Evolutionary Learning of Goal Priorities in a Real-Time Strategy Game

Jay Young and Nick Hawes

Abstract

We present a drive-based agent capable of playing the real-time strategy computer game Starcraft. Success at this task requires the ability to engage in autonomous, goal-directed behaviour, as well as techniques to manage the problem of potential goal conflicts. To address this, we show how a case-injected genetic algorithm can be used to learn goal priority profiles for use in goal management. This is achieved by learning how goals might be re-prioritised under certain operating conditions, and how priority profiles can be used to dynamically guide high-level strategies. Our dynamic system shows greatly improved results over a version equipped with static knowledge, and a version that only partially exploits the space of learned strategies. However, our work raises questions about how a system must know about its own design in order to best exploit its own competences.

Introduction

Autonomous AI systems should be aware of their own goals and be capable of independently formulating behaviour to address them. We would ideally like to provide an agent with a collection of competences that allow it to act in novel situations that may not be predictable at design-time. In particular, we are interested in the operation of AI systems in complex, oversubscribed domains where there may exist a variety of ways to address high-level goals by composing behaviours to achieve a set of sub-goals taken from a larger set. Our research focusses how such sub-goals might be chosen (i.e. where dispositions towards certain goals come from in the first place, and the conditions under which they might change), and also techniques for goal management in cases of conflicts between goals. This paper describes an integrated AI system capable of operating in a complex, oversubscribed domain: the real-time strategy game Starcraft [1]. We show how an integrated system utilising a drive-based architecture, data mining, learning, and novel strategy selection techniques can be used in an AI agent capable of playing a full game.

Background

A motivational framework (Hawes 2011) employs a system of drives, each of which generates goals to satisfy the desires of a system, plus processes to select and manage goals. These processes work in concert as mechanisms to direct system attention and produce goal-directed behaviour. Coddington and Luck (2003) make the case that motivational frameworks act as bridges between high-level, abstract planning systems and the environment in which an integrated system is situated, providing a sense of context to planning that would not otherwise be available. This affords the ability to include a variety of additional semantic information in the planning process, as well as providing a framework for dynamic goal generation.

Such frameworks are often studied as components of integrated intelligent systems (Nourbakhsh et al. 1999; Stoytche and Arkin: 2004; Coddington 2007; Molineaux, Klenk, and Aha 2010) and artificial life systems (Grand, Cliff, and Malhotra 1997; Scheutz 2004). However, due to the small numbers of goals present in existing systems, goal management is a relatively simple affair. Hanheide et al. (2010) describe a system similar in architecture to our own that manages just two goals, whereas the one discussed in this paper must manage upwards of forty. As the number of goals increases, the potential for goal conflict grows. This leads to a requirement for more sophisticated management processes, such as dynamic goal re-prioritisation, allowing agents to alter their behaviour to meet changing operational requirements. In the oversubscribed problem domains we are interested in, encoding all possible operating strategies at design time may be infeasible. Especially in our example case of the real-time strategy game Starcraft, which is continuous, asymmetric, and features partial observability. Such domains make good test-beds for research into AI systems (Buro and Furtak 2003).

Our system plays the Terran race in Starcraft. This is one of three possible factions, each with their own strengths and weaknesses. The main goals of the game centre around gathering resources, producing military units and using them to destroy an opponent's buildings. The path to achieving these high-level goals is a matter of player strategy. At the start of a game, the number of possible actions is constrained; a player starts off with a base, a handful of worker units, and enough resources to train one more worker unit. Thus, one sensible course of action is to utilise the existing workers to gather more resources. The starting resources can be used to train a new worker or saved to spend later on something else. A player might then choose to expend resources on con-

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

structing new buildings, such as a barracks to train marines, a factory to train mechanised units, or on defensive structures to ward off attacks. Or by adding more bases, launching attacks on their opponent, upgrading existing units, or perhaps do all of these simultaneously. There exists a continual expansion in the number of, potentially conflicting, choices that a player must make, and choices made early-on may not fully show their effects until later in the game.

