Noname manuscript No.
(will be inserted by the editor)

Controllable Procedural Map Generation via Multiobjective Evolution

Julian Togelius · Mike Preuss · Nicola Beume · Simon Wessing · Johan Hagelbäck · Georgios N. Yannakakis · Corrado Grappiolo

the date of receipt and acceptance should be inserted later

Abstract This paper shows how multiobjective evolutionary algorithms can be used to procedurally generate complete and playable maps for real-time strategy (RTS) games. We devise heuristic objective functions that measure properties of maps that impact important aspects of gameplay experience. To show the generality of our approach, we design two different evolvable map representations, one for an imaginary generic strategy game based on heightmaps, and one for the classic RTS game *StarCraft*. The effect of combining tuples or triples of the objective functions are investigated in systematic experiments, in particular which of the objectives are partially conflicting. A selection of generated maps are visually evaluated by a population of skilled StarCraft players, confirming that most of our objectives correspond to perceived gameplay qualities. Our method could be used to completely automate in-game controlled map generation, enabling player-adaptive games, or as a design support tool for human designers.

Keywords: Real-time strategy games, RTS, procedural content generation, evolutionary computation, multiobjective optimisation, *StarCraft*

1 Introduction and Motivation

Procedural content generation (PCG) refers to the automatic or semi-automatic generation of game content. PCG comes in many flavours, as there are many types of game

Julian Togelius · Georgios N. Yannakakis · Corrado Grappiolo
IT University of Copenhagen, 2300 Copenhagen S, Denmark
E-mail: julian@togelius.com, yannakakis@itu.dk, cogr@itu.dk

Mike Preuss · Nicola Beume · Simon Wessing
TU Dortmund, Otto-Hahn-Str. 14, Dortmund, Germany
E-mail: mike.preuss@tu-dortmund.de, nicola.beume@tu-dortmund.de, simon.wessing@tu-dortmund.de

Johan Hagelbäck
Blekinge Institute of Technology, Karlskrona, Sweden
E-mail: johan.hagelback@bth.se

content that can be generated (such as levels, adventures, characters, weapons, planets, plants, histories) and many ways in which the content can be generated (many of them based on methods from artificial intelligence (AI) or computational intelligence (CI), such as constraint satisfaction, planning or evolutionary computation, others based on e.g. fractals). PCG can also be used in different ways in games, for example for offline content creation during game development, in support tools for human designers or for 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 secondary storage space, academic curiosity or enabling completely new types of games. What is clear is that PCG is gaining increasing attention among both commercial game developers, indie developers and academic game researchers.

This paper presents a search-based approach to generating maps for real-time strategy games. More specifically, we use a multiobjective evolutionary algorithm to generate maps for both an imaginary generic strategy game based on heightmaps, and for the classic RTS game StarCraft, using objective 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, playable maps for strategy games. We have previously explored this method in two published papers [1,2]; this paper builds on those papers, and extends the work published there through adding experiments that combine three objectives rather than just two, through refining the objective functions, through rerunning most of the experiments and presenting a larger set of results, through analysing the results in more depth, through letting human players evaluate the results and through providing an extended background discussion.

We do not claim to have "solved" the problem of automatic strategy game map generation, as designing a suitably balanced map (especially a three-player map) is a hard task even for a skilled human designer; indeed, the maps we have been able to create have a number of areas to improve on, as evidenced by user studies. However, we believe our general approach is already more controllable and applicable than any other map generation algorithm available, and that with more refined constraints and heuristics it could produce professional-quality maps. Our main contributions are the general approach, two map representations, a number of reusable heuristics, and an in-depth study and evaluation of some attempts to evolve good strategy game maps using the proposed approach.

The paper is structured as follows: The next section gives a background to the research described in this paper, discussing the role of PCG for strategy games, the search-based approach to PCG and uses of multiobjective optimisation in games. In section 3, we present the two game domains we use for our experiments: an imaginary generic strategy game, and the classic RTS game *StarCraft*. Section 4 outlines some high-level design decisions. In section 5 we describe how we represent maps for both of these games. Section 6 first discusses our motivations for various objective functions to be optimised simultaneously by a multiobjective evolutionary algorithm (MOEA), followed by a description of the objective functions themselves. The missing puzzle piece before describing the experiments is the particular MOEA we used and its configuration (section 7). In section 8, we describe a systematic investigation

of the interplay of pairs and triples of the devised objective functions, in order to to clarify the design trade-offs in the problem and to ascertain the advantages of multi-objective techniques. Section 9 summarises the results of a user study, validating that our objective functions agree with perceived map design qualities. We conclude by discussing how this method can be used in some different content generation scenarios.

2 Preliminaries

2.1 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 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. These maps are usually created by professional map designers, having extensive experience of the game as well as key design considerations. 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 seeming dissimilar to previous maps she has played. In a multi-player game, maps might be generated that balance out the strengths of different players' playing styles and levels of proficiency, without resorting to explicit handicapping in terms of game rules or resources 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.

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 authoritative information has 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 parameters relate to the size and connectedness of the world's landmass; further, in the opinion of the authors, the resulting maps are often not very well balanced. Still, these maps are good enough, as Civilization poses very different challenges to an RTS such as StarCraft: Civilization is not usually played as a competitive game, and play sessions are extremely long, free-form and unpredictable.

