Multiobjective Exploration of the StarCraft Map Space

Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelbäck, and Georgios N. Yannakakis

***Abstract*—This paper presents a search-based method for generating maps for the popular real-time strategy (RTS) game *StarCraft*. We devise a representation of StarCraft maps suitable for evolutionary search, along with a set of fitness functions based on predicted entertainment value of those maps, as derived from theories of player experience. A multiobjective evolutionary algorithm is then used to evolve complete StarCraft maps based on the representation and selected fitness functions. The output of this algorithm is a Pareto front approximation visualizing the tradeoff between the several fitness functions used, and where each point on the front represents a viable map. We argue that this method is useful for both automatic and machine-assisted map generation, and in particular that the Pareto fronts are excellent design support tools for human map designers.**

Keywords: Real-time strategy games, RTS, procedural content generation, evolutionary multiobjective optimization

I. Introduction

Procedural content generation (PCG) refers to the automatic or semi-automatic generation of game content. PCG comes in many flavors, as there are many types of game content that can be generated (such as levels, adventures, characters, weapons, planets, plants, histories) and many ways in which it can be generated (many based on AI/CI methods such as constraint satisfaction, planning or evolutionary computation, others on e.g. fractals). PCG can also be used in different ways in games, for example for offline content creation during game development, support tools for human designers or fully automatic online content creation based on player actions. Similarly, there are different motivations for using PCG, such as speeding up game development, saving human designer effort/cost, saving main memory or DVD storage, academic curiosity or the making possible of completely new types of games. What is clear, however, is that PCG is gaining increasing attention among both commercial game developers, indie developers and academic game researchers.

This paper contributes to the flora of PCG approaches by presenting a search-based approach to generating maps to real-time strategy games. More specifically, we use a multiobjective evolutionary algorithm to generate maps for the game StarCraft, using fitness functions based on theories of player entertainment. We believe this approach has significant merits over previous approaches to generating terrains, and also that we are the first to automatically generate complete maps for a specific strategy game. We extend previous work published in [1] by devising a new map representation compatible with StarCraft, and a new set of fitness functions tailored to this map representation.

JT and GNY are with IT University of Copenhagen, 2300 Copenhagen S, Denmark. MP, NB and SW are with TU Dortmund, Otto-Hahn-Str. 14, Dortmund, Germany. JH is with Blekinge Institute of Technology, Ronneby, Sweden. Emails: julian@togelius.com, mike.preuss@cs.tu-dortmund.de, nicola.beume@cs.tu-dortmund.de, simon.wessing@cs.tu-dortmund.de, johan.hagelback@bth.se, yannakakis@itu.dk

A. Procedural map and terrain generation

Maps are central to many computer games, including First-Person Shooters (FPS) and many Role-Playing Games (RPG), in which the player experiences the world from a first-person perspective as he navigates a typically hostile environment. But they are perhaps most important for strategy games, both of the turn-based variety and Real-Time Strategy (RTS) games. In these games, the player views the playing area from a third-person perspective (usually from above) while directing one or several units as they traverse an area and perform missions, usually involving battle. In this paper, we will mainly be concerned with RTS games.

Most strategy games come with a set of hand-crafted maps, used both in single-player "campaign" mode and multi-player matches. However, there are numerous reasons for wanting to automatically generate maps. Perhaps the most obvious reason is that by generating a fresh map each time the game is played, you extend the life-span of the game by permitting the player to explore a fresh map and the specific challenges it entails each time the game is played. This also means that any advantages a player has accrued through learning a map by heart are nullified.

A slightly less obvious reason is that maps could be tailored to suit specific players or groups of players, and/or to generate particular gameplay experiences. For example, a player that has proven adept at a particular form of strategy might be presented with a freshly generated map that challenges her to develop other aspects of her strategic thinking; or, if she has been determined by the game to be less motivated by challenge and more by easy progress, a new map could be generated that plays to the strengths of her particular playing style while seemingly dissimilar to previous maps she has played. In a multi-player game, maps might be generated that balance out the strengths of of different players' playing styles and levels of proficiency, without resorting to explicit handicapping in terms of game rules or units supplied. Such a mechanism would place particular demands on models of player behavior and preferences, as well as on how the map creation algorithm can be controlled.

But one might also want to use procedural map generation algorithms as authoring and design support tools, to complement human creativity. In this case the PCG tools would be used off-line, before a game is shipped or before new high-quality maps are made available for download. The role of the algorithm would be to suggest new map designs according to specified parameters or constrains, which could then be modified and refined by human map designers.

978-1-4244-6297-1/10/$26.00 ©2010 IEEE

While most strategy games stick with prefabricated maps (possibly complemented with an end-user map editor), a significant minority are based on random map generation. An influential example is the *Civilization* series of epic turn-based strategy games, in which the default game mode sees the player playing on a newly randomly generated world map. No details have to the authors' best knowledge been released about Civilization's map generation algorithm, but the very short time taken to generate a map suggests a relatively uncomplicated algorithm. The available parameters for map generation are relatively few, the most important one relating to size and connectedness of the world's landmass.

