# Multi-Objective Tree Search Approaches for General Video Game Playing

Diego Perez-Liebana
School of Computer Science
and Electronic Engineering
University of Essex, UK
dperez@essex.ac.uk

Sanaz Mostaghim
Faculty of Computer Science
Otto von Guericke University
Magdeburg, Germany
sanaz.mostaghim@ogvu.de

Simon M. Lucas
School of Computer Science
and Electronic Engineering
University of Essex, UK
sml@essex.ac.uk

***Abstract*—The design of algorithms for Game AI agents usually focuses on the single objective of winning, or maximizing a given score. Even if the heuristic that guides the search (for reinforcement learning or evolutionary approaches) is composed of several factors, these typically provide a single numeric value (reward or fitness, respectively) to be optimized. Multi-Objective approaches are an alternative concept to face these problems, as they try to optimize several objectives, often contradictory, at the same time. This paper proposes for the first time a study of Multi-Objective approaches for General Video Game playing, where the game to be played is not known a priori by the agent. The experimental study described here compares several algorithms in this setting, and the results suggest that Multi-Objective approaches can perform even better than their single-objective counterparts.**

## I. Introduction

Creating Artificial Intelligence (AI) agents for games is usually tackled from the perspective of a bot trying to excel in one particular objective - typically, to achieve victory. Even if the objective of the agent is also to maximize score, it is often the case that an heuristic is designed for the bot to aim to accomplish both goals simultaneously. Other games may also require some additional objectives, such as exploring the game area, considering time elements or collecting power-ups that enhance the player abilities. Even in this case, a carefully designed heuristic, tailored to the specifics of the game at stake, may provide a single value that, if optimized, may lead to an optimal behaviour. This approach is taken in most research and algorithms used for game AI.

An alternative to this kind of approaches is to consider the game as a Multi-Objective (MO) problem. By doing this, the agent tries to find an equilibrium between two or more objectives to optimize, typically in opposition. For instance, as seen later in this paper, the agent may try to maximize both the score achieved and the areas of the level that are explored. According to some policy, the agent takes actions in order to reach states of the games where both objectives are maximized. Previous research shows that MO approaches can outperform significantly single-objective algorithms in specific games [1].

The value of considering multiple objectives at once can be even more relevant when the same agent is meant to play many different games, which rules and mechanics are unknown a priori. It seems reasonable to define a single objective heuristic function specifically designed to guide the agent in a known game, but it is fair to ask if MO approaches are better suited for unknown games, where it is unclear which objectives may have more or less importance.

The idea of creating agents able to play in multiple games is not new, although it is becoming more and more popular lately after the advances in frameworks like the Atari simulator ALE (Arcade Learning Environment; [2]) or the General Video Game AI (GVGAI) framework and competitions [3]. General Video Game aims to propose a challenge for algorithms capable of playing any game, even if it has not been seen before. The reasoning behind this is to make algorithms independent from specific domains, in order to perform research that can be traversal across different games. Also, it is possible to consider that playing any number of unseen games is closer to the problem of artificial general intelligence, as opposed to AI concerned with solving individual problems.

The goal of this paper is to shed some light into the performance of MO approaches in GVGAI, analyzing the performance of previous and new MO algorithms, and comparing them with other single-objective agents. This paper is organized as follows: Section II describes the GVGAI framework and games, used in the experiments performed for this research. Section III provides a background on the algorithms that form the basis of the agents employed on the experiments, which are described in Section IV. Experiments and results are discussed later in Section V, and conclusions and future work outlined in Section VI.

## II. The General Video Game AI Framework

This section describes the framework and games employed for the experimental part of this study. The GVGAI framework is a Java port of the original *py-vgdl*, a Python version implemented by Tom Schaul [4] as a benchmark for learning and planning problems.

One of the objectives of this framework is to allow an easy creation of new games and levels. Therefore, all these are described, in plain text, in a high-level Video Game Definition Language (VGDL). VGDL allows the definition of 2D real-time single-player arcade-type games, where the player can control an avatar and interact with objects located around the level. By means of an ontology, this language

permits the creation of multiple games. At the time of writing of this paper, 80 different games have been designed, forming 8 groups of 10 games each.

The GVGAI framework is able to read VGDL games and expose an interface to the agent, allowing queries about the game state (score, victory state, time steps), the avatar (position, resources collected, list of available actions) and other sprites in the game. It is important to highlight that the VGDL descriptions of the game and level are not given to the controller, which can only access game information through Java objects. Furthermore, the information about the nature of the sprites is hidden, so it is not possible to know in advance the utility or purpose of the other elements of the game.

The agent is provided, however, with a forward model, so it can simulate the effects of taking certain actions and reach the subsequent states. The games are, in general, of stochastic nature, so it is worth highlighting that multiple calls to the forward model from the same state and action may provide different next states. The controller can make use of this forward model to decide the next action to take in the game, as long as it does not exceed the $40ms$ of time budget allocated to each move. In case of an overspent, no action is executed on the next move. This framework and settings were used in the GVGAI competitions run in 2014 and 2015. For more details, the reader is referred to the 2014 GVGAI competition paper [3].

The games employed for this research are those of the first set, described in detail in Table I. It is worth highlighting the differences in the nature of these games. There are different ways to achieve victory on them (reaching an exit, capturing certain sprites, placing objects in specific locations, etc.), and the score schemes vary importantly from one game to the other. Not even all games share the same set of available actions (while *Aliens* allows the avatar to move sideways and shooting, *Butterflies* only allows movement in all directions).

