A Review of Computational Intelligence for StarCraft AI

Zhentao Tang^{†‡}, Kun Shao^{†‡}, Yuanheng Zhu^{†‡}, Dong Li^{†‡}, Dongbin Zhao^{†‡}, Tingwen Huang[§]

[†]State Key Laboratory of Management and Control for Complex Systems,

Institute of Automation, Chinese Academy of Sciences, Beijing, China, 100190

[‡]University of Chinese Academy of Sciences, Beijing, China, 100049

[§]Texas A&M University at Qatar, Doha, Qatar, POBOX 23874

Email: {tangzhentao2016, shaokun2014, yuanheng.zhu, lidong2014, dongbin.zhao}@ia.ac.cn, tingwen.huang@qatar.tamu.edu

Abstract—After artificial intelligent (AI) scientists have conquered Go game, StarCraft has been the next biggest challenge. A highly intelligent AI system that is able to beat human professional players is expected. In this paper, we review the recent development of computational intelligence (CI) in the field of StarCraft AI. Successful applications of CI techniques are analyzed and compared from different levels of AI functionality. It should be noted that current StarCraft AI is highly dependent on human experience and is far away from a completely intelligent one. New frameworks and techniques are still expected to improve the intelligence.

I. INTRODUCTION

In the history of computer games, StarCraft is probably the most famous one. Screen shots are presented in Fig. 1. It was first released by Blizzard Entertainment in 1998, and had been sold more than 9.5 million sets in the following ten years. StarCraft belongs to the category of real-time strategy (RTS) games, in which players need to operate according to real-time game states in order to beat opponents. The game has three races, Terran, Protoss, and Zerg, and each of them has their own advantages. Due to its popularity, StarCraft has also attracted intensive interest from researchers of artificial intelligence. Several competitions have been held every year to stimulate the development of StarCraft AI, including IEEE CIG, AIIDE, and SSCAIT StarCraft AI competitions [1].

At present, most StarCraft AI systems are developed based on Brood War Application Programming Interface (BWAPI)¹, a free and open source C++ framework that is used to interact with StarCraft: Brood War. With BWAPI, students and researchers have the access to create AI bots to play the game. For the game AI research, BWAPI also provides an ideal environment to study the control of multiple units with different difficulty levels. In addition to BWAPI, TorchCraft [2] is presented to promote the machine learning study in this area. TorchCraft is a library that enables state-of-the-art machine learning research on raw game data by interfacing Torch with StarCraft: BroodWar. More recently, DeepMind and Blizzard jointly introduce StarCraft II Learning Environment (SC2LE)

This work is supported by National Natural Science Foundation of China (NSFC) under Grants No.61573353, No.61603382 and No. 61533017, and National Priority Research Project NPRP 9166-1-031, funded by Qatar National Research Fund, Qatar. $^{\rm l}$ http://bwapi.github.io/



Fig. 1. Game screen shots of StarCraft I and StarCraft II.

[3] and its Python component PySC2². This is a reinforcement learning environment based on the StarCraft II game. Compared with learning platforms in StarCraft I, SC2LE poses a new grand challenge for reinforcement learning, representing a more challenging class of problems than considered in most prior work. SC2LE considers the observation, action, and reward specification for the StarCraft II domain. Besides, a suite of mini-games are also provided to focus on different elements of StarCraft II gameplay. In some respects, SC2LE offers a new and challenging environment for exploring deep reinforcement learning algorithms and architectures.

Generally speaking, to design a complete StarCraft AI, one has to solve a series of challenges, such as spatial and temporal reasoning, opponent modeling, adversarial planning and multiagent collaboration. These tasks can be generalized into three levels: strategy, tactics, and reactive control. Currently, most StarCraft AI bots are designed based on human experience, and bots can perform predesigned operations when certain conditions are satisfied. Unfortunately, such approach makes the system stubborn. Their intelligence is severely limited.

In recent years, with the fast development of computational intelligence, advanced intelligent methods have been adopted in the design of StarCraft AI, with the aim of gaining better performance. Especially with the great success in board games and video games, CI researchers have now put more effort on the more complicated StarCraft challenges. New ideas and new methods are put forward to increase the intelligence and improve the performance of current StarCraft AI systems. It is obliged to review the recent development and compare related

²https://github.com/deepmind/pysc2



Fig. 2. Main components of computational intelligence for StarCraft AI.

works in this field for researchers and practitioners.

