Journal of Information Processing Vol.0 No.0 0–6 (??? 1992)

14-K008f

[DOI: 10.2197/ipsjjip.0.0]

Regular Paper

To appear in vol. 23, no. 1, 2015.

Heuristic Search Exploiting Non-Additive and Unit Properties for RTS-Game Unit Micromanagement

Tung Duc Nguyen[1,†1] Kien Quang Nguyen[2,†2] Ruck Thawonmas[1,a)]

Received: xx 2, 2014, Accepted: xx 2, 2014

Abstract: This paper presents an approach that integrates fuzzy integral and fast heuristic search for improving the quality of unit micromanagement in the popular RTS game StarCraft. Unit micromanagement, i.e., detailed control of units in combat, is one of the most challenging problems posed by RTS games and is often tackled with search algorithms such as Minimax or Alpha-Beta. Due to vast state and action spaces, the game tree is often very large, and search algorithms must rely on evaluation methods from a certain limited depth rather than exploring deeper into the tree. We therefore attempt to apply fuzzy integral and aim for an evaluation method with high accuracy in the search. To achieve this aim, we propose a new function that allows fuzzy integral to cope with not only non-additive properties but also unit properties in RTS games. Experimental results are reported at the end of this paper, showing that our approach outperforms an existing approach in ~~terns~~ of win rates in this domain.

Keywords: Heuristic search, Fuzzy integral, Unit properties, RTS games, Unit micromanagement

1. Introduction

Real-time strategy (RTS) is a sub-genre of strategy video games which normally involves resource gathering, base constructing, strategy planning, and combat scenarios. In typical RTS games, such as Age of Empires, WarCraft or StarCraft, each player is supposed to produce many combat units and build up a powerful army with the ultimate goal of destroying all units and buildings of the enemy. With fast-paced gameplay and the existence of simultaneous moves, RTS games have reached a level of complexity unseen in other traditional games like Chess or Go. This also helped developing AI systems for RTS games gain increasing attention among the AI research community in recent years [1]–[9].

In StarCraft, one of the most popular RTS games, unit micromanagement not only is the key to winning a battle but also decides the result of the whole game. With high quality micromanagement, one side can completely destroy the other side that commands even more or stronger units. Although there have been several AI studies on this domain, bots whose unit behaviors are predefined via scripts (sequences of rule-based actions) still predominate in AI competitions. However, due to their non-adaptive nature, they are highly exploitable and can be countered quite easily by using appropriate countermeasures. Therefore, a recent trend in unit micromanagement is that of using search-based techniques to dynamically control units while still considering collaboration among them. Some state-of-the-art methods such as Alpha-Beta [10], UCT [11], and Monte Carlo planning [12] have been applied and achieved dominance over script-based techniques.

It is known that the performance of a search-based technique relies mainly on its heuristic evaluation function as such a function is required in almost all search algorithms. Even Monte-Carlo tree search, which performs simulation of possible future game-scenarios to evaluate the current state, due to time restraints in RTS games, still needs a weak evaluation function when such simulation cannot reach its end. In StarCraft, heuristic functions tend to be very generic (such as LTD and LTD2 used in [10]), and therefore cannot fully capture the game state, especially when multiple types of units can interact and boost each other.

Recently, Stanescu et al. [13] built a system to predict a combat's outcome, in terms of the winning probability of each side. Their prediction results could also be used as an evaluation function. However, that system considers each unit type separately and later combines the result of each type. It, therefore, does not take into account the interaction among different unit types in RTS games, which have non-additive properties as discussed in Section 2.1.

In order to improve the performance of search algorithms, we propose an evaluation method using fuzzy integral that can accurately evaluate a game state even when there are several unit types involved. We then apply this evaluation method to heuristic search for unit micromanagement. The contributions of this paper are as follows:

- an accurate evaluation method that can cope with non-additive properties and unit properties in RTS games and
- a successful application of this evaluation method to heuristic search for RTS-game unit micromanagement.

[1] College of Information Science & Engineering, Ritsumeikan University, Japan
[2] Graduate School of Information Science & Engineering, Ritsumeikan University, Japan
[†1] Presently with Framgia Vietnam, CO., LTD, Vietnam
[†2] Presently with Microsoft Development Ltd., Japan
[a)] ruck@ci.ritsumei.ac.jp

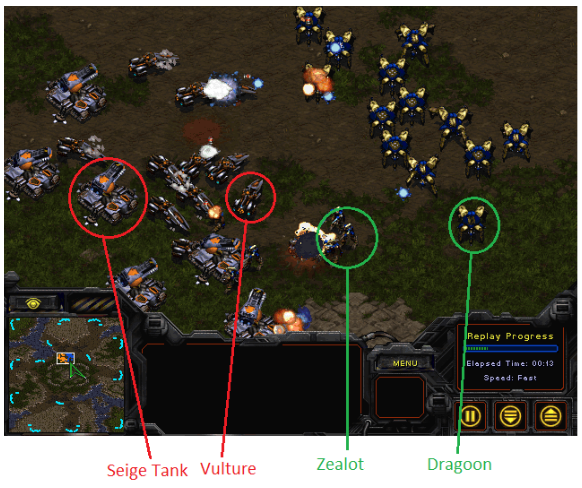

Fig. 1 A combination of Siege Tank and Vulture (red circles) against a combination of Dragoon and Zealot (green circles).