## III. Background

This section provides an overview of the techniques behind the agents (or *controllers*) employed in the experiments of this research study.

### A. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [5] is a reinforcement learning approach that has become very popular in the last decade due to its performance in the game of Go [6]. Since then, MCTS has been applied to diverse games and scenarios, and multiple variants and modifications have been proposed in the literature [7]. MCTS has achieved good results in many domains, including General Game (and Video Game) Playing, and it is the first algorithm (in combination with Deep Neural Networks) able to defeat a professional Go player in a complete $19 \times 19$ board, unbiased match [8].

MCTS is a tree based search algorithm that builds an asymmetric tree in memory using a forward model of the game. The algorithm adds one node at each iteration, estimating its game-theoretic value by using self-play from the state of the node to the end of the game or a given depth. Each node stores statistics regarding the number of time it is visited ($N(s)$), the number of times an action is taken from that node ($N(s,a)$) and the empirical average of the rewards obtained when picking an action $a$ from $s$ ($Q(s,a)$).

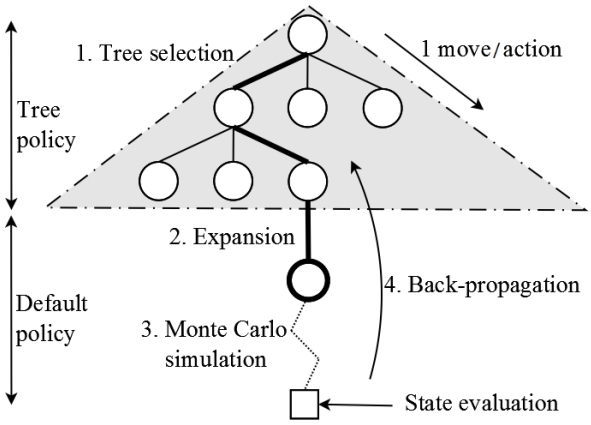


Fig. 1. MCTS algorithm steps

Figure 1 depicts the 4 steps of the MCTS algorithm. During the first step, *tree selection*, the tree is navigated from the root, using a *Tree Policy*, until reaching a node that is not totally expanded (i.e. there is at least one action that has not been picked from that state). The *expansion* step adds a new node to the tree, from which the *Monte Carlo simulation* (or roll-out) is performed, following the *Default Policy* until the end of the game or some specified depth. At that point, the state reached is evaluated with some heuristic function, and the result is *back-propagated* to all visited nodes in the tree during that iteration. This process is repeated until a time budget or a maximum number of iterations is consumed. The action to play in the real game is determined by a *Recommendation Policy*.

One of the most popular tree policies is Upper Confidence Bounds (UCB1; see Equation 1). This policy balances between exploitation (first term of UCB1) and exploration (second term) at every step in the tree selection phase, in order to explore the most promising states of the search space. This balance can be tempered by the value of $C$: setting high values gives priority to exploration, while values closer to 0 reward those actions $a \in A(s)$ with a higher expected reward.

$$a^* = \underset{a \in A(s)}{\arg\max} \left\{ Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \right\} \qquad (1)$$

The default policy can select actions uniformly at random, or biased by features of the states visited. Finally, the recommendation policy can choose an action for playing in the real game based on different metrics, such as $Q(s,a)$ or $N(s,a)$.

### B. Multi-objective Optimization

A Multi-objective Optimization Problem (MOP) refers to a scenario where two or more conflicting functions need to

| Game | Description | Score |
|---|---|---|
| **Aliens** | Similar to traditional Space Invaders, Aliens features the player (avatar) in the bottom of the screen, shooting upwards at aliens that approach Earth, who also shoot back at the avatar. The player loses if any alien touches it, and wins if all aliens are eliminated. | • 1 point is awarded for each alien or protective structure destroyed by the avatar.<br>• $-1$ point is given if the player is hit. |
| **Boulderdash** | The avatar must dig in a cave to find at least 10 diamonds, with the aid of a shovel, before exiting through a door. Some heavy rocks may fall while digging, killing the player if it is hit from above. There are enemies in the cave that might kill the player, but if two different enemies collide, a new diamond is spawned. | • 2 points are awarded for each diamond collected, and 1 point every time a new diamond is spawned.<br>• $-1$ point is given if the avatar is killed by a rock or an enemy. |
| **Butterflies** | The avatar must capture butterflies that move randomly around the level. If a butterfly touches a cocoon, more butterflies are spawned. The player wins if it collects all butterflies, but loses if all cocoons are opened. | • 2 points are awarded for each butterfly captured. |
| **Chase** | The avatar must chase and kill scared goats that flee from the player. If a goat finds another goat's corpse, it becomes angry and chases the player. The player wins if all scared goats are dead, but it loses if is hit by an angry goat. | • 1 point for killing a goat.<br>• $-1$ point for being hit by an angry goat. |
| **Frogs** | The avatar is a frog that must cross a road, full of tracks, and a river, only traversable by logs, to reach a goal. The player wins if the goal is reached, but loses if it is hit by a truck or falls into the water. | • 1 point for reaching the goal.<br>• $-2$ points for being hit by a truck. |
| **Missile Command** | The avatar must shoot at several missiles that fall from the sky, before they reach the cities they are directed towards. The player wins if it is able to save at least one city, and loses if all cities are hit. | • 2 points are given for destroying a missile.<br>• $-1$ point for each city hit. |
| **Portals** | The avatar must find the goal while avoiding lasers that kill him. There are many portals that teleport the player from one location to another. The player wins if the goal is reached, and loses if killed by a laser. | • 1 point is given for reaching the goal. |
| **Sokoban** | The avatar must push boxes so they fall into holes. The player wins if all boxes are made to disappear, and loses when the timer runs out. | • 1 point is given for each box pushed into a hole. |
| **Survive Zombies** | The avatar must stay alive while being attacked by spawned zombies. It may collect honey, dropped by bees, in order to avoid being killed by zombies. The player wins if the timer runs out, and loses if hit by a zombie while having no honey (otherwise, the zombie dies). | • 1 point is given for collecting one piece of honey, and also for killing a zombie.<br>• $-1$ point if the avatar is killed, or it falls into the zombie spawn point. |
| **Zelda** | The avatar must find a key in a maze to open a door and exit. The player is also equipped with a sword to kill enemies existing in the maze. The player wins if it exits the maze, and loses if it is hit by an enemy. | • 2 points for killing an enemy, 1 for collecting the key, and another point for reaching the door with it.<br>• $-1$ point if the avatar is killed. |