The probably simplest way of generating maps similar to those used by Civilization is to seed the ocean with embryonal islands, and having them grow out in random directions a predefined number of steps [2]. Slightly more advanced approaches involve using fractals, such as the diamond-square algorithm [3]. The diamond-square algorithm works by iteratively subdividing areas of space and offsetting the midpoint by random amounts. An advantage of this family of algorithms is that they are so fast that they can often be used for real-time terrain generation [4].

Recently, Doran and Parberry suggested the use of software agents for generating terrain [5]. In their approach, a large number of agents are let loose on an initially featureless piece of terrain and collectively shaping it. Each type of agent has a particular task, and the workings of some of them resemble forces of nature; so for example the river agents travel from mountains to coast following the steepest descent gradient. This approach is claimed to be more controllable than fractal-based terrain generation algorithms.

The *roguelike* genre of games (the original *Rogue* game as well as countless successors, such as *Nethack*, *Moria* and *Diablo*) is unique in being fundamentally based on random map generation. In these games the player fights through a randomly generated dungeon – walls, placements of monsters, traps and treasure are all generated at the beginning of each game or play session. The dungeon generators used here often work either similarly to fractal terrain generation approaches (generate a straight line from start to exit, iteratively deform the path a number of times, and then grow randomly branching paths until the room is filled), or by glueing together a number of prefabricated segments [2].

B. Search-based procedural content generation

The above examples represent what can be called *constructive* PCG. This means that the generation algorithm only makes one attempt: it proceeds from start to finish with none or only insignificant backtracking. In contrast to this, *generate-and-test* algorithms make several attempts, and only keep those candidate maps content instances that pass some sort of test. One example is Tarn Adams' ambitious game *Dwarf Fortress*, for which initial fractal map generation is usually repeated a couple of times, and the user is shown screenshots of "failed" maps along with explanations of what went wrong, e.g. wrong elevation distribution.

Search-based procedural content generation (SBPCG) is a particular type of generate-and-test PCG, where the generated candidate content is not simply rejected or accepted by the test but graded on one or several numeric dimensions, and where a search algorithm is used to find better content based on the evaluations of previously generated content.

Usually, some sort of evolutionary algorithm (e.g. a genetic algorithm or an evolution strategy) is used as the core algorithm for SBPCG. In these cases, a population of candidates (e.g. maps) is created randomly at the beginning of a run of the algorithm, and at each generation the worst candidates (according to some *fitness function*) are replaced with new candidates generated through mutation and/or recombination from the best candidates. Core concerns when devising an SBPCG solution to some content generation task is how to represent the content and how to devise the fitness function. An overview of SBPCG can be found in [6].

One of the main arguments for SBPCG is that it allows the designer to formulate the desired properties of the content more explicitly than with other content generation methods. Another argument is that it allows the use of content representations that sometimes yield infeasible solutions (e.g. unusable maps), as such candidates can be discarded but still form the basis for later, better candidates. The main argument against SBPCG is that it can be very time-consuming, making it less suitable for real-time PCG – but choosing the fitness function and the search space carefully can allow content to be generated in a fraction of a second.

There have been a few previous attempts to use evolutionary algorithms to generate height maps for terrains before. Frade et al. used genetic programming to evolve terrains, with the evolved expression tree mapping coordinates on a grid to elevation at that point. The fitness function was based on "accessibility" meaning that all flat areas should be connected while no individual flat area grows too big. Only the height map was evolved, no other features of the map [7].

Sorenson and Pasquier evolve simple dungeon layouts for e.g. rogue-like games, using a map representation where rooms and hallways of different sizes are placed on a two-dimensional surface which is by default untraversable. The fitness function is simply the length from start to finish, and the only constraint that the path should be connected [8]. Similarly, Ashlock et al. evolved path-planning problems in which the objective was to maximize distance from start to finish by placing walls at various positions and angles [9].

In the above examples, only parts of game environments (e.g. height maps and walls) are evolved – not complete, playable levels with e.g. items, monsters, resources. This is probably why the fitness functions are only tangentially related to actual game playability and entertainment; path length and accessibility do not alone make for a well-designed level. In contrast, some recent SBPCG papers have explicitly been based on notions of player entertainment. Togelius et al. evolved racing game tracks based on objectives inspired by Malone's entertainment dimensions [10]; Pedersen et al. evolved levels for Super Mario Bros based

on a data-driven model of player affect [11]; Hastings et al. evolved weapons for a 2D shooter based on player activity in the game [12]; Togelius and Schmidhuber evolved predator-prey games [13]; and Browne evolved board games based on measures derived from studies of successful games [14]. None of these studies concerned maps or terrains, however.

C. Multiobjective evolution

In standard evolutionary computation a single fitness function is used to evaluate candidate solutions. However, for many problems it is hard to combine all demands into a single objective measure; e.g. when we want a car to be cheap, fast and safe, we need to optimize in three fitness dimensions. Several objectives are typically conflicting, for example a faster car is typically less cheap.

The intuitive solution is to simply add the fitness measures together (using some weighting of each measure), and optimize for the resulting composite measure. This method has several drawbacks. One is that you do not know the appropriate weighting of the fitness dimensions until you have investigated the distribution of solutions among each dimension. Another is that optimization along a single dimension does not allow for exploration of the often complicated ways in which the various fitness dimensions interact (e.g., above a certain price threshold faster cars might not be less cheap).