2. Background

2.1 Non-additive Properties in RTS Games

There may be dozens of unit types that can interact in different ways in an RTS game. This leads to enormous numbers of possible unit combinations. A combination of two unit types may result in greater or less impacts than the sum of their individual impacts. Suppose $\mu(X)$ is the measure of effectiveness provided by unit combination X. In RTS games, it is common that $\mu(X)$ is non-additive, i.e., $\mu(X_1 \cup X_2) \geq \mu(X_1) + \mu(X_2)$ or $\mu(X_1 \cup X_2) \leq \mu(X_1) + \mu(X_2)$. For example, a combination of Siege Tank and Vulture is a standard tactic in StarCraft. Siege Tank (normally used in siege mode) is a powerful unit with massive damage and very long attack range. However, it has a minimum attack range of 2, making it quite weak against melee (short-range) units. On the other hand, Vulture is an extremely mobile unit which moves quickly enough to be able to "hit and run" against most melee units without risk. Thus, Siege Tank and Vulture are widely used to protect each other. In this case, one can generally say that $\mu(\{Tank, Vulture\}) > \mu(\{Tank\}) + \mu(\{Vulture\})$.

2.2 Fuzzy Measure and Fuzzy Integral

Measure is an important concept in mathematical analysis. A measure on a set is a function that assigns a number to each suitable subset of that set. Originally, the measure of a 'large' subset, which is composed of a finite number of 'smaller' disjoint subsets, is equal to the sum of the measures of the 'smaller' subsets. This is the main characteristic of a classical measure and known as the additive property. Due to the existence of non-additive properties in many practical applications, classical measure does not always perform well. To circumvent this issue, fuzzy measure was introduced that replaces the additive property with the weaker property of monotonicity [14].

Definition 1: A fuzzy measure on a measurable space (X, F) is a real-valued set function $\mu : F \to \mathbb{R}$ satisfying:

(1) $\mu(\emptyset) = 0$,

(2) $\mu(A) \leq \mu(B)$ whenever $A \in F, B \in F, A \subseteq B$.

Definition 2: A non-monotonic fuzzy measure on (X, F) is a real-valued set function $\mu : F \to \mathbb{R}$ satisfying only $\mu(\emptyset) = 0$.

Non-monotonic fuzzy measure matches well with non-additive properties in RTS games and can be learned by machine learning methods without expert knowledge.

Choquet integral [15], named after the French mathematician Gustave Choquet, is an expansion of Lebesgue integral and a classical fuzzy integral. It has been successfully applied in many areas such as statistical mechanics and potential theory. The definition of Choquet integral with respect to a fuzzy measure is as follows.

Definition 3: Let μ be a fuzzy measure on X. The discrete Choquet integral of a function $f : X \to \mathbb{R}$ with respect to μ is defined by

$$\int f(x) \circ \mu(X) = \sum_{i=1}^{n} \left(\left(f(x_i) - f(x_{i-1}) \right) \times \mu(A_i) \right) \qquad (1)$$

where $A_i = \{x | f(x) \geq f(x_i)\}$, $0 = f(x_0) \leq f(x_1) \leq f(x_2) \leq \ldots \leq f(x_n)$.

3. Related Work

3.1 Fuzzy Integral for Unit Selection Problem in RTS Game

So far, there has been only a limited number of AI research incorporating fuzzy measure and fuzzy integral into RTS game agents. Li, Y.J. et al. applied different fuzzy integrals to solve the unit selection problem in RTS games [16]. The key to winning in most RTS games is to build up a strong army with appropriate unit types which can gain massive destroy power against the enemy army. Different unit combinations give different effectiveness. Therefore, to estimate the power (strength) of each unit combination becomes one of the most essential tasks in the game. Due to interactions occurring among different unit types, the effectiveness of a unit combination cannot be simply calculated by using weighted average. As a result, they tried to apply fuzzy integral to that calculation. In their research, they proposed three new fuzzy integrals and compared them with the classical Choquet integral. Because we base our work on part of this previous work, the details of those fuzzy integrals are given below. Note that the work by Li, Y.J. et al. focuses on strategy planning (i.e., their main purpose is to find the most powerful unit combination to produce) while our work focuses on unit micromanagement (i.e., our main purpose is to control combat units in order to win the battle and gain strategic advantages over the enemy). In the following formulas, X is a unit combination, x is a unit type, $f(x)$ is defined as the proportion of x, and the fuzzy integral returns the effectiveness of that combination.