The structure of this survey is organized as illustrated in Fig. 2. The involved CI techniques include search-based algorithms, supervised learning, evolutionary computation, and reinforcement learning. In the context we will describe their applications in different levels, including build order, strategy recognition, winner prediction, gameplay, navigation, micromanagement, and mini games. The used platform and data sets are also described.

II. SEARCH-BASED ALGORITHMS

Search-based algorithms have long been used in board game AI. Unfortunately due to the complexity of the long-term planning and requirement of real-time operation in StarCraft, search-based algorithms are difficult in dealing with such complex decision-making, strategy planning and unit management. Nevertheless, researchers and engineers have successfully applied search-based techniques into the AI bots or smaller-scale subproblems, like simulation of combat and micromanagement of units. To satisfy the real-time requirement, search-based methods have to evaluate the result of each strategy and decision very rapidly and immediately.

A. Micromanagement

In micromanagement, AI needs to operate a squad of own units to fight against a squad of enemy units during a combat. Actions for each unit should be issued within every 55ms. For the limitation of vast state and action space, there is no search algorithm fast enough to be applied in full game environment successfully. However, hierarchical search is considered an efficient way to tackle the problem of large scale search space. [4] proposes Hierarchical Adversarial Search (HAS), which has a 3-layer version of the model. The top layer selects a set of goals to win the game, the middle layer generates feasible decisions and plans to achieve those goals, and the last layer evaluates those policies and commands the individual unit to execute those behaviors for the win. HAS outperforms stateof-the-art search-based algorithms such as Alpha-Beta based search [5], UCT based search [6], and Portfolio Search [7] in large-scale combat scenarios, in which up to 72 fighting units per player are engaged.

Recent years, a few approaches try to overcome the realtime limitation by using abstract scripts and game states. [8] proposes Puppet Search, which is an adversarial and lookahead search approach based on scripts. It innovatively selects a combination of a script and decisions to represent a move to be executed next. The part of script executes the move in the actual game and the other part of the abstract representation of the game state which can be applied by an adversarial tree search algorithm. Puppet Search has been implemented in a complete StarCraft bot and it matches or outperforms the stateof-the-art bots from the 2014 AIIDE StarCraft competition [9]. Puppet Search has also been tested in μ RTS, an abstract RTS game, and achieves a similar performance to other scripted and search-based agents in small scale, while outperforms them in large scale [10].

Game-tree-based search algorithms are conventional approaches for two-player games. However, this kind of algorithms requires a forward model to evaluate the state representation. In [11], the authors focus on the simulation of combat for two-player attrition in games. They present three forward models, Target-Selection Lanchester's Square Law Model (TS-Lanchester²), Sustained DPF Model (Sustained), and Decreasing DPF Model (Decreasing), that can be integrated into Monte Carlo tree search framework to play StarCraft. In order to deal with the enormous branching factors, the game state and action sets are abstracted as High-Level State Representation and High-Level Actions, and a mapping between low-level states and high-level states can be used by a game-tree-

based search algorithm to generate actions for each unit. The experiment results show that the combat models achieve better performance and are much faster than handcraft low-level models likes SparCraft.

B. Build Order

Search-based approaches are also used in build order optimization [12]. Build order is an essential component of macromanagement. It decides which kind of economy or production strategy to be adopted and what type of units to be produced. Build order optimization has been integrated into Build Order Search System (BOSS) [12] to find concurrent action sequences of buildings and units in the shortest time span. BOSS uses heuristics knowledge and abstractions to speed up the search for approximative solution in StarCraft. Experiment results show that BOSS is fairly comparable to professional StarCraft players in real-time.

III. SUPERVISED LEARNING

Replay data from human players provide valuable training samples for AI systems. Based on supervised learning (SL), it is possible to learn intelligent models without heavy human hand-crafted rules. SL refers to the technique that relies on labeled data and itself includes a number of methods. More data mean more chance to obtain a well-performed model. Motivated by that, a lot of groups have released various replay data sets successively, including Facebook [13] and Google DeepMind [3].

A. Micromanagement

Micromanagement is the basic unit control task for RTS games. [14] proposes a Bayesian model which is a distributed sensor-motor model for units control locally. Bayesian Programming is adopted into this model, and inverse programming is used as a fusion model for computing the complete inference in real-time. Authors implement two different Bayesian-based AI, BAIPB (Bayesian AI picking best) and BAIS (Bayesian AI sampling). The results show that BAIPB and BAIS both outperform the built-in AI in 12 and 36 ranged units, and BAIS is way better than BAIPB in both small and large armies.