Multiobjective evolutionary algorithms (MOEA) were invented to solve this problem, and are now a major research direction within evolutionary computation as well as common in industrial applications. An MOEA presumes at least two fitness functions and proceeds towards the *Pareto front* of Pareto-optimal solutions, i.e. solutions satisfying that there is no other solution being equal or better in all dimensions. The valuable result of an MOEA is its final set of solutions, whose subset of non-dominated solutions (optimal within the set) presents an approximation of the Pareto front.

When using two or three objectives[1], the Pareto front can be conveniently plotted in a graph, allowing visual exploration of the tradeoffs between these objectives. Visual or automated inspection of Pareto fronts helps to detect situations where a small improvement in one objective would lead to a huge loss in another, which is usually undesired. The possibility to visualize the tradeoffs inherent in a design problem makes multiobjective optimization via MOEAs a great but underused tool for design and authoring support.

Optimizing some aspect of a game for playability is inherently a multiobjective problem, as it is very hard to formulate a single-dimensional automatic measure of how entertaining a game is; it is indeed not trivial to formulate partial measures of game enjoyability. When designing game content, it would seem invaluable for a designer to be able to conveniently visualize the tradeoffs inherent in a design problem; when automatically generating game content tailored to particular players, it would also seem ideal to first generate a selection of candidate content from which appropriate game content for the particular player could then be chosen, based on their previous playing style and experience model. Despite this seemingly perfect fit, we have not seen any examples of MOEAs used for PCG; the closest we can find are examples of multiobjective evolution of NPC behavior [15].

[1]More than three objectives are usually hard to handle for any MOEA, as the number of incomparable solutions—better in some objective, but worse in another—grows exponentially with the number of objectives.

D. This paper

In this paper, we show that search based procedural content generation can be used to automatically create playable maps for a very popular real-time strategy game. In order to do this, we propose and motivate a number of fitness measures for such maps, which we argue can also be generalized to maps in other games. We also show how multiobjective evolution can be used as a design support tool, by exploring the tradeoffs between the proposed fitness functions.

In the following sections, we describe: the StarCraft game, which we use to test our maps; the map representation and genotype-to-phenotype mapping; our fitness functions; the multiobjective evolutionary algorithm we use; our experiments, and finally what we can learn from all this.

II. The StarCraft real-time strategy game

StarCraft is one of the most famous strategy games ever. It was released by Blizzard Entertainment in 1998 and has, as of 2009, sold more than 11 million copies [16]; it is famous for its fine balance between the different playable factions, and very popular for tournament play.

The game features three factions; terrans, humans that have left planet earth to travel to distant areas of our galaxy; zerg, a race of insectoid like creatures; and protoss, a humanoid race with very advanced technology and psionic abilities.

In the game the player has to plan and build a base with different structures, each with a specific purpose. To afford structures and building units the player has to gather resources from minerals and vespene gas, located around the game map. Units must be created to defend the home base and to attack and defeat the enemy players. Different units have different strengths and weaknesses; e.g., some are good defenders, some deal plenty of damage but are not very mobile, others are fast but don't do very much damage. The game also features a technology tree in which players can spend resources to research upgrades for units and structures.

The game can be played in a single-player story line mode, or a skirmish mode where the player battles against other players or computer controlled enemies. A large world-wide fan base has contributed large amounts of player generated content, such as multiplayer maps and map editors.

III. Map representation

In these experiments we evolve maps containing all of the crucial elements for a StarCraft map. These are: locations for bases and for two types of resources (minerals and Vespene gas), and areas of impassable terrain (mountains and rivers).

We use two different representations of the map — an indirect representation used for searching (the *genotype*), and

a direct representation for fitness testing and visualization (the *phenotype*). Each time fitnesses are calculated, a phenotype is created from each genotype. The genotype (indirect) representation is a fixed-length array of real values between 0 and 1. The length of the array is decided by the number and types of map elements. Four types of elements are possible, with parameters as follows:

- **Base:** ϕ and θ coordinates of each base
- **Mineral source:** x and y coordinates of each mineral source.
- **Gas well:** x and y coordinates of each gas well.
- **Impassable area:** These represent water or rocks in the finished map. Each impassable area has five parameters, namely x and y start coordinates, probability of left turn, probability of right turn, and probability of gap.

For our experiments, we generated maps with three bases, four resources of each type and five impassable areas, leading to genomes of length $3 \cdot 2 + 5 \cdot 2 + 5 \cdot 2 + 5 \cdot 10 = 76$.

This map representation has the advantage that it can be efficiently searched by many common global optimization algorithms, such as evolution strategies and particle swarm optimization. In particular, many of these algorithms assume a real-valued representation, and that local changes in the genotype have local effects in the phenotype. For example, when changing the ϕ coordinate of the base, the positions of nearby resources are not changed, and neither are the mountains; it is easy to imagine representations where this would not be the case, such as many fractal representations. Additionally, this representation is scale invariant; a phenotype of any size can be created out of the genotype.