3.1.1 Max-based Fuzzy Integral

Max-based fuzzy integral is developed with the policy of winner takes all. It considers only the most powerful unit combination which involves the unit type of interest as follows:

$$\int f(x) \circ \mu(X) = \sum_{i=1}^{n} \left(f(x_i) \times \max\left(\mu(S_i)\right) \right) \qquad (2)$$

where n is the number of unit types, and S_i is a combination that includes x_i.

3.1.2 Mean-based Fuzzy Integral

Mean-based fuzzy integral is calculated considering all the interactions that are related to each unit type. All the fuzzy measures which involve the unit type of interest will be selected, and the average value is computed as follows:

$$\int f(x) \circ \mu(X) = \sum_{i=1}^{n} \left(f(x_i) \times \frac{1}{m_i} \sum_{j=1}^{m_i} \mu\left(S_{ij}\right) \right) \tag{3}$$

where n is the number of unit types, S_{ij} is a combination that includes x_i, and m_i is the number of such combinations.

3.1.3 Order-based Fuzzy Integral

Order-based fuzzy integral takes into account the unit production in RTS games. Data analysis shows that advanced or strong units often dominate the proportion in the army and thus should be considered as having more cooperation with other units. For each unit type, the interaction will be calculated with the one having less proportion than it, and the largest $f(x)$ is combined with the fuzzy measure of all unit types as follows:

$$\int f(x) \circ \mu(X) = \sum_{i=1}^{n} \left(f(x_i) \times \mu(\{x | f(x) \le f(x_i)\}) \right) \tag{4}$$

where n is the number of unit types, and $f(x_1) \ge f(x_2) \ge ... \ge f(x_n) > 0$.

3.1.4 Technical Issue in the Previous Work

All of the three fuzzy integrals above have been proven to give better results than the classical Choquet integral when applied to RTS games. However, because their research focuses on finding the most powerful unit combination to produce, their function $f(x)$ is defined without considering the properties of individual units (damage, cooldown, current hit points). In detail, their $f(x)$ is defined as

$$f(x) = n/N, \tag{5}$$

where N is the maximum number of units that a player can have in the game and n represents the number of type-x units.

For example, suppose that there are three different unit combinations with the same proportions between the member unit types as follows:

(1) 5 Zealots and 5 Dragoons, each unit having 50 hit points,
(2) 5 Zealots and 5 Dragoons, each unit having 100 hit points,
(3) 10 Zealots and 10 Dragoons, each unit having 100 hit points.

It can easily be seen that the fuzzy integrals mentioned above return the same value for all of these three combinations. Thus, this prompted us to come up with another definition of $f(x)$ when we apply those fuzzy integrals to our problem—unit micromanagement. A comparison between the original function (5) and our proposed function is shown in Section 5.1.

3.2 Portfolio Greedy Search

Combat scenarios in RTS games can be classified as two-player zero-sum simultaneous move games, in which each player is supposed to find a sequence of moves that lead to a victory over the opponent. Search algorithms can be used to solve this problem, but they have to deal with various difficulties such as vast state and action spaces, huge branching factor, harsh constraints on computational resources. Churchill, D. and Buro, M. proposed a novel greedy search called Portfolio Greedy Search (PGS) that uses a hill-climbing technique to reduce the branching factor in RTS games' combats [11]. Instead of searching all possible actions for each unit, PGS only considers actions created by a set of scripts called a *portfolio*. In addition, although it determines a node's value based on playout and a heuristic evaluation function, only a single playout is carried out, unlike UCT which requires multiple playouts. In order to evaluate a game state, PGS first performs a deterministic script-based playout, and after the playout reaches its terminal condition, it calls the following evaluation formula.

$$LTD2(s) = \sum_{u \in U_1} \sqrt{hp(u)} \times dpf(u) - \sum_{u \in U_2} \sqrt{hp(u)} \times dpf(u) \tag{6}$$

Here s, hp, dpf, U_1, and U_2 denote a game state, hit points, damage per frame, set of units controlled by player 1 (the current player) and player 2, respectively. Intuitively, by assuming both players use the same policy and performing a playout, one can estimate which player has an advantage at a given state. It has been shown that Portfolio Greedy Search can outperform both state-of-the-art search methods—Alpha-Beta and UCT—in large StarCraft combat scenarios where the branching factor is large. As a result, in our work, we adopt this search method (see Algorithm 1 excerpted from [11] for the details) as our targeted heuristic search.