B. Tactics

Tactics is to consider where, when, and how the the two sides will attack opposite units and structures. A generative Bayesian model [15] is used to predict attacks and take tactical decisions, especially considering uncertainty in enemy locations and technology tree. The model uses 7649 uncorrupted 1vs1 replays, and has lower-level heuristics from units observations. It makes most of the strategic inference and updates the parameters through supervised learning. The Bayesian tactical model eventually accomplishes prediction of opponent tactics and self decision-making under constraints and uncertainty easily and usefully.

C. Macromanagement

For professional players, the most crucial factor to win the game is to select the suitable macromanagement or strategy. It is significant to recognize the opponent's strategy, which usually is expressed in the form of build order. Replay data are used to predict the strategy and detect the change of build order in [16]. It adopts feature-expanded decision trees to predict the strategy, and it employs an equation to calculate the winning ratio from the replays to detect the change of build order. In [17], the authors investigate a probabilistic framework to learn the strategy behaviors. Based on hidden Markov models, the behavior models are trained by 331 expert-level StarCraft games and demonstrate the capability of predicting opponent behaviors and inferring the likely strategic state sequence. [18] presents a Bayesian model for opening prediction in StarCraft. This model only predicts the openings of the opponent, and learns its parameters through labeling replays. While [19] proposes a generic Bayesian model for long term strategic planning with noisy observations. They apply this model to StarCraft, and it performs a high quality and robust prediction which can form an adaptive AI.

Since deep learning has achieved great success in many application scenarios, [20] uses deep learning to learn macromanagement decisions in StarCraft directly from replays that include 789,571 state-action pairs. After integrating the trained network in UAlbertaBot, which is an open source bot written by Dave Churchill, the new bot can play competitively against the original UAlbertaBot that uses a fixed rush strategy, and significantly outperform the built-in Terran bot in StarCraft.

Fog of war is a crucial characteristic for StarCraft which means players cannot see their opponent behaviors if they fail in sending scout units or exploring the region. Professional players usually can assess the situation by scouting in the early phase of the game according to their experience. However, it seems difficult and infeasible to code all their experience into a bot. In order to investigate the effect of fog of war, [21] uses Random Forest (RF), Muti-Layer Perception, K-Nearest Neighbor for strategy prediction by replay data. The experiment results show that the rule-set approach performs worse than these machine learning approaches especially in the fog of war.

D. Winner Prediction

An interesting and challenging topic of game AI research is to predict which player will win. [22] investigates the individual and mixed models for combats between different races to predict winner in a StarCraft match. They use Gradient Boosting Regression Trees and Random Forest approaches, and address the problem as a binary classification. Their work shows that economic is the most important feature across all match types and achieves an improved accuracy (above 63%) compared to the previous works [23], [24]. It is well to be reminded that [25] uses influence maps, which are numerical matrices representing the influence of each player's military or combat units in the map, to model the game states. This approach has reached an impressive level of precision similar to the human recognition.

IV. EVOLUTIONARY COMPUTATION

Evolutionary Computation (EC) refers to a family of algorithms that are inspired by biological evolution when solving optimization problems. It relies on the evolution of a population, in which each individual represents a candidate solution to the problem. Through selection and mutation operators, the population gradually evolves to increase in fitness, so that the current best solution is continually improved. Because of the capability of producing highly optimized solutions in a wide range of problem settings, EC has attracted considerable attention from CI, also including the interest from StarCraft AI.

A. Build Order

Due to the sequentiality of building commands, build-order list is naturally suitable for EC applications. Each individual of population can be defined as the list of building commands. Based on proper fitness, the list can be optimized to achieve certain optimizing purpose. One difficulty exists in the problem is the validity of the list. Due to the limitation of technology, production, and resources, the candidate solutions must fulfill the hard requirement to be valid for game run. It makes EC must pay attention to the initialization, crossover, mutation, and other processes, during the evolution. Another difficulty is the definition of fitness. Because of the diversity of unit types in StarCraft and advantage of each type over the other, it is hard to give a comprehensive fitness function that is capable of determining which solution is superior to the other. Sometime AI has to choose its strategy (here refers to build order) according to the current game state as well as enemy strategy.