The phenotype (direct) representation is designed to be easy to base fitness calculations on, and to convert to StarCraft's internal map format. The representation consists of a heightmap in the form of a 64×64 grid (the size of a small StarCraft map) where each cell can be either *passable* or *impassable* and three lists of x and y coordinates of bases, mineral sources and gas wells, respectively. The lists of resource sites are populated from the corresponding lists in the genotype representation by simply multiplying each x and y coordinate by 64.

The coordinates for each base are generated using a method based on polar coordinates. The two parameters for the base are treated as angle and length of an axis extending from the center of the map, at the end of which the base is placed. Additionally, the representation is constrained so that each base is forced to be within its own arc of the circle, meaning that for three bases each base is placed within its own 120 degree arc; the length of the axis is constrained to between $1/2$ and 1 of the radius of the map, meaning that bases cannot be place too close to the center of the map. By means of polar coordinates, we restrict base placement so as to make neighboring bases unlikely in order to increase the chances of obtaining a playable map. Coordinates lying outside the map are simply mapped to the outermost cell of the map in that direction. This increases the probability of placing bases on the map borders and is a desired effect.

All cells of each map phenotype are by default passable. Impassable areas are then "drawn" in a manner similar to turtle graphics [17]. The drawing of each impassable area starts at its designated x and y position by marking that cell as impassable. The "pen" then repeatedly moves one step in its current direction (starting direction is right) and marks the new cell as impassable, until it reaches a cell which is already impassable or the border of the map. At each cell, it decides whether to turn left, turn right and/or "lift the pen" and leave a gap in the line according to its designated probability for each of these actions. Only one of these actions is taken at each step, with a turn angle of 45 degrees. That is, if the turtle turns left, the next step starts over again at the same position without painting. If it does not turn left, the probability for a right turn is checked, and if it does not turn right, the probability for a gap is checked. If none of this applies, the turtle just moves one step forward in its current orientation and marks the new position as impassable. As it often happens that the resulting line is not closed, one attempt to draw towards the original x and y starting position is made by simply setting the orientation according to the vector between current and starting position and starting the whole process over again. One further additional constraint is used to prevent very long lines without turns: whenever 5 consecutive steps have been made into one direction, the orientation of the turtle is changed by rotating it 45 degrees into the direction to the starting position.

In order to ensure a completely deterministic genotype to phenotype mapping, a fixed random number table with 200 entries is used to decide whether to turn and/or leave gaps.

The last steps in the generation of a complete StarCraft map are that (1) a GIF image file is generated from the phenotype, in which each cell type has a different color, and that (2) the SCPM software (available at http://www.clanscag.com) automatically creates a complete StarCraft map from the image. Further manual editing is then possible using StarCraft map editors. The maps shown in this paper have been slightly edited for visual appeal, without changing the functional structure of the evolved maps.

IV. Fitness functions

In SBPCG, there is a distinction among three types of fitness functions: *interactive*, *simulation-based* and *direct* [6]. Interactive fitness functions rely on human game players playing the candidate content and providing direct or indirect feedback about its quality. While in a sense the ultimate type of fitness function, interactive fitness functions require massive amounts of player input and are only possible in some types of games, such as ongoing massively multiplayer games [12]. Simulation-based fitness functions assess content automatically through algorithmically playing the game or some aspect of the game using the candidate content. Such evaluations can potentially be accurate predictors of player enjoyment, but require both artificial intelligence capable of playing the game competently in a human-like manner and often substantial computation time [10], [13]. Direct fitness

functions base their fitness calculations directly on the phenotype representation of the content. Such fitness functions are obviously much easier to implement and faster to compute than simulation-based functions, but it is hard to devise direct fitness functions that accurately predict key aspects of player experience (except when basing them on data-driven player models built from extensive user studies [11]).

For this paper, we do not have the luxury of having human players sit through countless hours to test the tens of thousands of candidate maps the evolutionary algorithm generates, nor any reliable and efficient way of testing maps through algorithmic playthrough of the full game. However, we can simulate one key aspect of RTS gameplay: moving between two points along the fastest possible path. We use the classical A* algorithm for this task, which returns the number of cells along the shortest path (avoiding impassable areas) – if not otherwise specified, "distance" means number of cells on the shortest path found by A* in the rest of the paper. We defined eight different fitness measures (mainly based on distance) intended to reflect various desired game characteristics. It was at the time of their formulation not clear to which degree the various functions conflicted or induced searchable fitness landscapes. The experiments in this paper investigate the interplay of *pairs* of these functions.

The designed fitness functions are motivated by a number of desirable characteristics of good StarCraft maps:

- *Playability:* It should be possible to engage in normal gameplay: building up a base, attacking enemies etc.
- *Fairness:* All players should have similar possibility of winning the game given the same skill level. Note that this does *not* necessarily mean that starting positions should be or look similar.
- *Skill differentiation:* Superior tactics should win more often, so the map should allow use of different tactics.
- *Interestingness:* Maps should not all look the same, and should not be bland (e.g. symmetrical or featureless).

Before calculating any of the below fitness measures, the map is "sanity checked" by ensuring that every base and all resources are accessible (there exists a path which is not blocked by impassable areas) from every other base. Any map not satisfying these criteria is assigned a fitness of 0 in all objectives, effectively discarding it. This test ensures basic playability. All fitness functions are to be maximized and are normalized to values in $[0, 1]$.