TABLE I
SET OF 10 GAMES USED FOR THE EXPERIMENTS OF THIS PAPER.

be optimized simultaneously. It can be defined as:

$$optimize \quad \{f_1(\vec{x}), f_2(\vec{x}), \cdots, f_m(\vec{x})\} \tag{2}$$

subject to $\vec{x} \in \Omega$, involving $m(\geq 2)$ conflicting objective functions $f_i : \Re^n \rightarrow \Re$. $\vec{x} = (x_1, x_2, \cdots, x_n)^T$ are *decision vectors* from the feasible region $\Omega \subset \Re^n$ and $Z \subset \Re^m$ is the feasible objective region, which elements are called *objective vectors* and consist of $m$ objective (function) values $\vec{f}(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), \cdots, f_m(\vec{x}))$. Then, each solution $\vec{x}$ results in a set of $m$ different scores to be optimized. A solution $\vec{x}$ *dominates* another solution $\vec{y}$ if and only if:

1) $f_i(\vec{x})$ is not worse than $f_i(\vec{y})$, $\forall i = 1, 2, \ldots, m$.
2) For at least one objective $j$: $f_j(\vec{x})$ is better than its analogous counterpart in $f_j(\vec{y})$.

When both conditions are met, it is said that $\vec{x} \prec \vec{y}$ ($\vec{x}$ dominates $\vec{y}$), a condition that provides partial ordering between solutions in the objective space. In those cases where it is not possible to say that $\vec{x} \prec \vec{y}$ or $\vec{y} \prec \vec{x}$., it is said that these solutions are indifferent to each other and form part of the same *non-dominated set*. Given a non-dominated set $P$, it is said that $P$ is the *Pareto-set* if there is no other solution in the decision space that dominates any member of $P$, and their objective vectors build a *Pareto-front*.

Multiple MO approaches can be found in the literature, including evolutionary and reinforcement learning approaches [9], [10]. The weighted-sum approach is known to be one of the traditional methods, which provides a set of weights to balance between objectives according to the user preference. The resulting weighted sum leads to one objective function to be optimized and, depending on the weights provided, the algorithm can converge to different solutions, ideally on a Pareto-front. However, this linear scalarization approach fails to find optimal solutions for problems with non-convex-shape Pareto-fronts [9].

Another important concept in the MO literature relates to how to measure the quality of a non-dominated set. A popular metric for this is the Hypervolume Indicator (HV), which measures both the diversity and convergence of non-dominated solutions [11]. The HV of a Pareto front, $HV(P)$, is defined as the volume of the objective space dominated by $P$. Assuming the objectives are to be maximized, the higher the value of $HV(P)$, the better the front obtained.

**Algorithm 1** Pareto MO-MCTS node update [1].

```
1: function UPDATE(node, r̄, dominated = false)
2:     node.Visits = node.Visits + 1
3:     node.R̄ = node.R̄ + r̄
4:     if !dominated then
5:         if node.P ≺ r̄ then
6:             dominated = true
7:         else
8:             node.P = node.P ∪ r̄
9:     UPDATE(node.parent, r̄, dominated)
```

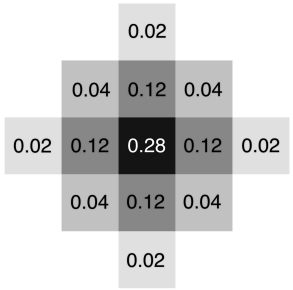


Fig. 2. Pheromone diffusion.

### C. Multi-objective MCTS

The first Multi-Objective Monte Carlo Tree Search (MO-MCTS) for real-time games, developed by D. Perez-Liebana et al. [1], tackles the problem of selecting an action with a reduced time budget in an MO setting. This algorithm adapts the traditional MCTS to deal with multiple objectives when evaluating a game state.

This algorithm requires that a game state evaluation function provides fitness for $m$ objectives, as a vector $\overline{r}$. $\overline{r}$ is back-propagated through all the nodes visited in the tree, as shown in the fourth step of MCTS, to update the accumulated reward vector $\overline{R}$. In MO-MCTS, each node in the tree keeps a local Pareto front approximation $P$, which is updated at each iteration with the reward vector $\overline{r}$. Algorithm 1 describes how the node statistics are updated in MO-MCTS.

