects 4528 times, but with 1116 unique parametrisations of that action. While a binary classifier may be constructed to determine whether the agent moves or attacks, building a classifier to select between the 1116 different types of move action is a much more difficult task. We harness the compressive properties of the QSRs described in the previous sections in order ease the difficulty of this problem. After being processed by our system, this particular learning trace contains just 116 unique states and 24 unique move actions. The problem of class imbalance remains, though is eased. To take into account the imbalance between classes in the data, we make use of the Matthews Correlation Coefficient (Matthews 1975) as the performance metric of the classifiers we evaluate.

The learning part of our system is implemented using the Weka data mining software (Hall et al. 2009). In the learning phase, an abstraction layer allows us to plug in any learning algorithm, or selection of learning algorithms we desire. This allows us to evaluate numerous techniques with very little effort. We report in Figure 4 the predictive accuracy of the three best classifiers discovered by our system, on each task, after a ten-fold cross validation procedure. It may be the case that improved performance can be gained through the application of meta-classifiers, feature selection or dimensionality reduction techniques. However, optimising classifier output is not the focus of this work, and as we will see the results we obtain are good enough for our practical purposes of building agents capable of imitation. We found the three best learning algorithms to be the C4.5 Decision Tree Classifier (Quinlan 1993), K-Star – an instance-based learner using an entropic distance measure (Cleary and Trigg 1995), and a Bayesian Network (Neapolitan 1990). In general we found the C4.5 decision tree classifier to be the best of these three, as shown in Figure 4.

Self-Configuration

QSRs may have several parameters associated with them that can be configured – for instance, in the Star calculus the division of angular zones can be varied – this can results in more coarsely or finely grained representations, as desired. Configuring these parameters by hand would be a tedious process, and so our system determines automatically the level of granularity it requires. The performance of learning mechanisms can be improved by hyper-parameter optimisation (Bergstra and Bengio 2012) – searching through the space of parameters that govern a learning mechanism is a standard approach. We include this configuration step in our system, but we also include the parameters that control the granularity of QSRs. Using a grid search, we explore this space of possible representation configurations to find those that provide the best classifier performance, using L_2 regularisation in order to prefer conservative representations (Moseley 1988). The output of our hyper-parameter optimisation step is then a 3-tuple $< R, P, c >$ where R is a vector of values describing a particular configuration of the available QSRs, P is a vector of values describing the configuration of a particular classifier, and c is the classifier itself. This is applied to the state representation, but not the qualitative representation of actions, which is fixed with 8 Star relation and 3 distance relations.

Model Implementation

Recall that a learned model provides a mapping between a state expressed in terms of QSRs and a discrete action, such as attack, patrol, move, and special abilities. Movement actions are given in terms of QSRs, however actions in the game world are given as metric positions, and so we need to translate the QSR specification in the learned model back in to metric positions in the game world in order to command units. The problem is to find metric points that match the QSRs specified in the action representations. Given a move action specified qualitatively, we wish to locate a point in metric space that best matches that description. As discussed previously, the number of points an agent can move to on a map may be very large, and so calculating the QSRs of every point to find those that best match would be expensive, and consider many superfluous points.

We employ a Monte Carlo sampling-based approach (Amar 2006). We randomly generate a set of points around the agent, then calculate the QSRs of these points, and rank them based on their distance from the target QSRs. The notion of distance in QSR space is given by conceptual neighbourhoods which act as transition matrices for QSRs (Gooday and Cohn 1994), an aggregation of the number of transitions between an origin and target set of QSRs can be employed as a distance measure. A threshold k over this distance measure is used to vary how closely the QSRs of candidate points must match our target QSRs, with $k == 0$ being an exact match. We envision in complex application domains there may exist a trade-off of generating more candidate points $vs.$ relaxing the closeness of matches, and so we leave these as parameters that may be tweaked. In our work with Starcraft, we are able to utilise $k == 0$, and recover the set of points X that match exactly our target QSRs. We then define a set of functions for selecting a point $x_i \in X$, these include $closest(X)$, $furthest(X)$, $max\_x(X)$, $max\_y(X)$ among others. For all of our experiments we utilise $closest(X)$ which returns the closest point to the agent that best matches the desired QSRs. Our use of the Region Connection Calculus ensures that points must fall into regions the agent can legally move through.