4. Methodology

Our main idea is to improve the quality of evaluation functions used in existing search algorithms by applying fuzzy measure and fuzzy integral to estimate the value of a game state. In our problem, we define that value as the difference between the power of the player's army and the enemy's army, where the power of one's army is estimated by calculating the following fuzzy integral:

$$Army\ power = \int f(x) \circ \mu(X) \tag{7}$$

Here $\mu(X)$ is the fuzzy measure of the corresponding unit combination and can be considered as its contribution to the total power. In addition, $f(x)$ is a unit statistic function defined by

$$f(x) = \frac{1}{N} \sum_{i=1}^{n} \frac{hp(u_i)}{max\_hp(u_i)} \tag{8}$$

where x is a unit type, N is the maximum number of units that a player can have in the game, n is the number of type-x units, u_i is a type x unit, and max_hp denotes the maximum number of hit points. Unlike the original function (5), our function includes the temporary unit status (in this case, the current hit points of each unit); it could thus solve the technical issue discussed in Section 3.1.4. Note that when all units have the possible maximum hit points, the value of $f(x)$ is equal to the proportion of type-x units and our function becomes the original one.

4.1 Data Collection and Analysis

We selected StarCraft: Brood War (SC: BW) as our research platform. It is a military science fiction, real-time strategy game released by Blizzard Entertainment in 1998. As of February 2009, SC: BW has sold more than 11 million copies and became one of the most popular video games of all time. For AI

Algorithm 1 Portfolio Greedy Search [11]

```
1: Portfolio $P$                                        ▹ Script Portfolio
2: Integer $I$                                          ▹ Improvement Iterations
3: Integer $R$                                          ▹ Self/Enemy Improvement Responses
4: Script $D$                                           ▹ Default Script
5:
6: procedure PortfolioGreedySearch(State $s$, Player $p$)
7:     Script enemy[$s$.numUnits(opponent($p$))].fill($D$)
8:     Script self[] ← GetSeedPlayer($s$, $p$, enemy)
9:     enemy ← GetSeedPlayer($s$, opponent($p$), self)
10:    self ← Improve($s$, $p$, self, enemy)
11:    for $r = 1$ to $R$ do
12:        enemy ← Improve($s$, opponent($p$), enemy, self)
13:        self ← Improve($s$, $p$, self, enemy)
14:    return generateMove(self)
15:
16: procedure GetSeedPlayer(State $s$, Player $p$, Script $e$[])
17:    Script self[$s$.numUnits($p$)]
18:    bestValue ← $-\infty$
19:    Script bestScript ← Ø
20:    for Script $c$ in $P$ do
21:        self.fill($c$)
22:        value ← Playout($s$, $p$, self, $e$)
23:        if value > bestValue then
24:            bestValue ← value
25:            bestScript ← $c$
26:    self.fill(bestScript)
27:    return self
28:
29: procedure Improve(State $s$, Player $p$, Script self[], Script $e$[])
30:    for $i = 1$ to $I$ do
31:        for $u = 1$ to self.length do
32:            if timeElapsed > timeLimit then
33:                return
34:            bestValue ← $-\infty$
35:            Script bestScript ← Ø
36:            for Script $c$ in $P$ do
37:                self[u] ← $c$
38:                value ← Playout($s$, $p$, self, $e$)
39:                if value > bestValue then
40:                    bestValue ← value
41:                    bestScript ← $c$
42:            self[u] ← bestScript
43:    return self
```

researchers, SC: BW is also an ideal test bed for AI algorithms, thanks to the BWAPI (Brood War API [17])'s comprehensive interface for accessing the game engine.

300 replays of professional one-versus-one SC: BW games were collected from the Internet [*1]. We focused only on Protoss (one of the three character races in StarCraft) vs. Protoss match-up. This is the only match-up that provides multiple most used unit types and is available in a simulator called SparCraft used in our experiments described in Section 5. A program written in C++ was used to analyze the replays, from which the data of 940 battles were obtained. Before and after each battle, the unit statistics and the score of both players were recorded. Those scores were given by the game system after the winner destroyed his enemy units and then used to learn the fuzzy measure in our research.

[*1] http://www.bwreplays.com/
http://www.teamliquid.net/replay/

4.2 Learning Fuzzy Measure by Genetic Algorithm

There is a relation among the score a player obtains in each battle, the strength of his army and the result of the battle. Our data analysis shows that the higher the score, the more powerful the army and that the higher chance of winning. Thus, we can consider the estimated power of the army as the expected value of the score. This leads to the problem of finding the fuzzy measure that "best" fits the collected data. In this paper, we used a genetic algorithm approach to learn the aforementioned fuzzy measure, as done in [16].