A simple 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 [3]. Certain features on land, such as forest areas, can be created in the same way. Simple constraints, such as not connecting certain land areas to fill in canals, can easily be added.

Other approaches involve using fractals, such as the diamond-square algorithm [4]. The diamond-square algorithm works by iteratively subdividing areas of space and offsetting the midpoint by random amounts. Such algorithms are most commonly used with height maps to generate, for example, believable mountains. An advantage of this family of algorithms is that they are so fast that they can often be used for real-time terrain generation [5].

Recently, Doran and Parberry suggested the use of software agents for generating terrain [6]. 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. The agents are applied in phases, with coastline agents followed by smoothing agents, etc. This approach is claimed to be more controllable than fractal-based terrain generation algorithms.

In many cases, several different algorithms need to be combined in order to create a rich and detailed large-scale landscape. When such combinations of procedural terrain generation algorithms need to allow for human editing at various levels of detail, specific problems arise, such as how to retain human micro-level edits when re-generating macro-level features [7].

None of the above approaches take balancing of the map into account, and a map generated using any of these techniques is unlikely to satisfy a competitive strategy game player, as it would unfairly advantage one player or another.

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 [3].

2.2 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. A good example is Tarn Adams' ambitious game *Dwarf Fortress*, for which initial fractal map generation is often 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 *objective 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 objective function. An overview of SBPCG can be found in [8].

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. However, choosing the objective function and the search space carefully can allow the whole process to finish 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 objective 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 [9].

Sorenson and Pasquier evolve simple dungeon layouts for e.g. roguelike games, using a map representation where rooms and hallways of different sizes are placed on

a two-dimensional surface which is by default non-traversible. The objective function is simply the length from start to finish, and the only constraint that the path should be connected [10]. Similarly, Ashlock et al. evolved path-planning problems in which the objective was to maximise distance from start to finish by placing walls at various positions and angles [11].

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 part of the reason why the objective 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 [12]; Pedersen et al. evolved levels for Super Mario Bros based on an empirically derived model of player affect [13]; Hastings et al. evolved weapons for a 2D shooter based on player activity in the game [14]; Togelius and Schmidhuber evolved rulesets for predator-prey games [15]; and Browne evolved board games based on measures derived from studies of successful games [16]. None of these studies concerned maps or terrains, however. Further, they all used either a single objective function or an arithmetic or ordinal combination of several objective functions, yielding in effect a single objective.

2.3 Multiobjective Evolution

In standard evolutionary computation a single objective function is sought to optimise and therefore 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 optimise in three objective dimensions. In many cases, the objectives are *partially conflicting*, for example a faster car is typically less cheap.

The intuitive solution is to simply add the objective measures together (using some weighting of each measure), and optimise for the resulting composite measure. This method has several drawbacks. One is that it is hard for the user to articulate her preferences by appropriate weights; this does not become easier until the trade-off between objectives has been explored. Another is that optimisation along a single dimension does not allow for exploration of the often complicated ways in which the various objective dimensions interact (e.g., above a certain price threshold faster cars might not be less cheap). Depending on these interactions, some desired combinations of function values may be unreachable for any weighting. Furthermore, it is a well-known fact in multiobjective optimisation that for some problems, no weighted-sum single-objective approach can reach some existing optimal compromise solutions a multiobjective algorithm could attain.[1]

[1] This is the case if the set of optimal compromises, also called the Pareto front (please see next paragraph), has a concave shape. Das et al. [17] discuss the problem in more detail, simple examples are e.g. given by Koski [18].

Multiobjective evolutionary algorithms (MOEA) are state-of-the-art methods for multiobjective problems, and are now a major research direction within evolutionary computation as well as common in industrial applications. An MOEA presumes at least two objective functions that are partially conflicting, and proceeds to search for a *Pareto front*. The Pareto front is the set of Pareto-optimal solutions, i.e. solutions that cannot be improved in one objective without worsening in another; it contains all possible optimal compromises between the objectives. A solution is called *dominated* when there is another solution that is better in at least one objective and worse in none. Elements of the Pareto front are not dominated by definition as no dominating points exist. In practice, only an approximation of the Pareto front can be expected to be found by the MOEA. In this context, the term of a *nondominated set* is important: A nondominated set contains only solutions that have not been dominated by other solutions so far. The MOEA iteratively improves a nondominated set as an approximation to the Pareto front and its final set is the algorithms result presented to the user.

When using two or three objectives, the Pareto front (approximation) can be conveniently plotted as a graph, allowing visual exploration of the tradeoffs between these objectives. Visual or automated inspection of Pareto fronts helps to understand the space of design possibilities. For example, one can detect situations where a small loss in one objective would lead to a huge improvement in another, or the opposite. The possibility to visualise the tradeoffs inherent in a design problem makes multiobjective optimisation via MOEAs a great but as yet underused tool for design and authoring support.

More than three objectives are usually hard to handle, as the objective space grows exponentially with the number of objectives. On the other hand the fraction of points being comparable to a point (either better or worse in all objective values but not both) becomes exponentially small. This makes a progress towards the Pareto front quite hard and requires much resources, i.e. the generation and evaluation of many points, which might be too time-consuming in case of complex objective evaluation like simulations. Moreover, the interpretation of results becomes hard as it is problematic to visualise results in case of more than three objectives.