The update of each local Pareto front $P$ (line 8 of Algorithm 1) works as follows: if $\overline{r}$ is not dominated by the front, it is added to $P$, considering that any (or all) points in $P$ can be dominated by $\overline{r}$ and therefore leave the set. In the case that $\overline{r}$ is dominated by $P$, the front remains unchanged, and the same occurs for the rest of the nodes until reaching the root.

By keeping these fronts, each node has an estimation of the quality of the states that can be reached from that point. Also, the quality of this estimation can be measured by calculating the HV of the front $P$ of the node. Following this reasoning, it is straightforward to see that the front of the root of the tree contains the best non-dominated front of the whole search.

The algorithm also provides information for the tree and recommendation policies. In the former case, it is possible to substitute the value of $Q(s, a)$ (which in the original algorithm refers to single rewards) from Equation 1 with the Hypervolume measure of the front, $HV(P)$. In the latter, it is possible to store information on the root node as to which action leads to each point in its non-dominated front. By providing weights to the different objectives, it is possible to determine which point in $P$ is selected and therefore pick the action that leads to it.

## IV. Approaches

This section describes in detail the different controllers employed in this study. First, Section IV-A defines the different heuristics that are used by the agents to conform the objectives. Then, Section IV-B specifies how the agents use these objectives differently to play the games described in Table I.

### A. Heuristics

*1) Score (Objective $O_1$):* This heuristic simply takes the score of the present game state, but a high integer is added if the game is won or lost ($10^6$ or $-10^6$, respectively). This heuristic is used by the sample MCTS, a controller provided with the framework, and included in the experiments for comparison. Albeit simple, this sample controller provides a reasonable performance: it finished in third position in the 2014 GVGAI Competition (18 qualified controllers) and in the mid table in the legs of the 2015 edition (close to 50 controllers).

*2) Level Exploration (Objective $O_2$):* The second heuristic consists on a value that, if maximized, aids the agent to maximize the number of positions (or grid cells) explored in the level. It is a nature-inspired technique based on *pheromone* trails, and its complete description can be found in [12]. In short, this mechanism works by simulating pheromones expelled by the avatar at each game tick, which spread into the neighbouring area. Each cell contains a pheromone value $p_{ij} \in [0, 1]$, where $i$ and $j$ are coordinates in the level board. $p_{ij}$ decays with time, and the change of pheromone $p_{ij}$ from one step to the next is given by the Equation 3:

$$p_{i,j} = \rho_{df} \times \rho_{\phi} + (1 - \rho_{df}) \times \rho_{dc} \times p_{i,j} \qquad (3)$$

$\rho_{\phi}$ is calculated as the sum of pheromone trail in all neighbouring cells divided by the number of neighbouring cells (note that an edge cell reduces the number of neighbours). $\rho_{df} \in (0, 1)$ sets the value of diffusion of the pheromone and, finally, $\rho_{dc} \in (0, 1)$ establishes the decay of the value at each game cycle. These values are set to $\rho_{df} = 0.4$ and $\rho_{dc} = 0.99$, as they have shown good results in a previous study [12]. Figure 2 shows an example of the decay of the pheromone trail from one step to the next.

This algorithm produces high values of pheromone trail in the close proximity of the avatar, as well as in positions recently visited. In order to maximize exploration of the level and movement through it, the heuristic should prefer positions where this value is small. Therefore, the value of the heuristic is calculated as $O_2 = 1 - p_{i,j}$, where $i$ and $j$ are

the coordinates of the avatar in the grid in a particular state. Nevertheless, using this objective as defined here, would not reward appropriately game states where the agent wins or loses the game. Therefore, as in the previous agent, a high positive or negative number is added to $1 - p_{i,j}$ to weight this factor in.

### B. Agents

Although the four agents presented here differ in the way the heuristics are composed to give a value of the quality of a given state, the algorithms share some common settings for a fair comparison. As they are all MCTS approaches, they need a value for the exploration-exploitation constant $C$, which is set to $\sqrt{2}$, a value that has given good results in the past for single player games, as the types of games in the framework.

Additionally, instead of using a time budget for each game tick, a limit of $50$ rollouts per decision was imposed for all algorithms[1]. This measure assures that all algorithms explore approximately the same amount of states and there is no dependency on the overload of the server where the experiments are run. Finally, each rollout was set to a limit of $10$ actions from the root.

*1) Sample MCTS:* This agent is sample MCTS controller supplied with the GVGAI framework. Therefore, the value function for the state is exactly the value of $O_1$.

*2) Weighted Sum MCTS:* This agent uses a single objective MCTS, which value function is the weighed sum of $O_1$ and $O_2$: $O_1 \times \alpha + O_2 \times \beta$. The two weights, $\alpha$ and $\beta$, are both set to $0.5$. This is just one possible, balanced value that has been chosen for this algorithm, but also for the next two agents when a decision between objectives needs to be made. An interesting line of further investigation could be to analyze changes of performance with different weights, although this was left out of scope for this study.

*3) Mixed Strategy MCTS:* This is a new algorithm proposed for this study. The idea is to tackle the Multi-objective problem as a mixed strategy [13], where each objective is managed by a different decision maker. In practical terms, this agent has two different value functions and, at the beginning of the game tick, a high level policy determines which value function should be used during the decision time for that move.