Our System

Our approach uses a drive-based architecture featuring multiple, asynchronous goal-generating processes, and a goal management system. Goal-generating processes encode high-level drives, as well as a payload of sub-goals that can satisfy a particular drive. Goals may vary from abstract, top-level goals down to more operation-specific sub-goals. Building on the motive processing work of Sloman et al. (1993) and Hanheide et al. (2010) we define that a goal generator is made up of the following basic components:

- A *motivating condition* describing a state of affairs, or desire, that the system wishes to be in place.
- A *measure of agitation* representing the degree to which a particular motivating condition may be violated, along with a threshold on this value.
- A *goal*, or *set of goals*, designed to produce behaviour to satisfy the motivating condition, each tagged with heuristic information detailing the kinds of resource commitments required in order to pursue a particular goal, and including a measure of priority.

The role of a goal generator is to process its inputs to produce a measure of agitation, and to generate goals once a threshold has been met. Generated goals are then passed on to management mechanisms, and then on to a planning layer if scheduled to be pursued. A system can have multiple, asynchronous goal generators. In some cases it may be that multiple goals can be generated and pursued concurrently. If we also consider mutually exclusive goals then a system requires a mechanism to manage which goals can be pursued at any one time, and which should be scheduled for later pursuance, or even abandoned. Such management processes may rely on various forms of meta-information. The management problem could then be treated as a search to find a set of goals that best satisfies the needs of the system at a particular point in time, or according to some overarching plan or strategy. We also consider *priority information* attached to goals. This information can be exploited to determine which goals should be preferred when conflicts arise. To map goals to priority values we introduce the concept of a *priority profile* (PP). As management processes attempt to arbitrate between conflicting goals, differing PPs result in different arbitration outcomes. The question remains as to where such mappings might originate, but for now we assume a PP will be supplied at design time. Later we shall discuss how these configurations can (and *should*) be learned by a system.

In Starcraft the key element of a high-level strategy is a build order. This describes how to order a sequence of building constructions to gain access to particular units and upgrades. Much like in Chess, players utilise starting strategies

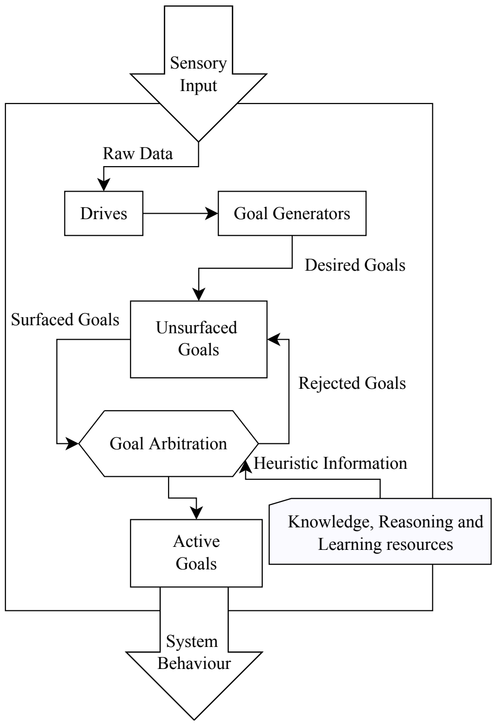

Figure 1: High-level system diagram

that dictate the first few moves that may lay the foundation for a single strategy, or a wide range of potentially branching strategies. Trade-offs exist in our domain, since focusing on a particular avenue of play necessitates making sacrifices in other areas, such as expending resources on unit upgrades, base expansions or defensive structures. These factors lend themselves to our idea of explicit prioritisation of goals.

Our system's goal generators primarily encode build order. They include motivators for each in-game unit and building that can be constructed. The system of motivators behaves in a greedy fashion, always attempting to generate goals to build and produce as much as possible. The system must then perform arbitration on this set of goals to produce *achievable* sets of goals in line with high-level strategy.