Experimental Evaluation

We select three benchmark scenarios for our evaluation – a kiting task, a 3vs2 micro-management task, and a final, more difficult task involving a series of combat scenarios of increasing difficulty and opponent make-up. We make these scenarios available alongside the source code of our agents. These scenarios were selected from an online repository of micromanagement training tasks [3] considered challenging enough to justify the creation of training maps for human players to improve their skills. We make no claims as to the optimality of our solutions, the only claim we make is that our agents are capable of reasonable performance on the given tasks. The full source code to these systems is provided[4] so as to allow future work to compare against our results and approaches directly.

Vulture Kiting

In real-time strategy games, kiting is a tactic whereby a unit will quickly attack a pursuing target before immediately retreating, taking advantage of differences in movement speed or attack range to stay out of danger. In the scenario we study – named *vultureDrive* – a single Vulture (a ranged, mechanical unit) is faced with constantly spawning waves of Zerglings (fast-moving melee units) and must kite increasingly large groups while killing as many as possible. Success is based on the ability of the controller to keep the number of Zerglings under control, while continuously moving out of danger, as attacking will cause the unit to momentarily stop, allowing the enemies to gain ground. In our *synthetic agent* we address the problem with an algorithm that, should there be more than one enemy within a specified range of the agent, causes the agent to move to the nearest point on the map that has the fewest enemies close to it – we call this the *move to safety* routine. Otherwise, the agent will attack the nearest enemy. A video of our agent performing this task is available at [5] and the full source code to the agent is available in our GitHub repository.

3vs2 Scenario

In our second task, we repeat the experiment of (Parra and Garrido 2013). In this task, the player is given control of two Dragoons from the Protoss race – large, ranged ground units – and is tasked with defeating a squad of three Hydralisks from the Zerg race – medium sized ranged units. The player is not only at a disadvantage in terms of force sizes, but certain game mechanics make the problem more difficult. In Starcraft, units deal less damage to targets that are smaller than them. The Hydralisks deal 100% damage to the Dragoons, whereas the Dragoons only deal 75% damage to the Hydralisks. While the Dragoons do more base damage per hit than the Hydralisks (20 vs. 10) the Hydralisks attack twice as fast as the Dragoons. Our synthetic agent implements a controller based on the work of (Young et al. 2012) and employs a tactic very similar to the previous Vulture kiting agent. Here, our Dragoon controllers similarly attempt to minimise the number of enemies they are within range of, preferring to flee to areas which have fewer enemy units and more friendly units, attacking only when they are able to overwhelm a target.

The Falls of the Empire

In the final scenario, we consider a more difficult task – a scenario named *The Falls of the Empire*. Initially a player is given access to four Dragoon units, and must progress around a series of gauntlets, defeating groups of enemies from each of the races along the way. The player is occasionally afforded reinforcements, and faces many different types of enemies in this scenario – ranged, melee, spellcasters – and must adjust their tactics accordingly. The map is split into four stages, each featuring combat scenarios of increasing difficulty. We use synthetic controllers similar to those from the previous task, and also employ a high-level reactive planner that moves the agent's units to the next combat scenario after each one is completed. Once there, the usual behaviour controllers take over.

Human Controllers

To study our ability to reproduce observed human behaviour, we gathered five human subjects who possessed minor prior familiarity with the Starcraft domain, in terms of the general concepts and mechanisms of the game, and with some hands-on experience. No information on the tasks was given beyond the instructions included with the training scenarios. For each subject we gathered 30 episodes of data per task,

[3] http://wiki.teamliquid.net/starcraft/Micro_Training_Maps

[4] https://github.com/jayyoung/BenchmarkStarcraftAgents

[5] https://vimeo.com/103123818

System	Avg. Score
Benchmark Agents	**20 ± 3.14**
Learned (From Benchmark)	17 ± 3.88
Human Controller	16 ± 4.43
Learned (From Humans)	11 ± 4.51

Figure 5: System Results for Vulture Kiting Task, given in terms of average number of kills.