The first two fitness functions relate mainly to the properties of the placement of players' starting bases, and to the impassable area around and between bases.

- f_{b0}*: Base space*. For playability, some space for other buildings is required next to the base. Out of the $5 \cdot 5$ cells surrounding a base, the base space is defined as the fraction of these cells that are passable and reachable within 5 steps (using A*) from the base. This fitness value is the mean of the base space of all bases.
- f_{b1}*: Base distance*. The measure makes sure that the bases are not too easy to reach from each other so that the players have the opportunity to develop their base before clashing with the others. It contributes to playability and skill differentiation as the game is more difficult for all players when starting close to each other. f_{b1} is the minimum distance between any two bases, divided by the sum of the map's width and height.

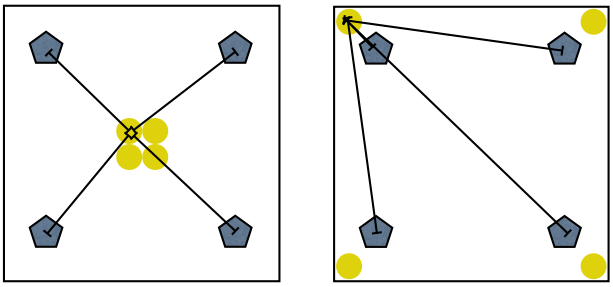

Fig. 1: Unsafe (left) and safe (right) resources. Bases are depicted by pentagons, resources as circles. The lines mark shortest possible paths for attackers/defenders.

The next four fitness functions relate to the placement of resources, relative to each other and to bases; all of these measures mainly contribute to fairness.

- f_{r1}*: Distance from base to closest resource*. The distance from each base to its closest mineral and its closest gas wells is calculated. f_{r1} is the quotient between the minimal and maximal distance to the closest resource for all bases.
- f_{r2}*: Resource ownership*. Each base is associated with its closest resource (done separately for minerals and gas wells) and the base is considered as the owner of that resource. In case a resource is the closest to more than one base, the bases own only a fraction of it each (assuming fair sharing). f_{r2} is the average fraction players own of their closest resources, where a value of 1 means that all resource are clearly assigned.
- f_{r3}*: Resource safety*. Another measure of how clearly resources are assigned to a single player, f_{r3} measures the average deviation of path lengths between one resource and all bases (see Fig. 1). So, for bases $b_1, ..., b_n$ and resources $r_1, ..., r_m$ we calculate all path lengths between resources and bases and group them by resource type: $\forall j = 1, \ldots, m : D_j = \{\text{dist}(r_j, b_i) \mid i = 1, \ldots, n\}$. $f_{r3} = \min\{s_{\text{gas}}, s_{\text{minerals}}\}$, where s_{gas} and s_{minerals} are simply the average standard deviations of the respective sets D_j.
- f_{r4}*: Resource fairness*. For each base, the shortest distance to both types of resources is calculated. The fitness is then calculated as $1 - (max - min)$, where max and min are the maximum and minimum distances between a base and its nearest resource.

The remaining two fitness functions deal with the character of the paths of the map. These functions mainly contribute to skill differentiation and interestingness.

- f_{p1}*: Choke points*. We consider the average narrowest gap on all paths between bases. The narrowest gap along a path is calculated by first calculating a shortest path

and then traversing along the path and counting the width of the path at each cell. Path width is calculated through determining whether the path is currently moving horizontally or vertically through comparison with the previous cell in the path, and searching orthogonally to the path direction until either an impassable cell or the border of the map is encountered. Choke points contribute to skill differentiation in that a good player will be able to exploit such points through using a smaller defending force to stop a larger attacking force, which cannot use the strength of its numbers as they have to pass sequentially through the narrow gap.

- f_{p2}*: Path overlapping*. We consider the paths from the bases to all resources and calculate to what extent they overlap. In case many cells are used from different bases we assume that the players' units are likely to meet. The value of f_{p2} is the average number of uses of the map's cells. It contributes to skill differentiation, as it increases the number of possible flash points which the player must monitor for conflicts.

V. Multiobjective evolutionary algorithm

Most MOEAs work relatively similarly. A population of search points (called individuals for historical reasons) is generated randomly at first, and then adapted to the problem in order to move towards the Pareto front by a repeated cycle of variation and selection. Variation creates new search points by mixing information of existing ones (recombination) and performing undirected steps with a defined expected length (mutation). Selection choses the best of the old and new individuals for the preceding iteration and deletes the others. This working principle has its advantages in the minimal necessary knowledge of the optimization problem (black box, no algebraic form or gradients needed) with which it is capable to handle complex problems. On the other hand, black box algorithms are somewhat slower than classical optimization algorithms on convex/very simple problems.

The most popular and long-established MOEA, NSGA-II [18], has proved its worth in many benchmark and real-world applications. However, it is nowadays outperformed by state-of-the-art MOEAs, such as the SMS-EMOA [19] which is known as a fast descendant of the NSGA-II.