Each chromosome was composed of $2^n - 1$ fuzzy measures corresponding to all possible unit combinations, i.e., $\mu(x_1)$, $\mu(x_2)$, ..., $\mu(x_1, x_2)$, ..., $\mu(x_1, x_2, ..., x_n)$ where x_1, x_2, ..., x_n represent n kinds of unit types. In our work, there were 8 unit types included in the collected replays ($n = 8$), and real-valued encoding was used, which resulted in the length of a chromosome being 255. The fitness calculation of a chromosome is described as follows:

(1) Extract the fuzzy measure from that chromosome.
(2) Extract real scores and values of the unit statistic function from the replays, and normalize those real scores between 0 and 1.
(3) Calculate the estimated scores by using fuzzy integral in (7).
(4) Calculate the root mean square error over the training data:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(score_{est} - score_{real})^2}, \qquad (9)$$

where N represents the number of cases in the training data; because we collected data from 940 battles, with each battle contributing two training cases (two sides), $N = 1880$.
(5) Calculate the fitness:

$$Fitness = \frac{1}{1 + RMSE} \qquad (10)$$

In our GA, a mixture of roulette wheel selection and elitist selection was used to construct a new population. Population size, number of generations, crossover rate, and mutation rate were set to 1000, 500, 0.75, 0.05, respectively (the last two settings were made following the recipe in [18]). The result of the learning process is illustrated in Fig. 2. Shown are the best fitnesses obtained at each generation by using different fuzzy integrals: max-based fuzzy integral, mean-based fuzzy integral, and order-based fuzzy integral. As can be seen, order-based fuzzy integral showed the best performance among those three methods, which also confirms the findings obtained by Li, Y.J. et al. [16]. Thus, only order-based fuzzy integral and the corresponding fuzzy measure are used in our evaluation below.

5. Evaluation

Two main series of experiments were carried out to compare the performance of the proposed fuzzy-based evaluation method and another existing evaluation method—LTD2 described in Section 3.2. SparCraft, an open source simulation package that can simulate StarCraft combats with a high level of accuracy [19],

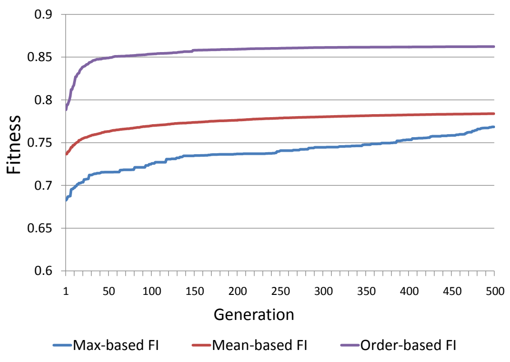

Fig. 2 Best fitnesses at each generation obtained by different fuzzy integrals. The best value is 0.86.

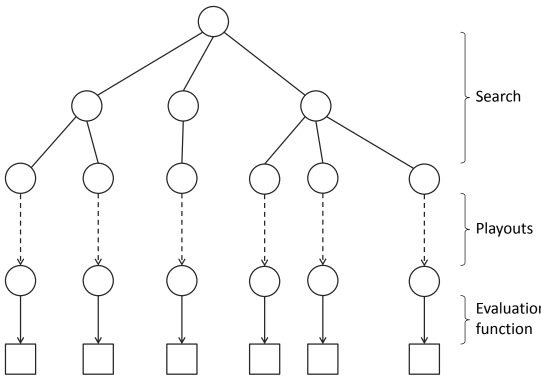

Fig. 3 The general mechanism of tree search using playout and a heuristic evaluation function for evaluating the nodes.

was used in the experiments. Researchers can easily implement new algorithms and integrate them into this simulator and use it as a test bed for their research. The aforementioned Portfolio Greedy Search, which uses playouts together with LTD2, henceforth called the baseline method, to evaluate a game state, is also available in this simulator and used in our experiments. It has been shown in [10] that using the baseline method gives much better performance than using only LTD2. We thus replaced LTD2 by the proposed fuzzy-based evaluation method and compared it with the baseline method (Fig. 3).

5.1 Direct Comparison of Evaluation Methods

In this experiment, we directly compared the proposed method with the baseline method. The settings used for Portfolio Greedy Search (Algorithm 1) were as follows:

- Time limit per search episode: 10ms
- Improvement iteration I: 1
- Response iterations R: 0
- Portfolio P: (NOKAV [*2], Kiting [*3])
- Initial enemy script: NOKAV

[*2] Following this script, a unit will attack the enemy unit with the highest attack-value defined by *damage per frame / hit points* within its attack range. However, in order not to do any over-kill, it will not attack enemy units which have already been targeted and can be destroyed by other friendly units this round. It will instead choose the next highest priority target, or wait if such a target does not exist.

[*3] Following this script, a unit will attack the closest enemy unit within its attack range. When it is reloading and unable to attack, it will move away from that enemy unit.