2.4 Multiobjective Evolution Applied to Games

Multiobjective evolution has been used for a number of different tasks in games, such as optimising controllers both for memory capacity and for playing well [19], optimising controllers for both playing well and playing and a human-like fashion [20], optimising several different measures of well-playing simultaneously [21] and for finding strategies that are well-performing yet behaviourally simple [22]. As far as we know, multiobjective techniques have not been used in procedural content generation before.

Optimising some aspect of a game for playability is inherently a multiobjective problem, as it is very hard to formulate a reliable single-dimensional algorithmic 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 invalu-

able for a designer to be able to conveniently visualise 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 her previous playing style and experience model. Additionally, variations from human-created solutions are desirable for a diversified game, and this is what evolutionary algorithms naturally are able to accomplish.

3 Game Domains

We test our algorithms by evolving maps for two different domains: an imaginary strategy game, containing some of the most common elements of strategy games, and the *StarCraft* RTS game. In both games, we assume that a map needs to include positions for player-controlled bases and positions for resources of different kinds. These features, or more or less isomorphic ones, are common to many strategy games of different types.

3.1 Generic strategy game

Our imaginary game has a key feature in common with many strategy games (including *Total War*, *Dawn of War* and several games created with the *Spring* engine), namely that the terrain is based on a height map capable of accommodating complex landscape features – especially hills and valleys of differing height and steepness. We suppose that elevation differences are associated with a movement penalty, so that moving up and down hillsides takes more time (or movement points) than moving along flat territory. There might or might not be visibility effects associated with the heightmap, such as units only being visible when in line of sight.

3.2 StarCraft

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. The game is famous for its exquisite balance between the different playable factions, and is 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 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 do

not 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.

StarCraft does not have hills and valleys like our imaginary strategy game above; the terrain is mostly flat. Instead of height maps, StarCraft is built on the notion of impassable and passable areas. Passable areas are those that ground troops can pass through, and impassable areas are elements such as rock formations and rivers, which cannot be passed by ground troops. Nevertheless, the illusion of passable mountain areas (plateaus) is created by painting the inner part of an area that is surrounded by impassable tiles in a different color and adding ramps. However, the movement is restricted in exactly the same way as if the impassable tiles were walls.

4 Meta-design considerations

Before designing the map representation and fitness functions, we had to decide on a number of high-level design questions that would delimit the space of possible maps we search. It is quite common in StarCraft and some other strategy games to create either two- or four-player maps, with one- or two-way symmetry, in order to guarantee the fairness of the map. In the opinion of the authors, symmetry makes a map more predictable (if you have seen a particular landscape feature close to your base, you can count on the enemy having an identical feature next to his own) and therefore less interesting. We reasoned that symmetry is a result of not having tools available for creating balanced asymmetric map. Therefore we decided not only to not enforce symmetry in the map representation, but also to generate three-player maps, where perfect symmetry is impossible and near-symmetry rather hard to achieve. The generation mechanism would have to find ways of creating asymmetric balanced maps.

It should be noted these high-level design decisions are not uncontroversial. As we will see, these decisions amounted to posing a design problem that would challenge even professional map designers, and which is beyond the capabilities of any known map generation algorithms.

5 Map Representation

The map representation for both domains (the imaginary strategy game and StarCraft) have many things in common, and differ mainly in the representation of terrain features. We start with what both representations have in common.

The naive map representation, laid out spatially like it would in the actual game, is unlikely to induce a good search space for evolutionary or other stochastic search algorithms, for reasons of dimensionality and locality. Therefore the evolutionary algorithm works on a *genotype* which is a somewhat indirect representation of the *phenotype*, the map which is used for objective testing and visualisation. The genotype

is about an order of magnitude smaller than than the phenotype in terms of memory size, and we believe it is also likely to induce a space that has better locality relative to several of our objectives. On the continuum of direct-indirect representations presented in [8], our representation would be at level one or two from the top (the "direct" end of the scale).

Each time objectives 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. These are the four types of elements encoded in the genotype:

- **Base:** ϕ and θ coordinates of each base.
- **Resource type 1:** x and y coordinates of each resource of type 1. In StarCraft, this translates to a mineral source.
- **Resource type 2:** x and y coordinates of each resource of type 2. In StarCraft, this translates to a well for vespene gas.
- **Terrain features**. The representation of these differ between the two game domains, but in both cases each terrain element is defined by 5 floating point values.

For the generic strategy game domain, we generate maps with 3 bases, 4 resources of each type and 10 terrain features, leading to genotypes of length $3 \cdot 2 + 4 \cdot 2 + 4 \cdot 2 + 5 \cdot 10 = 72$. For the StarCraft domain, we use 8 mineral fields and 7 vespene gas fields (minerals are more important when the game starts, vespene for later stages), leading to genotypes of length 86.

The indirect representation has the advantage that it can be efficiently searched by many common global optimisation algorithms, such as evolution strategies and particle swarm optimisation. 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. (See more about representation considerations in SBPCG in section 3.1 of [23].) However, one shall consider that even this very condensed map representation leads to relatively large genotypes ($\approx$ 50 to 100 real-valued variables), so that search in this large space is not trivial. This may be a reason why automated map creation has been tried only rarely in the past.

The phenotype (direct) representation is a spatial representation similar to how the map would be represented in the actual game engine. This representation consists of a two-dimensional array detailing the terrain, and lists of the x and y positions of all bases and resources. The terrain array is constructed very differently in the two domains, but the base and resource locations are generated in the same way.