The SMS-EMOA, which we use in this paper, generates only one new individual per cycle and removes the individual with the smallest hypervolume contribution, i.e. the one that dominates the smallest objective space. To accommodate the need for setting one or several constraints, we employ a modified selection scheme here. Individuals outside the allowed region get a penalty equaling their distance to it. When considering which individual to remove, the one with the largest penalty always gets precedence. Thus, valid individuals are never removed in the presence of invalid ones.

We employ standard recombination/mutation operators SBX and PM [20], and set the run length after some testing to 50000 evaluations. In all experiments, we use populations of 20 individuals, which we consider sufficient to achieve a representation of the Pareto front.

VI. Experiments

A. Initial Study

Before the main investigation of tradeoffs between our fitness functions, we performed initial exploratory studies to see whether the functions were possible to optimize, whether they were trivial, and whether there seemed to be any conflicts with other objectives at all.

We found that both of the base placement functions were very simple to optimize to maximal or near-maximal values. We therefore included both of them as constraints in the map generation. f_{b0} (base space) is additionally not conflicting with any other objective, so we made it a hard constraint (maps with $f_{b0} < 0.5$ are discarded) and do not use it as a proper objective. As there can still be some value to maps with low f_{b1} (base distance), maps with $f_{b1} < 0.5$ are just penalized by subtracting $0.5 - f_{b1}$ from all their fitness values; additionally, f_{b1} is used as an objective in its own right.

B. Main Study

The aim of our main study was to find out the degree of conflict between the map objectives we have invented. We performed a number of 2-objective runs, where we test pairs of objectives against each other. All objectives except f_{b0} were tested, and each pair was used in 10 runs; the results can be seen in tables I and II, using two different indicators of the degree of conflict.

Table I shows the average sizes of the final Pareto front approximations, i.e. the number of non-dominated solutions in the last generation. In the absence of any conflict between two objectives, the Pareto front would contain a single individual that maximizes both objectives. We therefore consider small fronts to be indicators of a low degree of conflict.

Table II shows the hypervolume of the final non-dominated sets relative to the reference point $(1, 1)$. A value of 1.0 indicates that both objectives are maximized to optimality (or close enough regarding the numerical accuracy of the tables). In case the Pareto front approximation is very accurate the hypervolume value even reflects the shape of the front, e.g. a diagonal line has a value of 0.5. For this indicator, *low* values indicate *high* degrees of conflict.

Note that the algorithm might or might not be able to find the true Pareto front, in general or within the allotted number of generations. There is no way of knowing the accuracy of the current Pareto front approximation; we can therefore not say with absolute certainty that there is or is not a conflict.

Fitness function f_{r2} (resource ownership) is very easy to optimize. It reached very high hypervolume values indicating that there is little conflict with other objectives, and that it might be more suitable as a constraint. However, surprisingly, when combining f_{r2} with f_{p1} (choke points) the whole population is non-dominated.

f_{p1} is itself very easy to optimize as having paths of minimal width it just requires small gaps in some impassable barrier; see figure 2 for examples. f_{r2} and f_{r3} are both attempts at measuring almost the same underlying quality, and predictably there is almost no conflict between them;

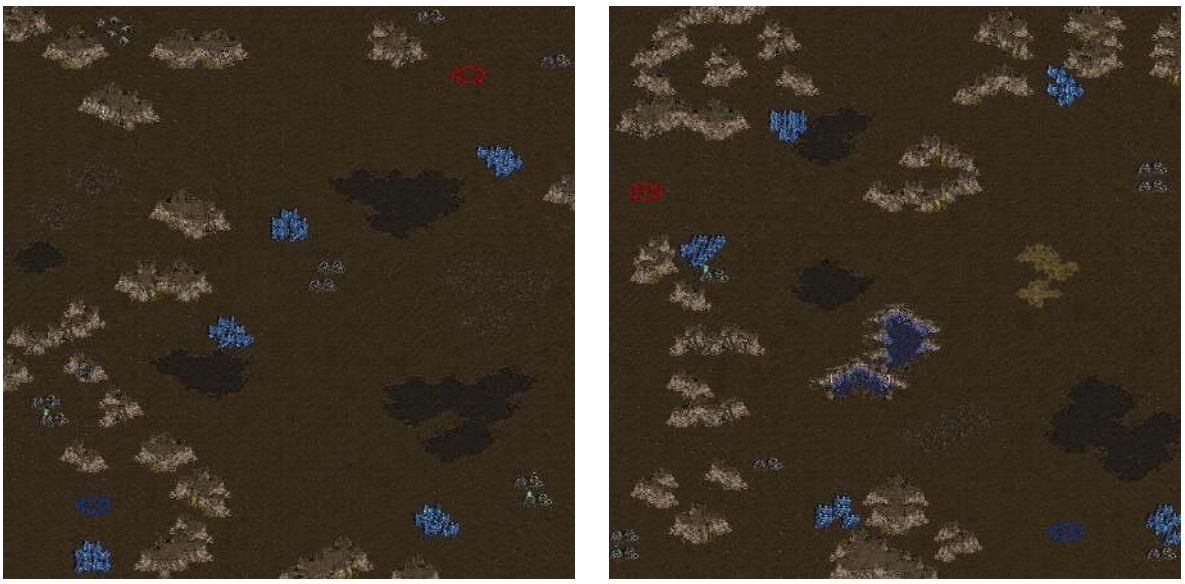

Fig. 2: Example maps generated by simultaneous optimization of f_{p1} and f_{r4}.