The first attempt of applying EC in StarCraft build-order optimization is the AI system of [26]. To fight against different enemies, the system searches to find a set of goals that best satisfy the needs at the current time. Authors use priority profile (PP) to determine which goal is preferred when conflicts arise. In addition to manually designed PP, they also use Genetic Algorithm (GA) to optimize the values, with each PP in the population to play games against build-in AI. The in-game scores define fitness. After a large number of game runs, which take three weeks even in distributed evaluation, the system with optimized PP is able to outperform the version based on hard-coded knowledge. Instead of optimizing goal priority, [27] directly applies GA to evolve a complete strategy for StarCraft, from the building plan to the composition of squads. They try two fitness metrics for the evaluation: victorybased fitness that is the vector of victory numbers against different opponents, and report-based fitness that is based on in-game metrics to give a smoother slope towards good solutions. The results show both fitness metrics are able to learn to defeat human-coded strategies and even a complete complex bot (OpprimoBot), but the first fitness has better winning rate. Both the above works use in-game scores or results for evaluation, so it is hard to give a direct evaluation to the build-order performance due to the influence from other modules like micromanagement and tactical strategies. Besides, the evaluation are given only at the end of each game run, which makes it impossible for real-time implementation.

In [28], authors consider multi-objective optimization with the target of producing most units of one or more types up to a certain time. They build a simulator so that it mimics the original StarCraft II characteristics but significantly reduces overall computational time. They use NSGA-II algorithm to solve the multi-objective problem, and Pareto fronts are produced by non-dominant sorting. Their method is able to produce a variety of build orders within the same game time, but they fail to further provide the best solution in terms of current game state and enemy strategies. In [29], authors also build a forward model to simulate outcome of build order. It can run continually in parallel, making it suitable for real-time and in-game implementation. To evaluate the candidate solutions, they define a unit makeup table and combine technology and upgrade bonus to assess both side forces. The proposed online evolutionary planning for in-game build order adaptation is capable of evolving diverse unit combination that clearly depends on combination of enemy units

B. Micromanagement

Another wide application of EC in StarCraft AI is micromanagement. In contrast to the sequentiality of build order, micromanagement has to promptly react to the change of game state and enemy unit behaviors. Motivated by that, AI systems usually use certain micromanagement policies and apply EC to optimize policy parameters. These policies include potential fields, neural networks, script policies, and so on.

In [30], authors first try to combine EC with potential fields to optimize the micromanagement for StarCraft. The fields on the enemy untis decide who to attack and which direction to retreat, while the fields placed at the center of the group direct the unit towards other friendly units when retreating. Parameters of field functions are tuning parameters. They define four objectives in the fitness and use the NSGA-II algorithm to optimize them at once. Compared to single objective, NSGA-II has a better learning curve. Unfortunately, the results are not satisfactory because they fail in outperforming other manually designed AI. The main reason authors give is due to the improper selection of potential functions and parameter ranges.

Both [31] and [32] use neuroevoluation to learn micromanagement. Their main difference exists in the structure of neural network. There are two types of inputs and three output nodes in [31]. Its input includes internal input that gives unit surroundings, as well as external input that represents overall game and combat information. The output neurons give unit decisions, advance or retreat, right or left, engage enemy or not. The network of [32] collects enemy and friendly unit information and takes unit own health, weapon cooldown as input. Its output nodes are simple, just fight and retreat commands. Both of these works use rtNEAT to evolve networks and consider four types of unit combat configurations, melee vs melee, melee vs range, range vs melee, and range vs range. The results are quite promising and demonstrate the methods are suitable for fast adaptation in real time. But authors in [32] also point out that it is hard to establish acceptation criteria and tell when to preserve winning behavior.

Other researchers also use simulators to mimic StarCraft micromanagement scenarios, so that squad or unit commands can be evolved in simulation and superior solutions are yielded. In [33], authors design a modular framework for simulating AI vs. AI conflicts through an XML specification. The realvalued behavioral parameters (28 in total) are evolved and finally the system achieves a success rate of 68% against the original version. But due to the large search space, each generation takes hours and a complete run takes approximately three days. To deal with that, [34] proposes a novel method by evolutionary search in the space of assignments of scripts to individual units. They combine Portfolio Greedy Search with online evolution, and use JarCraft, an open-source combat simulator to evaluate fitness. At last, it outperforms three other state-of-the-art algorithms for playing different StarCraft micro scenarios.

C. Gameplay

In addition to designing AI systems, EC is used to increase StarCraft gameplay. In [35], multi-objective evoluation algorithm is used to generate StarCraft maps. Each map is represented by indirect representation for searching and fitness testing, and direct representations for visualization. Multiple fitness functions are defined based on the measurement of playability, fairness, sill differentiation and interestingness. The output is Pareto front, such that each point corresponds to a viable map. Authors observe that fitness functions differ each other and show interesting conflicts. Some objectives lead to unsatisfied results, while some show nice features of the generated maps.