Resource locations in the phenotype are generated by simply multiplying the x and y values of each resource in the genome with the height and width of the terrain array. Base placement is a bit more involved. 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 be 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.

5.1 Generic strategy game terrain representation

In the generic strategy game, the terrain features are mountains. For each mountain we consider the two standard deviations (σ_x and σ_y) of a three-dimensional Gaussian distribution with a mean $[x, y]$ (representing the coordinates of the Gaussian mountain peak); and a weighting parameter, h, that adjusts the height of the Gaussian surface. The terrain array has size 100×100, and each cell can take on a discrete number between 0 and 99 representing elevation at that point. All cells of the heightmap are initially set to elevation zero.

The mountains are then drawn as Gaussian curves in two dimensions. The peak (x and y values for the mountain in the genome multiplied by 100) is elevated to the height set for that mountain (multiplied by the height parameter, h — h is 99 in these experiments). The standard deviation values along the x and y axes (σ_x and σ_y) are calculated by multiplying the corresponding value in the genome by 10. For cells that are affected by more than one Gaussian 3D bell, the highest value from any of them is used in the phenotype (final map).

5.2 StarCraft terrain representation

When generating StarCraft maps, the terrain array has size 64×64 (the standard size for a StarCraft skirmish map).

The five real numbers that define each terrain feature are interpreted as starting position (x, y), left and right turn probabilities, and pen lifting probability. All cells of each map phenotype are by default passable. Impassable areas are then "drawn" in a manner similar to turtle graphics [24]. 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. Checking left turns first consistently is done to enlarge the chance that the resulting curve is closed. However, as it still often happens that the resulting line is not closed (especially if the left turn probability is low), 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 deterministic genotype to phenotype mapping, a fixed random number table with 200 entries is used to decide whether to turn and/or leave gaps. (Non-deterministic genotype to phenotype mappings are known to induce significant evaluation noise [23].)

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[2] 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.

6 Evaluation Functions for Map Generation

In SBPCG, there is a distinction among three types of evaluation functions: *interactive*, *simulation-based* and *direct* [8]. Interactive evaluation 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 evaluation function, interactive evaluation functions require very large amounts of player input and are only possible in some types of games, such as ongoing massively multiplayer games [14]. Simulation-based evaluation 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 [12, 15]. Direct evaluation functions base their fitness calculations directly on the phenotype representation of the content. Such evaluation functions are obviously much easier to implement and faster to compute than simulation-based functions, but it is hard to devise direct objective functions that accurately predict key aspects of player experience (except when basing them on data-driven player models built from extensive user studies [13]).

In this paper, we will not attempt full simulation-based evaluation functions, as we do not have access to any game engine for our imaginary generic strategy game, and the StarCraft game is proprietary, closed source and does not have a satisfactory API. Even if we could script StarCraft to test aspects of our levels through automated playthrough, this would be prohibitively time-consuming as StarCraft cannot be sped

[2] available at http://www.clanscag.com

up to run much faster than real-time (this goes for most commercial games), and most evolutionary runs would need tens of thousands of objective evaluations. This is also why we do not use any interactive objective functions; we do not have access to enough cheap labor to manually play through and evaluate masses of algorithmically generated maps, especially those maps that would be considered "errors" in the trial-and-error process of evolutionary computation.

However, we can simulate one key aspect of RTS gameplay: moving between two points along the shortest 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 the ength of shortest path found by A* in the rest of the paper.

But this only answers the "how" question in relation to objective function design, not the "what" question: what sort of maps do we want to create? We agreed on a number of desirable characteristics of good strategy game maps, in the sense that they create conditions for enjoyable gameplay.

- *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).

These characteristics can be related to a number of theories of what constitutes enjoyable game experiences. For example, Malone analyses fun in gameplay into its components *challenge* (the right amount of it), *fantasy* and *curiosity* [25]. Czikszentmihalyi's *Flow* theory also centers on having the right amount of challenge [26], wheras Koster's "Theory of fun for game design" is fundamentally based on *learnability*, meaning that the player constantly improves aspects of his/her gameplay [27].

In terms of these theories, playability is of course strongly related to challenge, in addition to operating on a level below (and presupposed by) the aforementioned theories; fairness to both challenge (playing against a vastly superior or inferior enemy leads to a challenge imbalance) and learnability (playing against someone who, adjusted for superior/inferior map, is about your own strength encourages learning); skill differentiation to learnability and curiosity (encouraging players to try out new strategies); and interestingness to fantasy and curiosity.

We defined a number of different objective measures (mainly based on distance) for both the generic strategy game and for StarCraft, intended to reflect the desired map characteristics outlined above. It was at the time of their formulation not clear to which degree the various functions conflicted or induced searchable objective landscapes. The experiments in this paper investigate the interplay of *pairs* and *triples* of these functions, as it is computationally infeasible to optimise for all of the functions at the same time. All objective functions are to be maximised and are normalised to values in $[0, 1]$.