System	Avg. Win Rate
Benchmark Agents	**96**%
Learned (From Benchmark)	92%
Human Controller	67%
Learned (From Humans)	58%
Parra and Garrido	44%
Built-in AI	0%

Figure 6: System Results for 3vs2 task, given in terms of average win rate.

for a total of 150 episodes of data per task. In our training of the synthetic agents we also provided 150 episodes of data per task.

Discussion

In each experiment we provided the learning agents with full data sets (150 episodes per task), and evaluated them for 100 episodes. Figure 5 shows our results for the Vulture kiting scenario, giving the average score of each system in terms of number of kills, which is the performance metric provided by the scenario. There is a slightly larger drop in performance when the system is trained on the human data than synthetic (16 to 11 vs. 20 to 17). This deficit can be explained by the fact that the human controllers were users who had previously limited experience with Starcraft prior to the study. As such, while the synthetic agents achieved a roughly consistent level of performance across all episodes, some human data included initial episodes where the human controllers got to grips with the task, and performed relatively poorly during those episodes. This allowed mistakes and errors introduced by the expert to propagate to the learning agent. While we can easily identify where this learning curve starts and finishes, simply discarding the data may not be the best option – we suspect it could be harnessed to provide an agent with a more thorough understanding of the task domain, by observing directly the process of an expert attempting to accomplish some goal and improving its performance iteratively, rather than having to engage in that process itself. Though this, we may be able to discover reward functions that can be used for later improvement. For now, we leave further exploration of this for future study. In the 3vs2 scenario, Parra and Garrido observed that the default, built-in AI of Starcraft is unable to achieve victory at all, whereas their Bayesian Network-based system trained on the observation of human micromanagement achieved victory in 44% of cases. These results, along with our own, are displayed in Figure 6. The results show that our benchmark agents perform well on the task, as do agents that have learned to imitate them. Human controllers however achieve a much lower level of success, as do the imitating agents. Parra and Garrido do not report the baseline level of performance of their human subjects, only the imitating system, so we cannot make a direct comparison. The *Falls of the Empire* task proved very difficult for the human players, with very few progressing past the first stage of the map, which features five separate combat encounters of increasing difficulty, against 24 total enemies. The benchmark agents typically progressed in to the second stage of the map, but rarely further than this.

System	Avg. Score
Benchmark Agents	**42 ± 10.33**
Learned (From Benchmark)	32 ± 6.73
Human Controller	19 ± 8.11
Learned (From Humans)	16 ± ± 6.24

Figure 7: System Results for The Falls of the Empire, given in terms of average number of kills.

Our results show that our approach is successful in learning low-level tactical skills, and achieving a level of performance approaching that of the original expert controller. However we have considered a limited set of static benchmark scenarios – one major dynamic of the Real-Time Strategy domain is that it is typically *open world*, and the combat scenarios a player encounters will vary widely during full-game play. It may be the case that the C4.5 classifier we utilised does not scale to this scenario, due to the poor flexibility of decision trees in this regard. We also see that there is a clear coupling between the performance of an expert and the agent which imitates it, including the replication of mistakes. Agents need not always repeat the errors of their teachers, however. For instance, learned models may be used in the future as a starting point for reinforcement learning approaches (such as Q-learning) in order to cut out the need to engage in (often expensive and costly) state-space exploration, meaning agents could improve their models to levels of performance beyond what they have observed.

Conclusion

We presented an approach to Learning by Observation in the real-time strategy domain, evaluated in the game of Starcraft. Our system employed a qualitative-relational compression of spatial information in order to ease the difficulty of the learning task, before using a Monte-Carlo method to aid in decompressing the data in real-time. This allowed our system to control an agent to act in the chosen tasks, imitating the performance of observed experts, both human and synthetic. The synthetic controllers were of our own design, for which we make full source-code available to allow for future work in this area to build on our own. *The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No 600623 and the EPSRC grant EP/K014293/1.*

References

Amar, J. 2006. The Monte Carlo method in science and engineering. *Computing in Science & Engineering* 8(2).