Game balancing is another concerning factor for designers and players. [36] uses genetic algorithm to change game parameters to balance game. By altering the unit attack and health parameters in the game, AI system evolves its strategies to fight against human players, so that players will not find it is too hard or too easy to play. Fitness is a multi-objective function of the winning rate and the difference between default and newly evolved parameters. Results show significant promise in improving game balancing.

V. REINFORCEMENT LEARNING

Reinforcement learning (RL) is an area of machine learning, and is very suitable for sequential decision-making tasks. In the last few years, deep learning has achieved remarkable performances in various domains, including reinforcement learning. The combination-deep reinforcement learning (DRL) [37], can teach agents to make decisions in high-dimension state space by an end-to-end framework, and has dramatically improved the generalization and scalability of traditional RL algorithms. To some extent, DRL is also a promising direction in StarCraft AI [38].

A. Micromanagement

At present, many researchers focus on micromanagement as the first step to study StarCraft AI, especially with DRL methods. [39] establishes StarCraft micromanagement scenarios as complex benchmarks for reinforcement learning. RL agents have to tackle a lot of challenges, such as durative actions, delayed rewards, and large action spaces. The authors introduce the greedy MDP with episodic zero-order optimization(GMEZO) algorithm that performs better than Deep Q Networks and Policy Gradient. [40] introduces BiCNet to play StarCraft combat games. BiCNet is a new deep multiagent reinforcement learning framework. It makes use of bidirectional neural networks, and learns collaboration via a vectorised actor-critic framework. The dependency of multiple units is modeled by bi-directional RNNs in hidden layers, and its gradient update is efficiently propagated through the entire networks. BiCNet can successfully learn some coordination strategies similar to these of professional StarCraft players. Moreover, BiCNet is easily adaptable to various tasks with different units. In the given scenarios, BiCNet shows better performances than rule based methods and GMEZO.

In [39] and [40], the authors focus on developing centralized controllers to play micromanagement. [41] proposes a multiagent actor-critic method to tackle decentralized micromanagement tasks. In order to stabilize experience replay in deep multi-agent reinforcement learning, the authors present fingerprints and importance sampling methods. The experience replay with fingerprints and importance sampling(XP+FP/IS) can help DRL agents to disambiguate between episodes from different training process. XP+FP/IS can also partially recover performance on account of nonstationarity, and improve the performance over centralized RL controllers. [42] dedicates to explore more efficient state representation to break down the complexity caused by the large state space in micromanagement. The authors propose the parameter sharing multi-agent gradient descent Sarsa(λ) (PS-MAGDS) algorithm to solve the multi-agent decision making problem [43]. In addition, they introduce curriculum transfer learning to extend the RL model to various micromanagement scenarios. This helps to improve the sample efficiency, and shows a competitive performance with GMEZO and BiCNet. [44] proposes a new multi-agent actor-critic method called counterfactual multi-agent (COMA) policy gradients. COMA uses a centralised critic to estimate the Q-function and decentralised actors to optimise the agents' policies. In addition, it uses a counterfactual baseline to address the challenges of multi-agent credit assignment. In decentralised StarCraft micromanagement with partial observability, COMA significantly improves average performance over other multi-agent actor-critic methods. [45] revisits the idea of the master-slave architecture by incorporating both perspectives within one framework, and proposes master-slave multi-agent reinforcement learning (MS-MARL) method. MS-MARL highlights three key ingredients, i.e. composed action

representation, learnable communication and independent reasoning. With network designed to facilitate these explicitly, MS-MARL outperforms latest competing methods in challenging StarCraft micromanagement tasks.

Based on SC2LE, [46] proposes QMIX, a novel value-based DRL method that trains decentralised policies in a centralised end-to-end way. QMIX estimates joint action-values as a complex non-linear combination of per-agent values. These values condition only on local observations through a neural network. The authors evaluate QMIX on a set of difficult StarCraft II micromanagement tasks, and show that QMIX significantly outperforms existing value-based deep multiagent reinforcement learning methods.

B. Build Order

Apart from micromanagement, DRL method is also used to optimize the build order in StarCraft. This problem concerns the order of buildings and units based on current game situation. In contrast to hand-craft methods, [47] proposes two DRL models: neural network fitted Q-learning (NNFQ) and convolutional neural network fitted Q-learning (CNNFQ). NNFQ and CNNFQ are used to build StarCraft II bots to fight against the built-in AI in simple maps. Experimental results show that these two models are capable of finding the most effective production sequences to defeat opponents.