6.1 Generic Strategy Game Evaluation Functions

On generic strategy game maps, the A* algorithm measures the *weighted distance* between points. In our formulation, each transition between any two cells has a cost of 5 plus the difference in elevation between the two cells. This takes elevation changes into account and means that the shortest path between two points might mean going around a mountain or valley, even if the path straight across the mountain or valley would result in fewer cells traversed. We defined the following functions for generic strategy game maps:

- f_0*: Base distance*. The f_0 function is calculated as the average weighted distance between bases.
  Motivation: fairness and interestingness. For multiplayer games, all players should have bases at approximately the same effective distance from each other (either this means they are separated by long expanses of plains, or by mountain peaks). Bases should be not be too easily reachable from each other, to avoid too short games.
- f_1*: Base on ground.* The f_1 function promotes low elevation for bases and is expressed as: $f_1 = 1 - \sum_i \{h_i^B / N_B\}$, where h_i^B is the elevation of base i and N_B is the number of bases considered.
  Motivation: playability and fairness. Bases should be placed on flat areas to allow placement of adjacent buildings and spatial allocation of newly produced units. Bases should all be placed on the same elevation to avoid unfair advantages (cf. Masada).
- f_2*: Asymmetry*. The f_2 function corresponds to the average elevation difference between a strategically chosen cell (at position (w, h) where w is map width divided by 4 and h is map height divided by 4) and its counterparts on the opposite half of the grid in both x and y axes $(w, 3h)$, $(3w, h)$, $(3w, 3h)$.
  Motivation: interestingness. Symmetric maps might look artificial and boring, and if symmetry is common among produced maps (if the generating algorithm displays a preference for this) players might come to count on the same feature (base, mountain or resource) be available on the opposite side of the grid and adjust their strategies accordingly.
- f_3*: Resource distance*. The f_3 function is expressed as $f_3 = 1 - (\max\{D^R\} - \min\{D^R\}$, where $\max\{D^R\}$ and $\min\{D^R\}$ are, respectively, the maximum and minimum distances from any base to their nearest resource of any type.
  Motivation: fairness. All bases should have the same access to resources.
- f_4*: Resource clustering*. Function f_4 expresses the spatial diversity of resources within a map (within a number of *meta-cells*) and it is calculated via Shannon's entropy formula: $f_4 = -(1/logC) \sum_i (r_i/R) log(r_i/R)$, where c is the number of meta-cells the map is divided upon; r_i is the number of resources on meta-cell i and R is the total number or resources available. In this study, the map is divided into 9 square meta-cells, so that the first meta-cell contains all cells between $(0, 0)$ and $(32, 32)$ etc.
  Motivation: interestingness and skill differentiation. Maps where resources are clustered together ($f_4 \approx 1$) motivates some players to explore more, and gives

them more surprises; they also allow more skillful players to take advantage of their superior tactical knowledge by deciding when to explore and which areas to defend.

6.2 StarCraft Objective Functions

Based on the experiences gained from the map generator for the imaginary RTS game, we further develop objective functions for StarCraft.

On StarCraft maps, the A* algorithm simply measures the number of cells along the shortest path between two points, not traversing any impassable areas. As the existence of impassable areas may result in unplayable maps, we designed a simple "sanity check" that is executed before any objective function is run. This test ensures 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 value of 0 in all objectives, effectively discarding it. It should be noted that this constraint precludes the generation of "air war" maps, where the players can only reach others' bases using aircraft.

6.2.1 Base placement functions

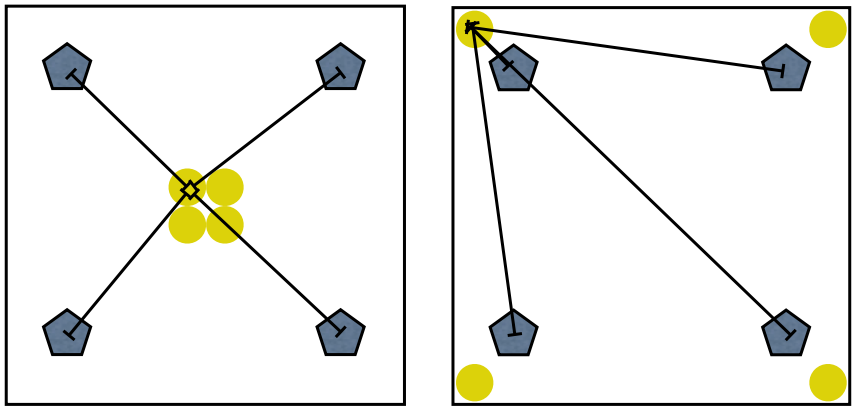
The first two objective functions relate mainly to the properties of the placement of players' starting bases, and to the impassable area around and between bases.

- f_{b0}*: Base space*. For playability, some space for other buildings is required next to the base. Out of the 5*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 objective 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 each player has 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.

6.2.2 Resource placement functions

The next four objective 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 safety*. Another measure of how clearly resources are assigned to a single player, f_{r2} 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 $r_1, ..., r_m$



(a) unsafe resources (b) safe resources

Fig. 1: Safe and unsafe resources. Bases are depicted by pentagons, resources as circles. The lines mark shortest possible paths for attackers/defenders.

we calculate all path lengths between resources and bases and group them by resource type:

$$\forall j = 1, \ldots, m : D_j = \{\mathrm{dist}(r_j, b_i) \mid i = 1, \ldots, n\}\,.$$

$f_{r2} = \min\{s_{\text{Gas}}, s_{\text{Minerals}}\}$, where s_{Gas} and s_{Minerals} are simply the average standard deviations of the respective sets D_j.