As mentioned previously, early in a game, once a steady flow of resources has been secured, the system must decide which buildings to construct in order to begin the production of an attacking force. Each building is capable of producing several different types of unit, such as military units, support units, or unit upgrades and special abilities. There may also exist synergy between different unit types – such as the ability of medics to heal biological units. A typical strategy might prioritise the production of marines, medics and tanks, necessitating the construction of barracks and factories. As these production goals surface they must go through a process of arbitration before being sent on to the active set.

When compiling the active set, the system assumes that all surfaced goals can be pursued concurrently, unless an analysis of resource costs during arbitration determines that this is not possible. If this happens it commits resources to goals with higher priorities. The assumption of concurrent action means the system is capable of distributing attention and pursuing many goals at once, such as launching attacks, managing resources, base expansions, production etc. to a higher degree than human players, who must carefully ration attentional focus.

Active goals are flagged as being pursued before being sent to a reactive planning layer. This layer produces a plan composed of a set of low-level commands, which it then executes. Executing plans are continually monitored for errors, and for completion. Should a plan encounter an error during execution it may be abandoned entirely, retried from the start, or a recovery attempt may take place by making use of sub-routines to repair or retry specific parts of the plan.

Inspired by the work of Lewis et al. (2011) and Hsieh and Sun (2008) the system design also exploits information gathered from data mining and analysis of a corpus of event logs from games played by expert players in professional league matches. Specifically we employed a time-series analysis of game-time and army size, which indicates that early in a game a player tends to prefer a small force to attempt to harass and hamstring an opponent, whereas during the mid-game an attacking force will typically be much larger. Later in a game, players tend to revert to a medium-sized force, typically of more powerful, expensive units. This allows us to encode in the system a goal generator equipped with a drive function relating game-time to army size. Our AI then attempts to launch attacks once enough units have been produced to satisfy the drive activation condition at a particular point in time. The unit make-up of forces can be specified at design-time, or can come about as a result of the reprioritisation of unit production goals. We later show how the system learns to exploit this aspect of its design, and employs novel army configurations.

Static Profiles

Our initial implementation was configured with a priority profile specified by a human designer. Due to the complexity of managing 40 goal representations, designing a PP is a complex task. This is exacerbated by the need to consider the priority of a particular goal in the context of the wider strategy, meaning that a small change to one goal priority may have wide-ranging implications. The context-dependent nature of PPs adds an extra layer of complexity to the configuration process.

Our hand-configured system achieved victories in 53% of 10,000 games against Starcraft's built in AI. A competent human player, or advanced AI, should expect to achieve a win rate of close to 100%. We also compared our system against the EISBot system of (Weber, Mateas, and Jhala 2010), repeating their evaluation of 300 games against Starcraft's built in AI. Our static system achieves a win rate of 54%, compared to EISBot's 73% in the same experiment. We also entered an early prototype of this system into the 2011 AIIDE Starcraft AI competition[2], where it performed relatively poorly against other AIs. We shall discuss the performance of this system further, but we mention these results now to serve as a motivating factor for the next phase of work.

Conceptually an AI player capable of learning and adapting its strategy during the game should perform better than those that rely on hard-coded strategies – the question that motivates our following work is then one of how we might systematically remove pieces of human-encoded knowledge in our system, such as hand-configured PPs, and replace them with systems to facilitate more dynamic behaviour.

Learning of Priority Profiles

We would like our system to be able to formulate new high-level strategies, using past experiences to improve future performance. To address this we use Genetic Algorithms (GAs) as a learning mechanism (Janikow 1993; Louis and McDonnell 2004), specifically the GA of Chop (2005) (the "Chopper GA") which employs heuristic measures to modulate population size.