C. Navigation

In StarCraft, units need to move efficiently from one position to another. Sometimes they have to avoid certain obstacles and unwalkable areas. To solve this problem in StarCraft, [48] presents value propagation (VProp), which is a parameter-efficient differentiable planning module based on value iteration. VProp can successfully be trained using reinforcement learning, and scale to larger maps. Based on VProp, units learn to navigate in dynamic StarCraft environments. Furthermore, the module enables units learning to plan when the environment includes stochastic elements, providing a cost-efficient learning system to build effective planners for a variety of navigation problems.

D. Mini-games

In [3], DeepMind researchers present initial baseline results for classical DRL agents applied to the StarCraft II domain. In mini-games, fully convolutional advantage actor-critic (FullyConv-A2C) agents learn to achieve a level of beginners. However, these agents are unable to make significant progress on the main game. [49] proposes asymmetric self-play to build Marine units, which greatly speeds up learning, and surpasses the count-based approach. [50] introduces the relational DRL, which uses self-attention to iteratively reason about the relations between entities in a scene and to guide a model-free policy. Through structured perception and relational reasoning, this method improves the efficiency, generalization capacity, and interpretability of conventional approaches. In SC2LE, the relational DRL agent achieves state-of-the-art performance on six mini-games, and surpasses human experts on four.

 TABLE I

 COMPARISONS OF DRL METHODS IN STARCRAFT AI.

Methods	Scenarios	Testbeds
GMEZO	Micromanagement	TrochCraft
BiCNet	Micromanagement	GymCraft
XP+FP/IS	Micromanagement	TorchCraft
PS-MAGDS	Micromanagement	BWAPI
COMA	Micromanagement	TorchCraft
QMIX	Micromanagement	SC2LE
(C)NNFQ	Build order	SC2LE
Reinforce+selfplay	Build order	TrochCraft
VProp	Navigation	TrochCraft
FullyConv-A2C	Mini-games	SC2LE
Relational DRL	Mini-games	SC2LE

VI. CONCLUSION

In this paper, we review the development of CI in StarCraft AI. Different methods and their successful applications in StarCraft AI are introduced. It is obvious that it is difficult if not impossible to use a uniform framework to design an intelligent system. Different tasks represent different abstraction and involves different time spans. Upper and lower tasks mutually interact with each other. As a consequence, a highly intelligent StarCraft AI system requires a combination of multiple CI techniques so that different advantages can be maximally utilized. It also encourages a novel framework to be efficiently implemented in the real-time environment.

References

- [1] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game AI research and competition in StarCraft," *IEEE Transactions on Computational Intelligence* and AI in Games, vol. 5, no. 4, pp. 293–311, 2013.
- [2] G. Synnaeve, N. Nardelli, A. Auvolat, S. Chintala, T. Lacroix, Z. Lin, F. Richoux, and N. Usunier, "TorchCraft: a library for machine learning research on real-time strategy games," *CoRR*, vol. abs/1611.00625, 2016.
- [3] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. P. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. W. Lawrence, A. Ekermo, J. Repp, and R. Tsing, "StarCraft II: A new challenge for reinforcement learning," *CoRR*, vol. abs/1708.04782, 2017.
- [4] M. Stanescu, N. A. Barriga, and M. Buro, "Hierarchical adversarial search applied to real-time strategy games," in *Proceedings of the Tenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-14)*, 2014, pp. 66–72.
- [5] D. Churchill and M. Buro, "Incorporating search algorithms into RTS game agents," in *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-12)*, 2012, pp. 2–7.
- [6] D. Churchill, A. Saffidine, and M. Buro, "Fast heuristic search for RTS game combat scenarios," in *Proceedings of the Eighth AAAI Conference* on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-12), 2012, pp. 112–117.
- [7] D. Churchill and M. Buro, "Portfolio greedy search and simulation for large-scale combat in StarCraft," in 2013 IEEE Conference on Computational Intelligence and Games (CIG), 2013, pp. 1–8.
- [8] N. A. Barriga, M. Stanescu, and M. Buro, "Puppet search: Enhancing scripted behavior by look-ahead search with applications to real-time strategy games," in *Proceedings of the Eleventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-15)*, 2015, pp. 9–15.