6.2.3 Path functions

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

- f_{p1}*: Path overlapping*. We consider the paths from the bases to all resources and calculate to what extent paths of different players overlap. In case many cells are used from different bases we assume that the players' units are likely to meet. The value of f_{p1} is the average number of users of cells belonging to a path. It contributes to skill differentiation, as it increases the number of possible flash points which the player must monitor for conflicts. To produce maps with few interaction for unexperienced players, we also optimise in the inverse direction (low values of f_{p1}) and which we denote as function f_{-p1}.
- f_{p2}*: Choke points*. We consider the average narrowest gap on all paths between bases. The narrowest gap along a path from A to B is calculated by first calculating a shortest path and then traversing along the path and counting the width of the path at each cell. Gap 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. If the narrowest gap is less than 10 cells wide, it is deemed a choke point. A copy of the map is

then made, and this gap is filled in with impassable cells on the copy of the map. A new attempt is then made to find the shortest path from A to B, and if a path still exists, the increase in length between the new and the old path is recorded. The choke points function for a pair of bases is calculated as:

$$0.5 \cdot (10 - g) + 0.5 \cdot \begin{cases} 0.5 \text{ if no new path is found,} \\ d/w \text{ otherwise.} \end{cases}$$

Where g is the width of the narrowest gap in the original path, d is the difference in length between the new and old path, and w is twice the diagonal length of the map.
Choke points contribute to skill differentiation in that a good player might be able to exploit such points by 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. Here, we also consider the inverse function f_{-p2} to create easy maps.

7 Optimisation by Multiobjective Evolutionary Algorithm

Most MOEAs work similarly. A population of search points (called *individuals*) 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. Like other evolutionary algorithms, MOEAs are *black box* algorithms, meaning that they do not rely on explicit domain knowledge. The most popular and long-established MOEA, NSGA-II [28], has proven 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 [29].

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 part of the 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 equalling 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 the NSGA-II standard recombination/mutation operators *simulated binary crossover* (SBX) and *polynomial mutation* (PM) from [28] with (near) default parameter values of $\eta_c = 20$ and $\eta_m = 15$[3]. SBX has been introduced in [30] and is based on a polynomial distribution that is 1-centered and that is the flatter, the lower the respective η value is. While in SBX, the distribution is applied as multiplier to the

[3] In [28], both parameters have been set to 20, other authors use 20 and 15. However, the difference in algorithm behavior is most likely negligible.

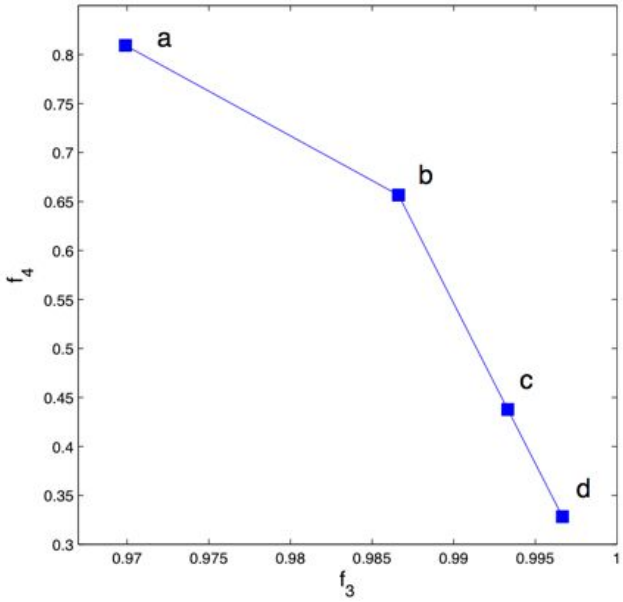


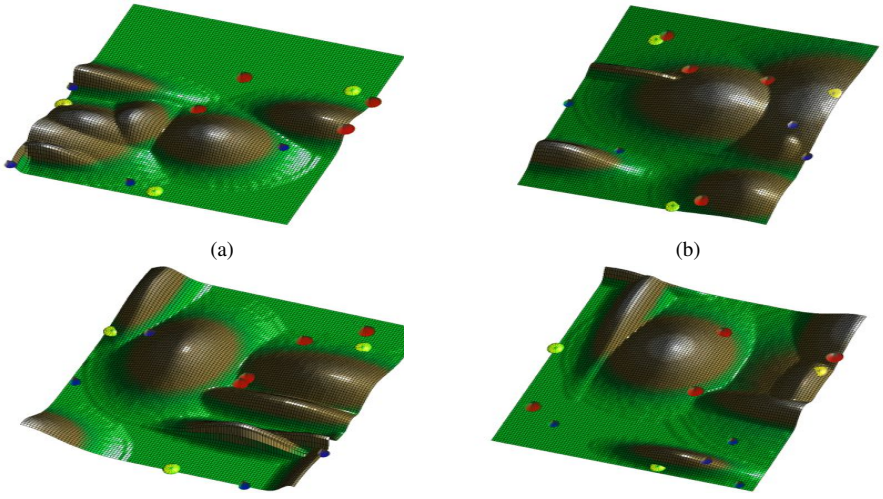
Fig. 2: Pareto front approximation for the objective pair f_3-f_4. The solutions a, b, c, d correspond to the 4 maps in Fig. 3.

difference of the parents (each variable separately), the mutation follows the same scheme but works directly on the variable values of one individual.