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57(5):469–483.

Behera, A.; Cohn, A.; and Hogg, D. 2012. Workflow activity monitoring using dynamics of pair-wise qualitative spatial relations. *Advances in Multimedia Modeling*.

Bergstra, J., and Bengio, Y. 2012. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research* 13:281–305.

Cleary, J., and Trigg, L. 1995. Kˆ*: An Instance-based Learner Using an Entropic Distance Measure. *ICML* 5:1–14.

Clementini, E. 1997. Qualitative representation of positional information. *Artificial Intelligence* 95:317–356.

Floyd, M.; Esfandiari, B.; and Lam, K. 2008. A case-based reasoning approach to imitating RoboCup players. *21st International Florida Artificial Intelligence Research Society Conference*.

Frommberger, L. 2008. Learning To Behave in Space: a Qualitative Spatial Representation for Robot Navigation With Reinforcement Learning. *International Journal on Artificial Intelligence Tools* 17(03):465–482.

Gooday, J., and Cohn, A. 1994. Conceptual neighbourhoods in temporal and spatial reasoning. *Spatial and Temporal Reasoning, ECAI*.

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The WEKA data mining software: an update. *SIGKDD Explorations* 11(1):10–18.

Jung, T., and Polani, D. 2012. Learning RoboCup-Keepaway with Kernels. *JMLR: Workshop and Conference Proceedings* 1:33–57.

Matthews, B. W. 1975. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et biophysica acta* 405(2):442–451.

Molineaux, M.; Aha, D. W.; and Sukthankar, G. 2008. Beating the Defense : Using Plan Recognition to Inform Learning Agents.

Montana, J., and Gonzalez, A. 2011. Towards a unified framework for learning from observation. *IJCAI Workshop on Agents Learning Interactively from Human Teachers*.

Moseley, L. 1988. Introduction to machine learning.

Neapolitan, R. E. 1990. *Probabilistic reasoning in expert systems: theory and algorithms*. John Wiley & Sons, Inc.

Parra, R., and Garrido, L. 2013. Bayesian networks for micromanagement decision imitation in the RTS game starcraft. *Advances in Computational Intelligence*.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*, volume 1.

Randell, D.; Cui, Z.; and Cohn, A. 1992. A spatial logic based on regions and connection. *3rd International Conference on Knowledge Representation and Reasoning*.

Renz, J., and Mitra, D. 2004. Qualitative direction calculi with arbitrary granularity. *Lecture notes in computer science*.

Ros, R.; Llu, J.; and Mantaras, R. L. D. A Case-Based Approach for Coordinated Action Selection in Robot Soccer. (February 2009):1–50.

Shimada, K.; Takahashi, Y.; and Asada, M. 2010. Efficient Behavior Learning by Utilizing Estimated State Value of Self and Teammates. *Robot Soccer World Cup XIII* 1–11.

Sridhar, M., and Cohn, A. 2010. Unsupervised learning of event classes from video. *AAAI* 1631–1638.

Van de Weghe, N.; Cohn, A.; De Tre, G.; and De Maeyer, P. 2006. A qualitative trajectory calculus as a basis for representing moving objects in geographical information systems. *Control and Cybernetics* 35(1):97.

Vieira, D. C. D. L.; Adeodato, P. J. L.; and Gon, P. M. 2010. Improving Reinforcement Learning Algorithms by the Use of Data Mining Techniques for Feature and Action Selection. *IEEE International Conference on Systems Man and Cybernetics* 1863–1870.

Young, J.; Smith, F.; Atkinson, C.; Poyner, K.; and Chothia, T. 2012. SCAIL: An integrated Starcraft AI system. *IEEE Computational Intelligence and Games*.