Table 1 Combat scenarios used in the experiments.

Scenario	No. of unit types	Unit types
1	1	Zealot
2		Dragoon
3		Dark Templar
4		Archon
5	2	Zealot, Dragoon
6		Zealot, Dark Templar
7		Zealot, Archon
8		Dragoon, Dark Templar
9		Dragoon, Archon
10		Dark Templar, Archon
11	3	Zealot, Dragoon, Dark Templar
12		Zealot, Dragoon, Archon
13		Zealot, Dark Templar, Archon
14		Dragoon, Dark Templar, Archon
15	4	Zealot, Dragoon, Dark Templar, Archon

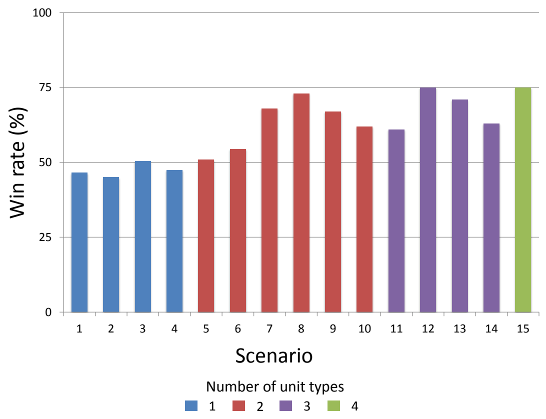

Fig. 4 Win rates of the proposed method against the baseline method when applied in Portfolio Greedy Search. Average win rate: 61%.

- Script used for playouts: NOKAV.

The first setting was made considering the fact that in practice the game runs at 24 frames per second, i.e., 42ms for each frame, and there are several modules for handling different tasks in a StarCraft AI bot. Thus, 10ms given to an essential module like unit micromanagement each frame is considered as a reasonable time limit. Meanwhile, the other settings were based on the recipe in [11].

The experiment consisted of a series of combat scenarios, in which two players control similar armies. Each scenario initially had 24 Protoss units of 4 types available in the simulator SparCraft, with the number of units in each type being randomly chosen in range $[1, 23]$ subject to having in total 24 units each side. All 15 available combinations (Table 1), each corresponding to a combat scenario, were tested. This setting enabled us to examine the effectiveness of the proposed fuzzy-based method in handling situations that have collaboration among different kinds of units. At the beginning of each battle, two forces are placed separately in a random fashion but symmetrically around the vertical line running through center of the map, as done in the previous work [11].

100 games were played for each combat scenario, giving 1500 total games. The performance of the proposed method against the baseline method is presented in Fig. 4. As can be seen, there were no significant differences between the two methods in simple combat scenarios. This can be explained by the fact that

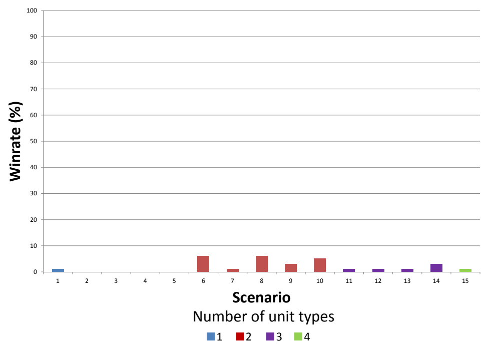

Fig. 5 Win rates of the evaluation method using the original function against the baseline method when applied in Portfolio Greedy Search. Average win rate: 2%.

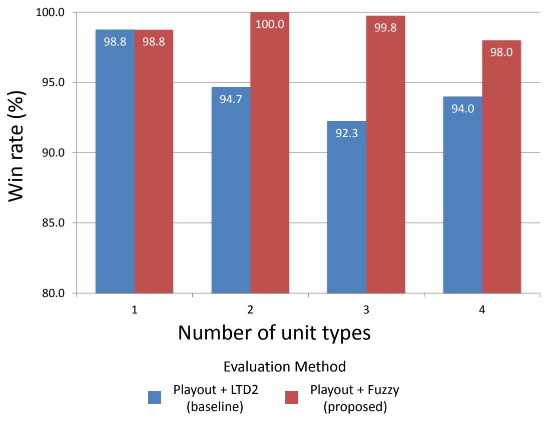

Fig. 6 Win rates of Portfolio Greedy Search against the script NOKAV.

there was only 1 unit type in scenarios 1–4, and therefore both LTD2 formula and fuzzy integral could readily evaluate game states as accurately as their counterparts. However, when combat scenarios became more complicated since more unit types were involved and interacted with each other, the proposed method outperformed the baseline method. In this case, the proposed method achieved a win rate above 50%, i.e., it helped improve the performance of Portfolio Greedy Search.