After some testing, the run length was fixed to $50,000$ evaluations for the generic strategy game and to $100,000$ evaluations for StarCraft. Small further progress after this time is still observed sometimes but considered irrelevant. In all experiments, we use populations of 20 individuals, which we consider sufficient to achieve a reasonable approximation of the Pareto front as we are only interested in a small number of resulting maps and a rough impression of the front. Increasing the population size will increase the runtime (in evaluations) at least linearly, leading to unacceptable waiting times if we think of applying the technique as supportive method in a map design context.

8 Experiments

We performed a large number of experiments using both game domains in order to find partial conflicts between objective functions and generate interesting Pareto fronts. In order to investigate whether multiobjective evolution can provide a tangible advantage over other optimisation techniques, we need to know whether there exist partial conflicts between the objectives, meaning that a tradeoff will need to be made in optimising two objectives simultaneously. If it is further found that the individual objectives correspond to desirable properties of maps (as will be investigated in a user study for the StarCraft domain) this is a strong indication that tradeoffs between



(a) (b) (c) (d)

Fig. 3: The four generated maps taken from the Pareto front approximation for the objective pair f_3-f_4 displayed in Figure 2. Bases are illustrated as yellow spheres; resources are depicted as either red (type 1) or blue (type 2) cones.

different desirable qualities fundamentally exist in the map design problem itself. The investigation of degree of conflicts between objectives in this section therefore serves both to clarify the usefulness of multiobjective techniques for designing maps, and indirectly to investigate properties of the underlying design problem. The experiments in this section are ordered by game domain (generic strategy game versus StarCraft) and number of objectives (2 versus 3).

8.1 Generic strategy game

For the generic strategy game, we only explored the interplay of pairs of objectives. The experiments were chiefly concerned with finding which pairs of objectives exhibit partial conflicts. Therefore multiple evolutionary runs were done with 12 pairs of the 5 objectives, and the resulting Pareto fronts exhibited.

Partial conflicts (as indicated by substantial Pareto fronts) were found between objective pairs (f_1 (base on ground) and f_2 (asymmetry)), (f_1 and f_3 (resource distance)), (f_2 and f_3) and (f_3 and f_4 (resource clustering)). The resulting Pareto front for objective pair f_3 and f_4 is shown in Figure 2. Four maps taken from that front are depicted in Figure 3.

These conflicts can all be explained in qualitative terms. For example, the easiest way of optimising base on ground is to simply remove all mountains – this makes sure that all bases are at elevation zero. But this also makes for a completely topologically symmetrical map. Almost all additions of mountains to the map reduce the symmetry (increase asymmetry score) but most such additions will also elevate some

base, reducing the base on ground score. Of course, some configurations of mountains exist where asymmetry is high (though probably not maximal) while all bases are on ground, but such configurations are hard to find – this is why the conflict is *partial* between the objectives. A similar explanation can be given for the conflict between resource distance and resource clustering, which is visualised in the figures referred to above: in most configurations where all bases have the same access to resources, the resources are by necessity quite far from each other, so clustering is low.

It appears from these pictures that the algorithm finds maps that are interestingly different from one end of the Pareto front to the next. In particular, the heightmap-based representation turns out to provide a relatively high locality in the spaces defined by the various fitness functions, a crucial component of evolvability. Any judgment about playability must be qualified by the fact that the maps are not created for any game in particular. Still, the various fitness functions and constraints on the terrain generation contain an implicit game design sketch, which could relatively easily be fleshed out to a full game.

8.2 StarCraft

The most extensive set of experiments concerned maps for the *StarCraft* game. Before the main investigation of tradeoffs between our objective functions, we performed initial exploratory studies check whether functions were possible to optimise or trivial on their own, and whether there seemed to be conflicts with other objectives at all. Both base placement functions (f_{b0} and f_{b1}) were very simple to optimise to maximal or near-maximal values, so they are included as constraints; maps with less than 0.5 on any of these functions are discarded immediately. Additionally, f_{b1} is used as an objective in its own right. These initial experiments were followed by a systematic exploration of the search space induced by our representation and fitness function.

8.2.1 Two-Objective Experiments

Table 1: Average number of individuals in the final non-dominated fronts for each function combination.

	f_{r1}	f_{r2}	f_{p1}	f_{-p1}	f_{p2}	f_{-p2}
f_{b1}	8.3636	6.5455	7.4545	9.7273	4.4545	7.3636
f_{r1}		4.0909	2.4545	3.7273	3.3636	1.5455
f_{r2}			5.1818	2.5455	2.8182	2.4545
f_{p1}				17.0909	3.3636	1.0000
f_{-p1}					3.0909	2.2727
f_{p2}						17.0909