A PP naturally provides a compact representation of high-level strategic concerns upon which a GA can act. We score the fitness of a particular PP using Starcraft's internal scoring mechanism. This is a multi-dimensional representation of performance, covering such elements as the amount of resources gathered, the number of enemy units killed, and the number of buildings constructed. In professional games where there is a hard time-limit, this score is often used as a tie-breaker. We also apply our own weightings to the score in order to bias in favour of profiles that typically result in victories in the game, as it is possible to achieve a high score and yet still ultimately be defeated. As such, when compiling a score, we weight the component relating to destroyed enemy units 30% higher than other components, and apply the same overall bonus to scores that resulted in a victory.

Using these measures we instantiate an initial population of strategies by applying a Gaussian mutation operator to a set of five hand-configured profiles – a technique based on case-injection (Louis and McDonnell 2004; Johnson and Louis 2004). These five profiles encode basic approaches to prioritising each goal, and are configured primarily through (human) trial and error. Each PP in the population is used to play games against Starcraft's built-in AI players on each map in our training set of seven maps taken from the 2011 AIIDE Starcraft AI competition map pool. At the end of a game the weighted score is recorded, which we take as the fitness of a particular individual against a particular opponent. The duration of a single game of Starcraft, played at normal speeds, can potentially be upwards of an hour. This presents an obstacle for any kind of machine learning that requires the evaluation of a large number of games. We evaluate 1000 generations of our GA, equating to our system playing just over 250,000 games of Starcraft, which took around three weeks to complete at the maximum in-game speed. The key to this was the use of a distributed network of computers, each equipped with several virtual machines playing instances of the game to evaluate members of the current GA population. This allowed us to play many games in parallel, evaluating between 40 and 50 complete populations per day.

[2]http://skatgame.net/mburo/sc2011/

As games are played we record a set of observations made during the course of play, such as the size of the map, the time from the start of the game to the opponent's first attack, the first military units observed being employed by the opponent, and similar strategic concerns. For now such observations are chosen based on human knowledge of the game, as we leave open the question as to what information can be used to most effectively predict an opponent's strategy for future work. These observations allow us to make comparisons between the performance of particular PPs under given conditions. That is, we can say that utilising PP *x* has historically resulted in a higher score than employing PP *y* given that a set of strategic observations *z* has been made about the current game state that match more closely those attached to *x* than *y*.

Dynamic Profiles

Humans rarely stick to a single strategy throughout the course of a game. Instead, players react to the choices made by an opponent and the various situations that may arise during play, drawing on past knowledge to make good decisions. We extend our motivation system to do this by including the novel capability to dynamically reconfigure its PP, allowing for the modulation of high-level strategy.

Our training procedure provides a database of PPs and associated performance scores. The semantic information provided by the set of observations attached to each allows our AI to reason about the set of strategies that are available to it. In order to switch strategies, observations must first be made in real-time and matched to PPs in the database, producing a set of candidate strategies and their own observation sets. This set of candidates may initially be large, and there may not exist a direct mapping between observations made so far and those of any of the candidates. In addition, some candidate PPs may represent a change in strategy, requiring many modifications to its own current course of action. This may repeat as each new observation is made, leading to *thrashing* – a rapid switching between many radically disparate strategies, producing behaviour that is incoherent.

To address these issues, we rank each candidate strategy by calculating a heuristic measure of predicted improvement. First, we take into account the expected improvement between PPs by calculating the absolute score disparity between the current PP and the candidate. We then calculate the strategic distance between them as the sum of Hamming distances between the current and candidate PPs. This represents the degree of strategic change required to move between them. Finally, we consider the information disparity between profiles by comparing differences between observation sets. We weight each of these factors with individual coefficients that allow us to control the attitude the system has towards switching between PPs (e.g. by valuing switches that require minimal strategic changes, or controlling the measure of risk involved in switching to strategies where information disparity may be high). We prefer to bias towards strategically similar PPs, meaning that our system ranks candidate strategies that are genetically close to the one currently being followed more highly, along with a small information disparity (if one exists at all) and thus minimal risk. With this we approach the problem of PP reconfiguration as one of path-finding through a *strategy space*. We use the predicted improvement measure to provide a trajectory through this space using a best-first approach. Strategy navigation in this way is simplistic, and more sophisticated methods may provide better performance, such as A* path finding.