In this agent, the agent may choose to use a MCTS that maximizes the score (using $O_1$) as value function, or one that maximizes exploration ($O_2$ as heuristic). In the experiments performed for this study, and as indicated for the previous agent, the selection of one or another tree is uniformly random (probability of $0.5$ for each).

*4) MO-MCTS:* This agent uses the MO-MCTS algorithm described in Section III-C, using the two objectives $O_1$ and $O_2$. When the algorithm has finished its allotted number of iterations, a weighed sum $O_1 \times \alpha + O_2 \times \beta$ is calculated (as before, with both weights set to $0.5$) for each member of the local Pareto front $P$ of the root. Then, the point with the highest value is selected, and the action that leads to this point is picked for execution in the real game.

[1]This value is, on average across games, similar to the number of iterations that Sample MCTS performs in $40ms$.

| | **Sample MCTS** | **Weighted Sum MCTS** | **Mixed Strategy MCTS** | **MO-MCTS** |
|---|---|---|---|---|
| **% Victories** | 32.24 (0.67) | 29.80 (0.66) | 30.51 (0.66) | **42.38 (0.70)** |

TABLE II
PERCENTAGE OF VICTORIES ACHIEVED ACROSS ALL GAMES IN THE FIRST GVGAI COMPETITION SET.

## V. EXPERIMENTS

The experiments were conducted in the $10$ games shown in Table I, for each one of the four agents described in the previous section. Each game has been played in $5$ different levels, $100$ times each, adding up to $500$ plays per game and agent[2].

It is possible to analyze these results using two different metrics: percentage of victories achieved and score average at the end of the game. Regarding the percentage of victories, it is worth highlighting the best results obtained by the winners the two editions of the GVGAI competition[3]. The winner of the 2014 edition, *adrienctx*, achieved $51.2\%$ of victories in the test set [3], while the best entry of the 2015 legs obtained a final $45.8\%$ of victories across 3 test sets. This is, therefore, a far from solved problem where about half of the games played are lost.

Figure 3 show the percentage of victories of all agents in the $10$ games tested. As can be seen, the best two agents are Sample MCTS and MO-MCTS, always being one of these two controllers the one that achieves the highest percentage of victories on each game. MO-MCTS achieves a clearly higher result in $5$ games (*Aliens*, *Frogs*, *Missile Command*, *Portals* and *Zelda*).

Table II shows the percentage of victories of all agents across all games and, as can be seen, MO-MCTS achieves the highest percentage of victories across the algorithms compared. Sample MCTS is the second best algorithm in this ranking, with Weighted Sum and Mixed Strategy MCTS staying behind.

It is also interesting to analyze the results in some of the games in particular. For instance, *Aliens* is a game that has been traditionally well played by the Sample MCTS agent, achieving close to $100\%$ of victories on the $40ms$ per tick setting of the official competition, which allowed for an average close to $100$ iterations per game cycle. As Figure 3 shows, when the number of iterations is reduced to $50$, the performance of Sample MCTS drops, while MO-MCTS is still achieving $99,80\%(0.20)$ of victories. Another very impressive result is the one obtained in *Frogs*, which has been a game that has posed many problems to MCTS

[2]Due to a high computational cost, some games haven't reached $500$ games after 2 weeks of experiments. The number of repetitions is indicated in the first column of Table III

[3]www.gvgai.net

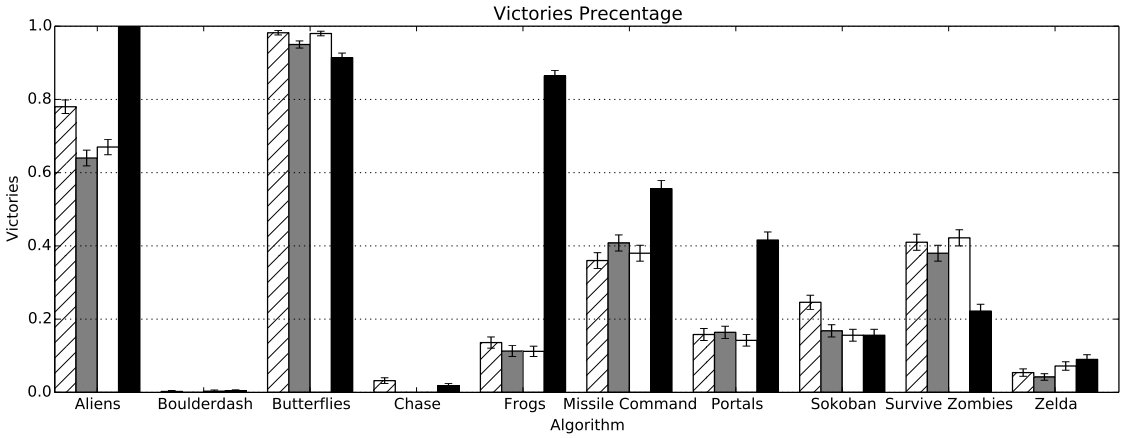


Fig. 3. Percentage of victories (with std. error). Four approaches are compared per game. From left to right: Sample MCTS, Weighted Sum MCTS, Mixed Strategy MCTS, MO-MCTS.

approaches. However, it seems that the MO-MCTS agent is able to use the exploration heuristic wisely to find the goal of the level in $86.40\%$ $(1.53)$ of the games played.