Furthermore, to prove that our proposed $f(x)$ function is more effective than the original function (5), we show in Fig. 5 the performance of the latter function against the baseline method with the same scenarios above. This figure indicates that the win rates of the original function against the baseline method are only around 2% on average. This can be explained as since the properties of a unit (in this case, the current hit points) are not considered in the original function, its playout evaluation is not accurate; for example, as long as both sides have a same unit combination, evaluation using the original function will always return “tie” regardless of each side units’ hit points.

5.2 Search vs. Script

We also performed an indirect comparison of the two methods by evaluating them against NOKAV (No-OverKill-Attack-Value). NOKAV is a common script used in many StarCraft bots and has been shown to be the most effective script so far; its outline is given in a footnote in Section 5.1.

This experiment had the same combat scenario setup as used in the previous one, i.e., two players were made to control similar armies to assure fairness. With the same setup in Section 5.1, each scenario was played 100 times, giving 400, 600, 400, and 100 games in total for the scenarios with one, two, three, and four types of units involved, respectively. As shown in Fig. 6, the search-based AIs (both the baseline method and the proposed method) achieved dominance over the scripted AI, winning over 90% of battles. This confirms the fact that script-based approaches often lack foresight and do not deal well with highly dynamic and real-time aspects in RTS games as compared to search techniques. It can also be seen that Portfolio Greedy Search using the proposed evaluation method obtained better performances than the original one, which is similar to the result in Section 5.1.

6. Conclusion and Future Work

In this paper, we presented a new approach for constructing an evaluation function that is essential in most search algorithms when applied to computer games. We learned the fuzzy measure from real game data and used it to calculate the order-based fuzzy integral as an estimation for the value of a game state. We carried out experiments with various settings to evaluate the proposed method and the results were encouraging. We were successful in improving the existing heuristic search and achieved a better quality of unit micromanagement in SparCraft, a simulator the popular RTS game StarCraft.

Our next step is to apply the heuristic search that uses the proposed evaluation method to a future version of our StarCraft bot—ICEbot [*4] and test it on the real game. We plan to collect more replay data and perform clustering of them in order to obtain better fuzzy measures that can handle more scenarios seen in the game. In addition, it is also our intention to combine the proposed method in this paper with one of our previous works that utilizes potential flow for positioning combat units [21], and aim for a more human-like StarCraft agent.

References

[1] Chung, M., Buro, M. and Schaeffer, J.: Monte Carlo planning in RTS games, *2005 IEEE Symposium on Computational Intelligence and Games*, Essex, UK, pp. 1–8 (2005).

[2] Churchill, D. and Buro, M.: Incorporating Search Algorithms into RTS Game Agents, *The Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, California, pp. 1–7 (2012).

[3] Li, Y.J.: Integrating genetic algorithm and fuzzy integral for evaluating game units combination in RTS game, M.Phil., Dept. of Computing, The Hong Kong Polytechnic University (2012).

[4] Ontanon, S.: The combinatorial multi-armed bandit problem and its application to real-time strategy games, *Proc. The Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Massachusetts, pp. 58–64 (2013).

[5] Sailer, F., Buro, M. and Lanctot, M.: Adversarial planning through strategy simulation, *2007 IEEE Symposium on Computational Intelligence and Games*, Honolulu, HI, pp. 80–87 (2007).

[6] Sharma, M., Holmes, M., Santamaria, J., Irani, A., Isbell, C. and Ram, A.: Transfer learning in real-time strategy games using hybrid CBR/RL, *International Joint Conference on Artificial Intelligence*, Hyderabad, India, pp. 1041–1046 (2007).

[*4] ICEbot has participated in Student StarCraft AI Tournament (SSCAI [20]) since 2012. It was ranked 1st and 2nd in Mixed division of SSCAI 2012 and SSCAI 2013 respectively.