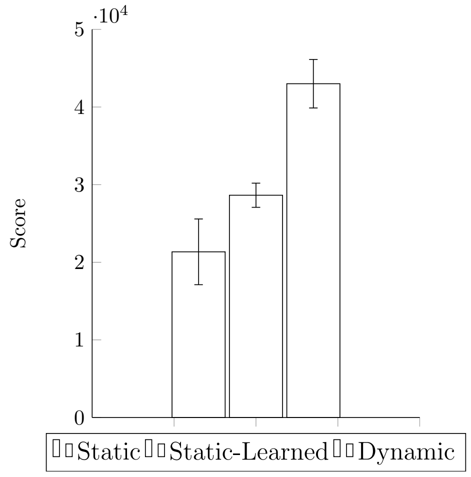

Figure 2: Comparison of system performances in terms of in-game score. Averaged results over 10,000 games.

Results

Quantitative Analysis

Figure 2 shows a comparison between three different versions of our system on in-game scores recorded across 10,000 games, with each game played on a random map from our acceptance set (3 maps from the aforementioned map pool not used for training) against a random AI opponent. The Static system uses a single, non-changing PP encoded by a human designer. The Static-Learned system uses a single, static PP chosen from the database of learned PPs based on the observations that can be made at the start of a game (the size of a map, the faction chosen by an opponent, starting position etc.). The Dynamic system exploits the full range of PP reconfiguration mechanisms discussed above. The performance of each system is stable, such that each system exhibits a consistent level of performance, within a certain error range as shown. We see that the Dynamic system out-performs the other systems in terms of in-game score, indicating that it is more successful along the metrics from which the score is calculated – resource gathering, building and unit production, destruction of enemy structures etc.

We also compare the win/loss ratios of the three systems, since scores by themselves do not represent whether a player has won or lost a game (a long, action-packed game would allow a player to earn a high score, but may still result in an eventual loss for that player). Table 1 shows that, on average, the system capable of strategy-switching out-performs alternative systems in terms of games in which it is victori-

	Static	Static-Learned	Dynamic
Win Rate	53%	59%	68%

Table 1: Comparison of wins between system types.

	EISBot	Static	Static-Learned	Dynamic
Win Rate	73%	54%	58%	66%

Table 2: Comparison of victory rates between our system and EISBot.

ous. In terms of the relationship between increased in-game scores and game victories, in this experiment, an increase in average game score of 109% appears to correlate with an increase in win rate by roughly 28% when comparing our strategy-switching system to our basic system using human encoded knowledge. Starcraft is a game that requires mastery of both macro-level strategy and micro-level tactics to succeed. Our approach focuses largely on macro-level strategy, so to see such improvements while mainly focusing on the macro side of the strategic space is encouraging.

Comparison against existing work

We compare our system against EISBot (Weber, Mateas, and Jhala 2010), a state-of-the-art Starcraft system which takes a similarly goal-directed approach to gameplay, though using more statistical and case-based approaches. They also evaluate their AI against the in-game AI players of Starcraft across a selection of maps from the 2010 AIIDE Starcraft AI tournament [3]. This entails playing twenty games against each of the three AI opponents on each of five maps, totalling three hundred games. We repeat their experiment with our systems, allowing us to provide a direct comparison between our work and theirs. Table 2 shows our results in comparison. We see that our dynamic system does not perform as well as EISBot, though it does still perform better than the other versions of itself. The disparity of performance with EISbot may be partially explained by bottlenecks in our system pertaining to unit micro-management, something we have briefly touched upon and shall re-visit in our later discussion.