TABLE I: AVERAGE NUMBER OF NON-DOMINATED INDIVIDUALS IN THE FINAL POPULATION FOR EACH FUNCTION COMBINATION.

	f_{r1}	f_{r2}	f_{r3}	f_{r4}	f_{p1}	f_{p2}
f_{b1}	6.9	1.6	5.0	7.8	2.9	7.5
f_{r1}		5.8	9.1	3.4	3.7	7.6
f_{r2}			1.2	2.7	20.0	1.3
f_{r3}				7.3	3.3	8.7
f_{r4}					2.8	8.1
f_{p1}						4.2

TABLE II: AVERAGE HYPERVOLUME VALUES OF THE NON-DOMINATED INDIVIDUALS IN THE FINAL POPULATION.

	f_{r1}	f_{r2}	f_{r3}	f_{r4}	f_{p1}	f_{p2}
f_{b1}	0.675	0.724	0.394	0.673	0.644	0.075
f_{r1}		1.000	0.452	0.993	0.895	0.107
f_{r2}			0.504	0.993	0.900	0.114
f_{r3}				0.473	0.479	0.053
f_{r4}					0.891	0.108
f_{p1}						0.099

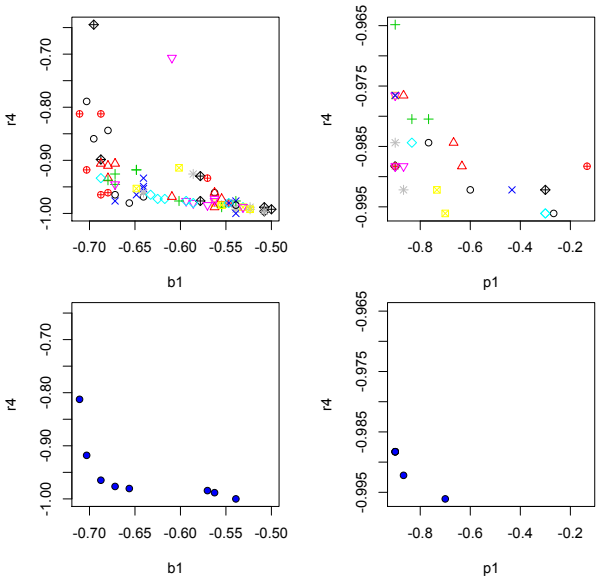

Fig. 3: Pareto front approximations for (f_{b1}, f_{r4}), (f_{p1}, f_{r4}). The upper plots depict the results of 10 separate runs; the lower ones their combined non-dominated individuals.

the average Pareto front size is just over 1. All hypervolume values involving f_{p2} (path overlapping) are very small, maybe due to inadequate normalization. An improvement would be to normalize with respect to free cells only rather than all cells. Figure 3 shows a selection of Pareto front approximations for the objectives (f_{b1}, f_{r4}) and (f_{p1}, f_{r4}). The diversity is low and special mechanism are required to improve it. For technical reasons, the objectives have been negated and values transformed to $[-1, 0]$ for minimization in the SMS-EMOA.

C. Map Generation

Figure 2 depicts two maps resulting from the simultaneous optimization of f_{r4} and f_{p1}. The map was generated using the method described in section III but with only two bases. These maps are used as training maps in the CIG 2010 RTS StarCraft competition [21]. The large blue and red circles mark the two bases. Minerals are indicated by light blue diamonds, gas wells by a crater. The impassable areas are drawn either as mountains (gray) or as water (dark blue). The bases are situated close to the map borders (probably due to the base placement method and the f_{b1} constraints), the impassable areas are perforated with small gaps (f_{p1}) and the resources are very evenly distributed (f_{r4}).

D. Discussion

Our various fitness functions turned to differ greatly in how easily they were to optimize and their potential for interesting conflicts with other objectives. The base placement functions f_{b1} and f_{b2}, were so easy to optimize that they could be converted to constraints.

The result of optimizing for the resource placement functions looked very different upon visual inspection. We were less than satisfied with functions f_{r1} and f_{r2}; the latter because it is too easy to optimize, and the former because it results in maps that do not look very StarCraft-like. f_{r4}, which considers all resources rather than just the closest ones, renders much more palatable results. This suggests that a map generator could use something like f_{r4} to generate the global resource placement, and then simple place one resource of each type within a single-screen line of sight from each base. A similar measure that allows the difficulty of the resources to be scaled would be interesting as well.

Optimizing the choke point function f_{p1} tends to generate scattered and disconnected impassable areas, suggesting that optimizing for low values of the same functions could generate areas of compact impassable areas and open spaces. This is a very nice feature, and when used together with a conflicting objective allows us to generate a continuum between extremes in terms of both gameplay and visual appearance. At the same time, a more refined choke point function could be devised that aims for single gaps in otherwise connected impassable barriers. We see some potential in maximizing the interaction of players by the path sharing function, but in its current form it is hampered by inappropriate normalization. Yet other similar measures may help to design maps of different character and complexity in order to scale between different levels of player experience.