Not all games are favorable to MO-MCTS, however, and in some cases it is possible that the excessive exploration is actually a disadvantage. A good example could be *Survive Zombies*, where one of the best strategies is to locate a spot in the level safe from zombies. Exploring the level too much may lead to find more enemies and therefore to lose the game.

Regarding the score obtained by the different approaches, Figures 4 and 5 show the average of scores achieved on each game. Note that the range of typical scores varies considerably between different games. According to this metric, MO-MCTS obtains a higher score than the other algorithms in $7/10$ games, only beaten or matched by Sample MCTS on the rest.

The numerical results for victories and scores can be seen in Table III, which also includes standard errors and indication of those results that are statistically significant. As can be seen, MO-MCTS is significantly better in terms of scores than all approaches in 7 games (non parametric Wilcoxon signed-rank test with p-value $< 0.05$), and better than Weighted Sum and Mixed Strategy in the other 2 (*Sokoban* and *Chase*).

Another interesting result is the fact that MO-MCTS behaves clearly better, both in terms of victories and average of scores, than the other two multi-objective approaches. This seems to suggest that, although approaching general video game playing as a multi-objective problem seems to be a better choice in general, the way these objectives are employed in the heuristics and in the search algorithm are very relevant. In other words, Weighted-Sum and MO-MCTS use exactly the same expressions for both objectives (see Sections IV-B2 and IV-B3), but the former forms a single value by combining them with weights, while the latter uses both values to guide the search. It is very clear that the second approach works better in this setting.

Finally, it is also worth mentioning that the Mixed Strategy MCTS approach proposed here does not obtain results on a par with the best two algorithms tried in this study. Nevertheless, this does not mean that the results obtained are not interesting. This agent spends $50\%$ of its moves only considering new places to move to, without taking into account a maximization of the score. Note that this is the only agent that completely ignores the score in half of its moves: the other agents consider this value at every turn.

Taking this into account, it is worth noting, as seen in Table III, that Mixed Strategy MCTS is (significantly) better than Weighted-Sum MCTS in 2 games in terms of, precisely, score; and similar in performance in the other 8 (never worse). This could suggest that it is not necessary to keep a permanent focus on maximizing score, and that in some cases it could be even better to change targets completely. This result reinforces the idea of mixed strategies for GVG, and encourages further investigation on how to balance better (i.e. more dynamically) the different objectives while playing.

## VI. Conclusions

This paper compares, for the first time in the literature, several Multi-Objective approaches for General Video Game Playing, performing experiments in 10 games from the GVGAI framework. The algorithms proposed are the Sample (single-objective) MCTS, and three variants of MO approaches: weighted-sum, a mixed strategy and a Multi-Objective MCTS (MO-MCTS) that builds Pareto fronts in the nodes of the tree in order to take actions that lead to non-dominated solutions.

One of the most interesting results found is that the MO-MCTS algorithm obtains, on average, a higher percentage of victories considering all games together, and a higher average of scores than the other agents in most of them.

Additionally, it has been shown that the way several objectives are combined does affect the quality of the results. Even where two approaches (Weighted-Sum MCTS and MO-MCTS) were using the same expression for both objectives,

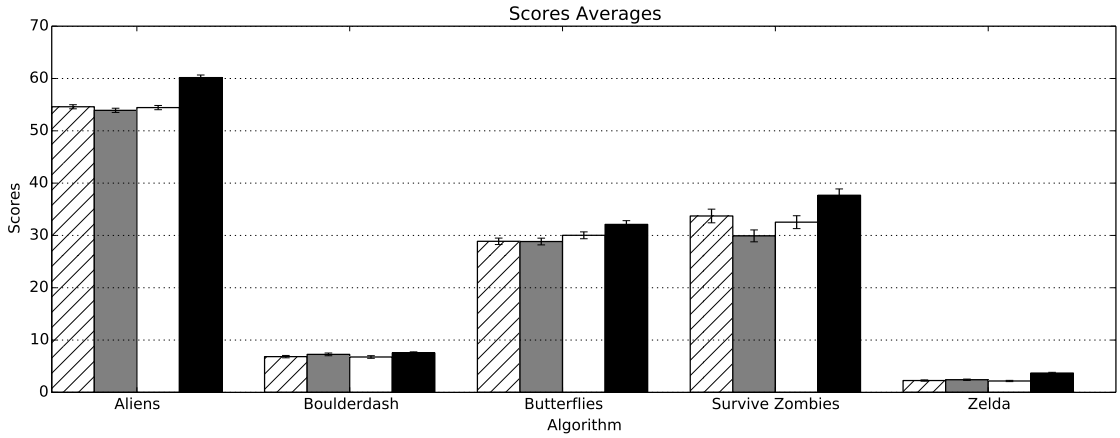


Fig. 4. Average of scores (with std. error). Four approaches are compared per game. From left to right: Sample MCTS, Weighted Sum MCTS, Mixed Strategy MCTS, MO-MCTS.