Qualitative Analysis

We have seen that our system shows increased performance when harnessing a database of learned strategies, but we are also interested in *why* such strategies are effective and result in better performance. Learning algorithms based on case initialisation techniques run the risk of creating endless generations of minor permutations of the same base solutions. To avoid this it is necessary for the algorithm to escape the initialisation cases, while maintaining the sense of strategic viability and success that they provide. We believe this to be the case, as our system exhibits behaviour not encoded at design time.

Perhaps the most interesting learnt strategy we observed is one which centres around the use of mechanised Goliath units and Medic units. Medic units are unable to attack directly, but are able to heal friendly biological units (such as other medics). Goliath units are large, ranged units capable of attacking air and land targets. A strategy learned by the system prioritises goals associated with the production and upgrading of these unit types quickly. As the AI moves armies of units around the map, it attempts to do so such that all of the comprising units stay within a certain range of each other, maintaining a rough formation. This is part of the system that is hard-coded at design time. In this case, the smaller medic units surround the larger Goliath units, forming a human shield, protecting the Goliaths with a wall of medics who, though unable to attack by themselves, heal each other as they take damage from enemy melee units attempting to break through. At the same time, this enables the Goliaths to deliver ranged attacks from relative safety. This is a good example of how the system learns to take advantage of not only game mechanics, but its own design.

Discussion

Although our system was capable of learning successful strategies, training proved expensive in terms of time and computational power. One reason for this is that our learning algorithm was unable to exploit the complex relationships between goals in our representation of high-level game strategy. Our system falls prey to a phenomenon known in biology as Fitness Epistasis (Chan, Aydin, and Fogarty 2003; Reeves and Wright 1995), meaning in our case that our system behaviour comes as a result of potentially complex interactions between its various goals as they are pursued. For instance, there exists a relationship between the goal to create buildings that produce a particular unit, and the goals to produce the particular units that come from that building. These relationships are not explicitly represented in our learning system, and cannot be specifically and directly exploited or reasoned about. This problem is not as evident in simpler systems implementing motivation frameworks, due to the smaller number of goals and ease of arbitration, but as the quantity of goals increases this problem becomes more evident. A main question raised by this work then is how learning in the way we have described might be aided by semantic information that can be used to describe and exploit goal relationships, and how those relationships should be represented. Systems armed with relatively large goal sets perhaps must be equipped with mechanisms to analyse *why* a particular solution might perform poorly, thus allowing for the focus of learning resources on improving particular areas of a strategy, while holding other parts of a strategy static.

We also assumed our AI to be competent at every strategy it can produce through the stochastic learning process, but we see that is not the case due to the prior competences encoded in the system – specifically micro-management tactics. It is unsurprising then that it evolves strategies that

[3]http://eis.ucsc.edu/StarCraftAICompetition/

best exploit these prior competences. To be more successful, the AI must not only learn high-level strategy, but also micro-level unit control tactics (Shantia, Begue, and Wiering 2011). Due to the complexity of the problem, and extensive solution space, this is significantly more expensive. In more general terms, the expected benefit from completing a task (i.e. implementing a certain strategy in our case) might also require some method of describing how well an agent's design or capabilities might be suited to perform that task.

Our general comment on these matters is this – successful learning of high-level strategy must be informed and complemented by introspective knowledge of a system's own architecture and competences. The ways this might be accomplished remain an open question, but such internal representations should be given the same degree of importance and careful treatment as representations of operating domain dynamics.

Conclusion

We presented a motivational architecture that supports the ability to filter and manage a set of goals in order to support autonomy in an AI system. We then showed how learning can be utilised to solve the specific problem of prioritisation between individual goals. Applying these techniques to an AI capable of playing the real-time strategy game Starcraft allowed us to produce a database of learned strategies, allowing the system to out-perform a separate version reliant on hard-coded knowledge. Our work reveals that a more detailed and careful treatment of what knowledge a system possesses – both in terms of relationships between its goals, as well as its own prior competences – is key to taking our approach forward. In short, we believe that a system must know about how it functions itself in order to best learn about how its environment functions.