IEEE Symposium Series on Computational Intelligence SSCI 2018

- [9] D. Churchill, M. Preuss, F. Richoux, G. Synnaeve, A. Uriarte, S. Ontañnón, and M. Čertický, "StarCraft bots and competitions," *Encyclopedia of Computer Graphics and Games*, no. 1, pp. 1–18, 2016.
- [10] N. A. Barriga, M. Stanescu, and M. Buro, "Game tree search based on non-deterministic action scripts in real-time strategy games," *IEEE Transactions on Games*, vol. 10, no. 1, pp. 69–77, 2018.
- [11] A. Uriarte and S. Ontañnón, "Combat models for RTS games," *IEEE Transactions on Games*, vol. 10, pp. 29–41, 2018.
- [12] D. Churchill and M. Buro, "Build order optimization in StarCraft," in Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-11), 2011, pp. 14–19.
- [13] Z. Lin, J. Gehring, V. Khalidov, and G. Synnaeve, "STARDATA: A StarCraft AI research dataset," *CoRR*, vol. abs/1708.02139, 2017.
- [14] G. Synnaeve and P. Bessiere, "A Bayesian model for RTS units control applied to StarCraft," in *Computational Intelligence and Games*, 2011, pp. 190–196.
- [15] G. Synnaeve and P. Bessière, "Special tactics: A Bayesian approach to tactical decision-making," in *Computational Intelligence and Games*, 2012, pp. 409–416.
- [16] H. C. Cho, K. J. Kim, and S. B. Cho, "Replay-based strategy prediction and build order adaptation for StarCraft AI bots," in 2013 IEEE Conference on Computational Intelligence and Games (CIG), 2013, pp. 1–7.
- [17] E. Dereszynski, J. Hostetler, A. Fern, T. Dietterich, T. T. Hoang, and M. Udarbe, "Learning probabilistic behavior models in real-time strategy games," in *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-11)*, 2011, pp. 20–25.
- [18] G. Synnaeve and P. Bessiere, "A Bayesian model for opening prediction in RTS games with application to StarCraft," in *Computational Intelli*gence and Games, 2011, pp. 281–288.
- [19] G. Synnaeve and P. Bessière, "A Bayesian model for plan recognition in RTS games applied to StarCraft," in *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-11)*, 2011, pp. 79–84.
 [20] N. Justesen and S. Risi, "Learning macromanagement in StarCraft from
- [20] N. Justesen and S. Risi, "Learning macromanagement in StarCraft from replays using deep learning," in 2017 IEEE Conference on Computational Intelligence and Games (CIG), 2017, pp. 162–169.
- [21] H. Cho, H. Park, C. Y. Kim, and K. J. Kim, "Investigation of the effect of fog of war in the prediction of StarCraft strategy using machine learning," *Computers in Entertainment*, vol. 14, no. 2, 2016.
- [22] Y. N. Ravari, S. Bakkes, and P. Spronck, "StarCraft winner prediction," in Proceedings of the Twelve AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-16), 2016, pp. 2–8.
- [23] G. Erickson and M. Buro, "Global state evaluation in StarCraft," in Proceedings of the Tenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-14), 2014, pp. 112–118.
- [24] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction," in 2009 IEEE Conference on Computational Intelligence and Games (CIG), 2009, pp. 140–147.
- [25] A. A. Sánchez-Ruiz and M. Miranda, "A machine learning approach to predict the winner in StarCraft based on influence maps," *Entertainment Computing*, vol. 19, pp. 29–41, 2017.
- [26] J. Young and N. Hawes, "Evolutionary learning of goal priorities in a real-time strategy game," in *Proceedings of the Eighth AAAI Conference* on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-12), 2012.
- [27] P. García-Sánchez, A. P. Tonda, A. M. García, G. Squillero, and J. J. M. Guervós, "Towards automatic StarCraft strategy generation using genetic programming," 2015 IEEE Conference on Computational Intelligence and Games (CIG), pp. 284–291, 2015.
- [28] H. Köstler and B. Gmeiner, "A multi-objective genetic algorithm for build order optimization in StarCraft II," KI - Künstliche Intelligenz, vol. 27, pp. 221–233, 2013.
- [29] N. Justesen and S. Risi, "Continual online evolutionary planning for ingame build order adaptation in StarCraft," in *Proceedings of the Genetic* and Evolutionary Computation Conference, ser. GECCO '17, 2017, pp. 187–194.
- [30] J. B. Svendsen and E. A. Rathe, "Micromanagement in StarCraft using potential fields tuned with a multi-objective genetic algorithm," Master's thesis, Norwegian University of Science and Technology, 2012.
- [31] I. Gabriel, V. Negru, and D. Zaharie, "Neuroevolution based multiagent system for micromanagement in real-time strategy games," in

Proceedings of the Fifth Balkan Conference in Informatics, ser. BCI '12, 2012, pp. 32–39.