[7] Synnaeve, G. and Bessiere, P.: A Bayesian model for opening prediction in RTS games with application to StarCraft, *2011 IEEE Conference on Computational Intelligence and Games*, Seoul, pp. 281–288 (2011).
[8] Weber, B.G., Mateas, M. and Jhala, A.: Building human-level AI for real-time strategy games, *Proc. The AAAI Fall Symposium*, Virginia, pp. 329–336 (2011).
[9] Wender, S. and Watson, I.: Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft:Broodwar, *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, Granada, pp. 402–408 (2012).
[10] Churchill, D., Saffidine, A., and Buro, M.: Fast heuristic search for RTS game combat scenarios, *Proc. The Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, California, pp. 112–117 (2012).
[11] Churchill, D. and Buro, M., Portfolio greedy search and simulation for large-scale combat in StarCraft, *2013 IEEE Conference on Computational Intelligence in Games*, Niagara Falls, ON, pp. 1–8 (2013).
[12] Wang, Z., Nguyen, K.Q., Thawonmas, R. and Rinaldo, F.: Monte-Carlo planning for unit control in StarCraft, *2012 IEEE 1st Global Conference on Consumer Electronics*, Tokyo, pp. 263–264 (2012).
[13] Stanescu,M., Hernandez,S.P., Erickson, G., Greiner R. and and Buro M.: Predicting Army Combat Outcomes in StarCraft *Ninth Artificial Intelligence and Interactive Digital Entertainment*, Massachusetts, pp. 86–92 (2013).
[14] Wang, Z. and Klir, G.J.: Fuzzy Measure Theory, Springer (1992).
[15] Lodwick, W.A. and Kacprzyk, J.: Fuzzy Optimization: Recent Advances and Applications, Springer (2010).
[16] Li, Y.J., Ng, P.H.F., Wang, H.B., Shiu, S.C.K. and Li,Y.: Apply different fuzzy integrals in unit selection problem of real time strategy game, *2011 IEEE International Conference on Fuzzy Systems*, Taipei, pp. 170–177 (2011).
[17] Heinermann, A.: BWAPI—An API for interacting with StarCraft: BroodWar (online),
available from ⟨http://code.google.com/p/bwapi/⟩ (accessed 2014-2-3).
[18] Mitchell, M.: An Introduction to Genetic Algorithms, MIT Press (1998).
[19] Churchill, D.: SparCraft—StarCraft Combat Simulation (online),
available from ⟨http://code.google.com/p/sparcraft/⟩ (accessed 2014-2-3).
[20] [SSCAI] Student StarCraft AI Tournament (online),
available from ⟨http://sscaitournament.com/⟩ (accessed 2014-2-3).
[21] Nguyen, T.D., Nguyen, K.Q. and Thawonmas, R.: Potential flow for unit positioning during combat in StarCraft, *2013 IEEE 2nd Global Conference on Consumer Electronics*, Tokyo, pp. 10–11 (2013).

Tung Duc Nguyen received his B.Eng. degree in human and computer intelligence from Ritsumeikan University, Shiga, in Japan in March 2014. His research interests include machine learning, game theory and artificial intelligence applied to computer games. During his undergraduate study, Mr. Nguyen was a member of the Intelligent Computer Entertainment Laboratory and a recipient of the Higher Education Development Support Project on ICT (HEDSPI) Scholarship as well as that of the IEEE GCCE 2013 Outstanding Student Paper Award. Since April 2014, he has joined Framgia Vietnam CO., LTD in Vietnam.

Kien Quang Nguyen received his B.Eng. degree in human and computer intelligence from Ritsumeikan University, Shiga, in Japan in March 2012. After a six-month working experience in Vietnam, he studied for and received his M.Eng. degree in human information science in September 2014 at the same university. As a member of the Intelligent Computer Entertainment Laboratory, Mr. Nguyen is the principal developer of the winning ghost team at the 2011 IEEE Congress on Evolutionary Computation (CEC) Ms. Pac-Man Versus Ghosts Competition and one of the main developers of the winning StarCraft bot in 2012 Student StarCraft AI Tournament - Mixed Division.
He was a recipient of the Human Higher Education Development Support Project on ICT (HEDSPI) Scholarship during his undergraduate study and a recipient of the Japanese Government scholarship during 2012-2014. His research interests include machine learning, game theory, neural networks, and evolutionary algorithms. He is now with Microsoft Development Ltd. in Japan.

Ruck Thawonmas received the B.Eng. degree in electrical engineering from Chulalongkorn University, Bangkok, in Thailand in 1987, the M.Eng. degree in information science from Ibaraki University, Ibaraki, in Japan in 1990, and the D.Eng. degree in information engineering from Tohoku University, Miyagi, in Japan in 1994. Before joining Ritsumeikan University, Shiga, in Japan in April 2002, he had worked at various institutions: Hitachi, Ltd.; RIKEN; University of Aizu; and Kochi University of Technology. Since April 2004, he has been a Full Professor at the Department of Human and Computer Intelligence where he leads the Intelligent Computer Entertainment Laboratory. His research interests include game AI, automatic comic generation, and player-behavior analysis.
Dr. Thawonmas was a recipient of the Japanese Government Scholarship during 1987-1993. His laboratory has won a number of game AI competitions: the winning controllers at the 2009 IEEE Congress on Evolutionary Computation (CEC) and the 2009 IEEE Conference on Computational Intelligence and Games (CIG) Ms. Pac-Man Competitions (screen-capture version), the winning ghost team at the 2011 IEEE CEC Ms. Pac-Man Versus Ghosts Competition, the winning human bot and judge bot at the 2011 BotPrize (at the 2011 IEEE CIG), and the winning Ms. Pac-Man controller at the latest Ms. Pac-Man Versus Ghosts Competition.