References

Buro, M., and Furtak, T. 2003. Rts games as test-bed for real-time ai research. *Proceedings of the 7th joint conference on information sciences*.

Chan, K.; Aydin, M.; and Fogarty, T. 2003. An epistasis measure based on the analysis of variance for the real-coded representation in genetic algorithms. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, volume 1, 297 – 304 Vol.1.

Chop, N. E., and Calvert, D. 2005. The chopper genetic algorithm a variable population ga. In *Proceedings of Artificial Neuronal Networks in Engineering 2005*.

Coddington, A., and Luck, M. 2003. A motivation-based planning and execution framework. *International Journal on Artificial Intelligence Tools.* 13:5–20.

Coddington, A. 2007. Integrating motivations with planning. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, AAMAS '07, 127:1–127:3. ACM.

Grand, S.; Cliff, D.; and Malhotra, A. 1997. Creatures: artificial life autonomous software agents for home entertainment. In *Proceedings of the first international conference on Autonomous agents*, AGENTS '97, 22–29. New York, NY, USA: ACM.

Hanheide, M.; Hawes, N.; Wyatt, J.; Gobelbecker, M.; Brenner, M.; Sjoo, K.; Aydemir, A.; Jensfelt, P.; Zender, H.; and Kruijff, G.-J. M. 2010. A framework for goal generation and management. In *In Proceedings of the AAAI Workshop on Goal-Directed Autonomy*.

Hawes, N. 2011. A survey of motivation frameworks for intelligent systems. *Artificial Intelligence* 175:1020–1036.

Hsieh, J.-L., and Sun, C.-T. 2008. Building a player strategy model by analyzing replays of real-time strategy games. In *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, 3106 –3111.

Janikow, C. Z. 1993. A knowledge-intensive genetic algorithm for supervised learning. *Mach. Learn.* 13:189–228.

Johnson, J., and Louis, S. 2004. Case-initialized genetic algorithms for knowledge extraction and incorporation. *In: Knowledge incorporation in evolutionary computation* 57–80.

Lewis, J. M.; Trinh, P.; and Kirsh, D. 2011. A corpus analysis of strategy video game play in starcraft: Brood war. In *Proceedings of the 33rd Annual Conference of the Cognitive Science Society*, 687–692.

Louis, S., and McDonnell, J. 2004. Learning with case-injected genetic algorithms. *Evolutionary Computation, IEEE Transactions on* 8(4):316 – 328.

Molineaux, M.; Klenk, M.; and Aha, D. 2010. Goal-driven autonomy in a navy strategy simulation. In *AAAI '10*.

Nourbakhsh, I. R.; Bobenage, J.; Grange, S.; Lutz, R.; Meyer, R.; and Soto, A. 1999. An affective mobile robot educator with a full-time job. *Artificial Intelligence* 114:95–124.

Reeves, C. R., and Wright, C. C. 1995. Epistasis in genetic algorithms: An experimental design perspective. In *Proceedings of the 6th International Conference on Genetic Algorithms*, 217–224. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Scheutz, M. 2004. Useful roles of emotions in artificial agents: A case study from artificial life. In *AAAI'04*, 42–48.

Shantia, A.; Begue, E.; and Wiering, M. 2011. Connectionist reinforcement learning for intelligent unit micro management in starcraft. In *International Joint Conference on Neural Networks*.

Sloman, A.; Hogg, D.; Humphreys, G.; Partridge, D.; Ramsay, A.; and Beaudoin, L. 1993. A study of motive processing and attention. In *Prospects for Artificial Intelligence*, 229–238. IOS Press.

Stoytche, A., and Arkin:, R. C. 2004. Incorporating motivation in a hybrid robot architecture. In *Journal Of Advanced Computational Intelligence and Intelligent Informatics*, volume 8, 269–274.

Weber, B.; Mateas, M.; and Jhala, A. 2010. Applying goal-driven autonomy to starcraft. In *Artificial Intelligence and Interactive Digital Entertainment*.