- [32] J. S. Zhen and I. Watson, "Neuroevolution for micromanagement in the real-time strategy game StarCraft: Brood War," in AI 2013: Advances in Artificial Intelligence, S. Cranefield and A. Nayak, Eds., 2013, pp. 259–270.
- [33] N. Othman, J. Decraene, W. Cai, N. Hu, M. Y. H. Low, and A. Gouaillard, "Simulation-based optimization of StarCraft tactical AI through evolutionary computation," in 2012 IEEE Conference on Computational Intelligence and Games (CIG), Sept 2012, pp. 394–401.
- [34] C. Wang, P. Chen, Y. Li, C. Holmgård, and J. Togelius, "Portfolio online evolution in StarCraft," in *Proceedings of the Tweleveth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (AIIDE-16), 2016, pp. 114–120.
- [35] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, and G. Yannakakis, "Multiobjective exploration of the StarCraft map space," in *Proceedings of the 2010 IEEE Conference on Computational Intelli*gence and Games (CIG), 2010, pp. 265–272.
- [36] M. Morosan and R. Poli, "Automated game balancing in Ms PacMan and StarCraft using evolutionary algorithms," in *Applications of Evolutionary Computation*, G. Squillero and K. Sim, Eds. Cham: Springer International Publishing, 2017, pp. 377–392.
- [37] D. Zhao, K. Shao, Y. Zhu, D. Li, Y. Chen, H. Wang, D. Liu, T. Zhou, and C. Wang, "Review of deep reinforcement learning and discussions on the development of computer Go," *Control Theory and Applications*, vol. 33, no. 6, pp. 701–717, 2016.
- [38] Z. Tang, K. Shao, D. Zhao, and Y. Zhu, "Recent progress of deep reinforcement learning: from AlphaGo to AlphaGo Zero," *Control Theory and Applications*, vol. 34, no. 12, pp. 1529–1546, 2017.
- [39] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala, "Episodic exploration for deep deterministic policies: an application to StarCraft micromanagement tasks," in *International Conference on Learning Representations*, 2017.
- [40] P. Peng, Q. Yuan, Y. Wen, Y. Yang, Z. Tang, H. Long, and J. Wang, "Multiagent bidirectionally-coordinated nets for learning to play Star-Craft combat games," *CoRR*, vol. abs/1703.10069, 2017.
- [41] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. S. Torr, P. Kohli, and S. Whiteson, "Stabilising experience replay for deep multi-agent reinforcement learning," in *International Conference on Machine Learning*, 2017.
- [42] K. Shao, Y. Zhu, and D. Zhao, "StarCraft micromanagement with reinforcement learning and curriculum transfer learning," *IEEE Transactions on Emerging Topics in Computational Intelligence*, DOI:10.1109/TETCI.2018.2823329, 2018.
- [43] —, "Cooperative reinforcement learning for multiple units combat in StarCraft," in *IEEE Symposium Series on Computational Intelligence*, 2017, pp. 1–6.
- [44] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *The 32nd AAAI Conference on Artificial Intelligence*, 2018.
- [45] X. Kong, B. Xin, F. Liu, and Y. Wang, "Revisiting the master-slave architecture in multi-agent deep reinforcement learning," *CoRR*, vol. abs/1712.07305, 2017.
- [46] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. N. Foerster, and S. Whiteson, "QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning," *CoRR*, vol. abs/1803.11485, 2018.
- [47] Z. Tang, D. Zhao, Y. Zhu, and P. Guo, "Reinforcement learning for buildorder production in StarCraft II," in *The 8th International Conference* on Information Science and Technology (ICIST 2018), 2018.
- [48] N. Nardelli, G. Synnaeve, Z. Lin, P. Kohli, P. H. S. Torr, and N. Usunier, "Value propagation networks," *CoRR*, vol. abs/1805.11199, 2018.
- [49] S. Sukhbaatar, I. Kostrikov, A. Szlam, and R. Fergus, "Intrinsic motivation and automatic curricula via asymmetric self-play," *CoRR*, vol. abs/1703.05407, 2017.
- [50] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart *et al.*, "Relational deep reinforcement learning," *CoRR*, vol. abs/806.01830, 2018.