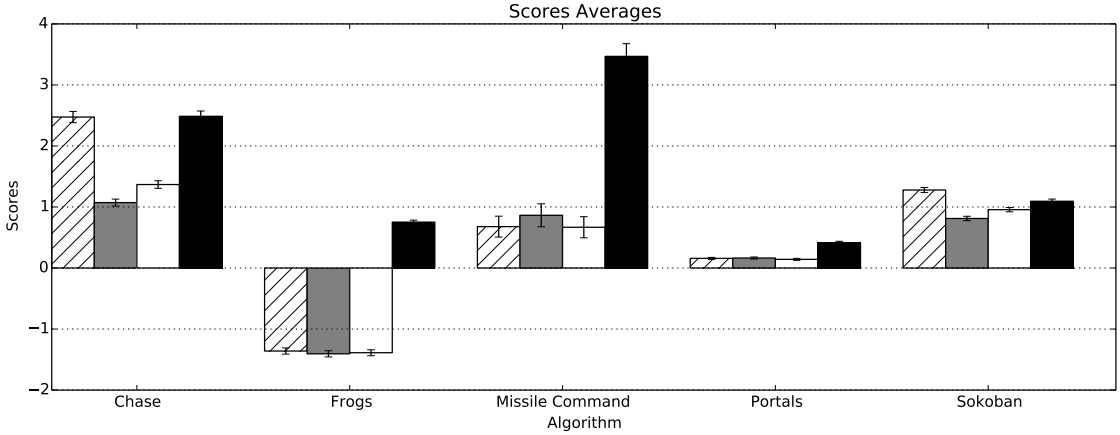


Fig. 5. Average of scores (with std. error). Four approaches are compared per game. From left to right: Sample MCTS, Weighted Sum MCTS, Mixed Strategy MCTS, MO-MCTS.

the latter performed much better because of the way these objectives are integrated into the nodes of the search tree. The Mixed Strategy approach did not achieve the same results that the best two agents found in this study did, but it was able to obtain slightly better results than the weighted sum agent with quite different approaches, suggesting that some further investigation is needed in this respect.

The work performed in this paper opens several lines of future work. First of all, it would be interesting to analyze how changing the weights in the recommendation policy affects the performance of MO-MCTS. Actually, varying these weights dynamically in response to changes in the environment seems to be a reasonable idea, both for MO-MCTS and Mixed-Strategy MCTS. This would, of course, open the challenge of finding objective adaptive heuristics for General Video Game Playing. In fact, it is worth investigating if a more involved selection of the heuristic that guides the search in Mixed-Strategy MCTS lies better results. For example, it would be fair to assume that a deeper exploration is beneficial at the beginning of the game, or when new areas of the levels are discovered.

Furthermore, other multi-Objective approaches can also be considered. An idea is to use the Epsilon-Constrained approach [14], that considers one of the objective functions as a constraint ($O_i < \epsilon$, in case of maximization) and only the other function is subject to optimization. The value of $\epsilon$ depends on the user-preferences and is a constant value. This would allow to establish limits for certain objectives, which could force the recommendation policy to select actions that lead to more adequate points in the Pareto front of the MO-MCTS tree root. Finally, the performance of other algorithms, like a Rolling-Horizon version of NSGA-II [1], could also be studied in general video game playing.

## REFERENCES

[1] D. Perez-Liebana, S. Mostaghim, S. Samothrakis, and S. Lucas, "Multi-Objective Monte Carlo Tree Search for Real-Time Games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7:4, pp. 347–360, 2014.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran,