VII. Conclusions

In this paper, we used multiobjective evolutionary algorithms, together with a relatively indirect map representation to evolve complete playable maps for the RTS game StarCraft. A number of fitness functions measuring map qualities connected to playability, fairness, skill differentiation and interestingness were defined and their interplay investigated. We believe this is the first time search-based procedural content generation has been used to create playable game maps, and possibly the first time multiobjective optimization has been used for any sort of content generation for an actual game. We have also shown that there exist a number of interesting tradeoffs between map objectives, which can be used together with multiobjective optimization to automatically explore the boundaries of design space for a particular class of content. Such a mechanism can be used both for completely automated adaptive content generation, and to assist human content designers.

Future work will deal with optimizing more than two objectives simultaneously, and on combining direct fitness functions (as used here) with simulation-based fitness functions. We will also verify the quality of generated maps through user studies. Maps generated by our methods will be used in the CIG 2010 RTS competition on StarCraft [21].

Acknowledgments

This research was supported in part by the Danish Research Agency, Ministry of Science, Technology and Innovation; project name: Adaptive Game Content Creation using Computational Intelligence (*AGameComIn*); project number: 274-09-0083.

References

[1] J. Togelius, M. Preuss, and G. N. Yannakakis, "Towards multiobjective procedural map generation," in *Workshop on Procedural Content Generation in Games, co-located with 5th Intl. Conference on Foundations of Digital Games*. ACM Digital Library, 2010, (to appear).

[2] T. Adams, "Re: Optimization-based versus "constructive" PCG," 2009, Post to the "Procedural Content Generation" Google Group.

[3] G. Miller, "The definition and rendering of terrain maps," in *Proc. of SIGGRAPH Computer Graphics*, vol. 20, no. 4, 1986, pp. 39–48.

[4] J. Olsen, "Realtime procedural terrain generation," 2004, http://oddlabs.com/download/terrain_generation.pdf.

[5] J. Doran and I. Parberry, "Controllable procedural terrain generation using software agents," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 2, 2010.

[6] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation," in *Proc. of the European Conference on Applications of Evolutionary Computation (EvoApplications)*, vol. 6024. Springer LNCS, 2010, pp. 141–150.

[7] M. Frade, F. F. de Vega, and C. Cotta, "Evolution of artificial terrains for video games based on accessibility," in *Proceedings of the European Conference on Applications of Evolutionary Computation (EvoApplications)*, vol. 6024. Springer LNCS, 2010, pp. 90–99.

[8] N. Sorenson and P. Pasquier, "Towards a generic framework for automated video game level creation," in *Proceedings of the European Conference on Applications of Evolutionary Computation (EvoApplications)*, vol. 6024. Springer LNCS, 2010, pp. 130–139.

[9] D. Ashlock, T. Manikas, and K. Ashenayi, "Evolving a diverse collection of robot path planning problems," in *Proceedings of the Congress On Evolutionary Computation*, 2006, pp. 6728–6735.

[10] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards automatic personalised content creation in racing games," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2007.

[11] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modeling player experience in Super Mario Bros," in *Proc. of the IEEE Symposium on Computational Intelligence and Games*, 2009, pp. 132–139.

[12] E. Hastings, R. Guha, and K. O. Stanley, "Evolving content in the galactic arms race video game," in *Proc. of the IEEE Symposium on Computational Intelligence and Games*, 2009.

[13] J. Togelius and J. Schmidhuber, "An experiment in automatic game design," in *Proc. of the IEEE Symposium on Computational Intelligence and Games*, 2008, pp. 111–118.

[14] C. Browne, "Automatic generation and evaluation of recombination games," Ph.D. dissertation, Queensland Univ. of Technology, 2008.

[15] A. Agapitos, J. Togelius, S. M. Lucas, J. Schmidhuber, and A. Konstantinides, "Generating diverse opponents with multiobjective evolution," in *Proc. of the IEEE Symposium on Computational Intelligence and Games*, 2008, pp. 135–142.

[16] "Starcraft," http://en.wikipedia.org/wiki/Starcraft/. URL visited on 2010-03-27, 2010.

[17] H. Abelson and A. diSessa, *Turtle Geometry—The Computer as a Medium for Exploring Mathematics*. MIT Press, 1986.

[18] K. Deb, A. Pratap, and S. Agarwal, "A fast and elitist multi-objective genetic algorithm: NSGA-II," *IEEE Trans. on Evolutionary Computation*, vol. 6, no. 8, pp. 181–197, 2002.

[19] N. Beume, B. Naujoks, and M. Emmerich, "SMS-EMOA: Multiobjective selection based on dominated hypervolume," *European Journal of Operational Research*, vol. 181, no. 3, pp. 1653–1669, 2007.

[20] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, no. 2, pp. 115–148, 1995.

[21] J. Hagelbäck, M. Preuss, and B. Weber, "CIG 2010 StarCraft RTS AI Competition," 2010, http://ls11-www.cs.tu-dortmund.de/rts-competition/starcraft-cig2010/.