| Game (Repetitions) | Algorithm | Victories (%) | Significantly better than ... | Scores | Significantly better than ... |
|---|---|---|---|---|---|
| **Aliens (500)** | A: Sample MCTS | 78.00 (1.85) | B , C | 54.60 (0.39) | B |
| | B: Weighted Sum MCTS | 64.00 (2.15) | Ø | 53.91 (0.40) | Ø |
| | C: Mixed Strategy MCTS | 67.00 (2.10) | Ø | 54.44 (0.40) | Ø |
| | **D: MO-MCTS** | 99.80 (0.20) | A , B , C | 60.17 (0.50) | A , B , C |
| **Boulderdash (311)** | A: Sample MCTS | 0.25 (0.25) | Ø | 6.83 (0.23) | Ø |
| | B: Weighted Sum MCTS | 0.00 (0.00) | Ø | 7.26 (0.26) | Ø |
| | C: Mixed Strategy MCTS | 0.32 (0.32) | Ø | 6.76 (0.25) | Ø |
| | D: MO-MCTS | 0.40 (0.28) | Ø | 7.53 (0.21) | Ø |
| **Butterflies (500)** | A: Sample MCTS | 98.20 (0.59) | B , D | 28.88 (0.62) | Ø |
| | B: Weighted Sum MCTS | 95.00 (0.97) | D | 28.84 (0.64) | Ø |
| | C: Mixed Strategy MCTS | 98.00 (0.63) | B , D | 30.02 (0.66) | Ø |
| | **D: MO-MCTS** | 91.40 (1.25) | Ø | 32.11 (0.72) | A , B , C |
| **Chase (500)** | A: Sample MCTS | 3.20 (0.79) | B , C | 2.47 (0.09) | B , C |
| | B: Weighted Sum MCTS | 0.00 (0.00) | Ø | 1.07 (0.06) | Ø |
| | C: Mixed Strategy MCTS | 0.00 (0.00) | Ø | 1.37 (0.06) | B |
| | D: MO-MCTS | 1.80 (0.59) | B , C | 2.49 (0.09) | B , C |
| **Frogs (461)** | A: Sample MCTS | 13.60 (1.53) | Ø | −1.36 (0.05) | Ø |
| | B: Weighted Sum MCTS | 11.28 (1.47) | Ø | −1.41 (0.05) | Ø |
| | C: Mixed Strategy MCTS | 11.20 (1.41) | Ø | −1.39 (0.05) | Ø |
| | **D: MO-MCTS** | 86.40 (1.53) | A , B , C | 0.75 (0.03) | A , B , C |
| **Missile Command (500)** | A: Sample MCTS | 36.00 (2.15) | Ø | 0.68 (0.17) | Ø |
| | B: Weighted Sum MCTS | 40.80 (2.20) | Ø | 0.86 (0.19) | Ø |
| | C: Mixed Strategy MCTS | 38.00 (2.17) | Ø | 0.67 (0.17) | Ø |
| | **D: MO-MCTS** | 55.60 (2.22) | A , B , C | 3.47 (0.21) | A , B , C |
| **Portals (500)** | A: Sample MCTS | 15.80 (1.63) | Ø | 0.16 (0.02) | Ø |
| | B: Weighted Sum MCTS | 16.40 (1.66) | Ø | 0.16 (0.02) | Ø |
| | C: Mixed Strategy MCTS | 14.20 (1.56) | Ø | 0.14 (0.02) | Ø |
| | **D: MO-MCTS** | 41.60 (2.20) | A , B , C | 0.42 (0.02) | A , B , C |
| **Sokoban (500)** | **A: Sample MCTS** | 24.60 (1.93) | B , C , D | 1.28 (0.04) | B , C , D |
| | B: Weighted Sum MCTS | 16.80 (1.67) | Ø | 0.81 (0.04) | Ø |
| | C: Mixed Strategy MCTS | 15.60 (1.62) | Ø | 0.96 (0.03) | B |
| | D: MO-MCTS | 15.60 (1.62) | Ø | 1.09 (0.04) | B , C |
| **Survive Zombies (500)** | A: Sample MCTS | 41.00 (2.20) | D | 33.72 (1.31) | B |
| | B: Weighted Sum MCTS | 38.00 (2.17) | D | 29.92 (1.14) | Ø |
| | C: Mixed Strategy MCTS | 42.20 (2.21) | D | 32.54 (1.23) | Ø |
| | **D: MO-MCTS** | 22.20 (1.86) | Ø | 37.68 (1.21) | A , B , C |
| **Zelda (500)** | A: Sample MCTS | 5.40 (1.01) | Ø | 2.26 (0.12) | Ø |
| | B: Weighted Sum MCTS | 4.20 (0.90) | Ø | 2.41 (0.13) | Ø |
| | C: Mixed Strategy MCTS | 7.20 (1.16) | B | 2.17 (0.12) | Ø |
| | **D: MO-MCTS** | 9.00 (1.28) | A , B | 3.69 (0.14) | A , B , C |

TABLE III

PERCENTAGE OF VICTORIES AND AVERAGE OF SCORE ACHIEVED (PLUS STANDARD ERROR) IN 10 DIFFERENT GAMES. FOURTH AND SIXTH COLUMNS INDICATE THE APPROACHES THAT ARE SIGNIFICANTLY WORSE THAN THAT OF THE ROW, USING THE NON-PARAMETRIC WILCOXON SIGNED-RANK TEST WITH P-VALUE $< 0.05$. BOLD FONT FOR THE ALGORITHM THAT IS SIGNIFICANTLY BETTER THAN THE OTHER 3 IN EITHER VICTORIES OR SCORE.


D. Wierstra, S. Legg, and D. Hassabis, "Human-level Control through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 02 2015.

[3] D. Perez-Liebana, J. Togelius, S. Samothrakis, T. Schaul, S. M. Lucas, A. Couetoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 General Video Game Playing Competition," *IEEE Transactions on Computational Intelligence and AI in Games*, 2015.

[4] T. Schaul, "A Video Game Description Language for Model-based or Interactive Learning," in *Proceedings of the IEEE Conference on Computational Intelligence in Games*, 2013, pp. 193–200.

[5] G. M. J.-B. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-Carlo Tree Search: A New Framework for Game AI," in *Proceedings of the Artificial Intelligence for Interactive Digital Entertainment Conference*, 2006, pp. 216–217.

[6] C.-S. Lee, M.-H. Wang, G. M. J.-B. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong, "The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 1, pp. 73–89, 2009.

[7] C. Browne, E. J. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4:1, pp. 1–43, 2012.

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, vol. 529, no. 7587, pp. 484–489, 01 2016.

[9] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley, 2001.

[10] P. Vamplew, R. Dazeley, A. Berry, R. Issabekov, and E. Dekker, "Empirical Evaluation Methods for Multiobjective Reinforcement Learning Algorithms," *Machine Learning*, vol. 84, pp. 51–80, 2010.

[11] E. Zitzler, *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. TIK-Schriftenreihe Nr. 30, Diss ETH No. 13398, Swiss Federal Institute of Technology (ETH) Zurich: Shaker Verlag, Germany, 1999.

[12] D. Perez-Liebana, J. Dieskau, M. Hunermund, S. Mostaghim, and S. M. Lucas, "Open Loop Search for General Video Game Playing," in *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO '15*. Association for Computing Machinery (ACM), 2015, pp. 337–344.

[13] R. B. Myerson, *Game Theory : Analysis of Conflict*. Cambridge (Mass.), London: Harvard university press, 1997, 1991.

[14] K. Miettinen, *Non-linear Multiobjective Optimization*. Kluwer, International Series in Operations Research and Management Science, 1999.