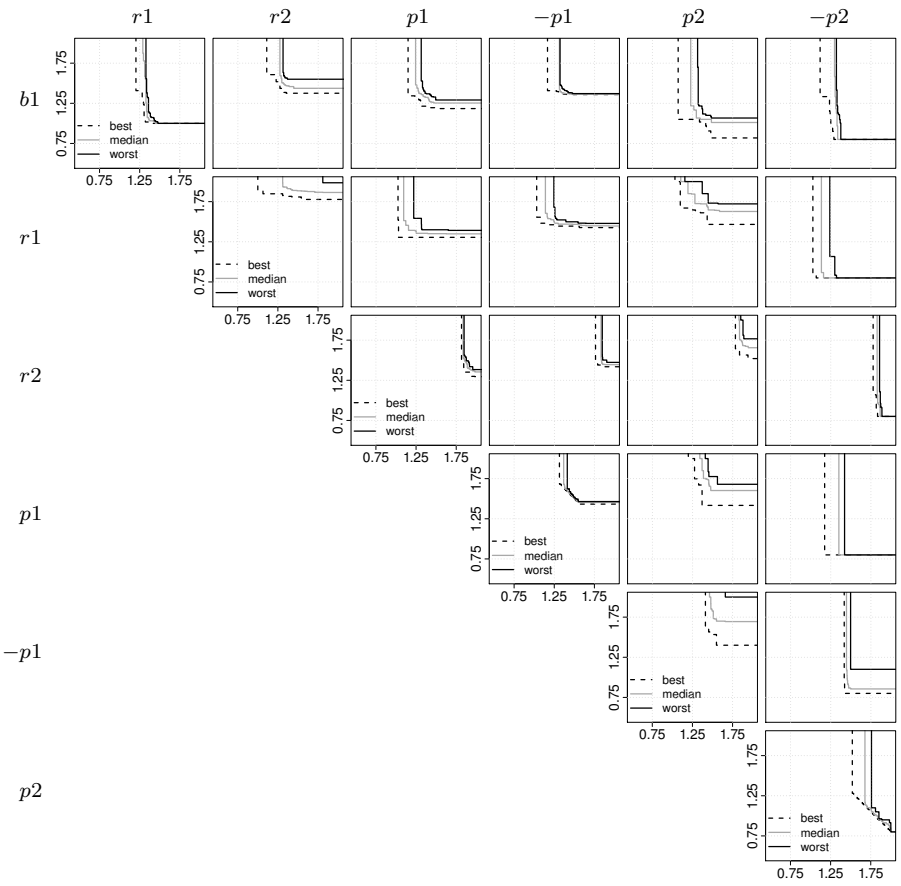


Fig. 4: Estimated attainment function of the 2-dimensional experiments. The columns and rows have the following order of function: $f_{b1}, f_{r1}, f_{r2}, f_{p1}, f_{-p1}, f_{p2}, f_{-p2}$.

The aim of our main 2-objective study was to find out the degree of conflict between the map objectives we developed. We performed runs with all pairs of those objectives that were non-trivial to optimise on their own, with the aim of revealing trade-offs between them. For each pair 11 runs were performed. The results can be seen in tables 1 and 2, using two different indicators of the degree of conflict.

Table 1 shows the average sizes of the final Pareto front approximations, i.e. the number of non-dominated solutions in the last generation. Small fronts are usually indicators of a low degree of conflict. For the pairs of opposing functions f_{p1}, f_{-p1} and f_{p2}, f_{-p2} all points are Pareto-optimal and so a large number of points showing different trade-offs are obtained. Table 2 shows the hypervolume of the final non-dominated sets relative to the reference point $(2, 2)$. For this indicator, *low* values indicate *high* degrees of conflict. (A value of 2.0 indicates that both objectives can be maximised simultaneously.) A strong conflict seem to exist for f_{r2} with f_{p1}, f_{-p1} and f_{p2}. A weak conflict can be observed for f_{r1} and f_{-p2}.

Table 2: Average hypervolume values of the final non-dominated fronts for each function combination.

	f_{r1}	f_{r2}	f_{p1}	f_{-p1}	f_{p2}	f_{-p2}
f_{b1}	0.6858	0.4076	0.5512	0.4457	0.7680	0.8520
f_{r1}		0.1379	0.5668	0.4553	0.2705	1.0380
f_{r2}			0.0978	0.0982	0.0563	0.2047
f_{p1}				0.3290	0.2401	0.7938
f_{-p1}					0.2123	0.6729
f_{p2}						0.3551

Figure 4 shows the *estimated attainment function* (EAF) for each pair of objectives. An EAF is an approximation of the shape of a pareto front based on density functions [31,32]. The size of dominated area (around the upper right corner) corresponded to the hypervolume of the front.

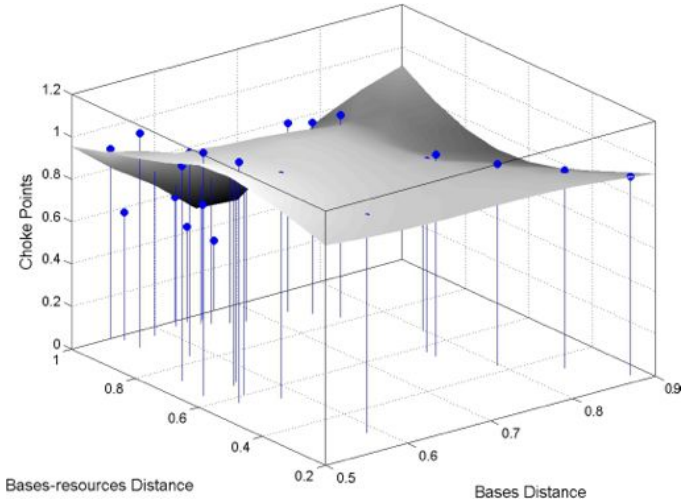
The structure of the matrix in Fig. 4 equals the structure of Tables 1 and 2. A strong conflict can be observed regarding f_{r2}, f_{p2}. Here, all the function values are big (bad) compared to the better values achieved in combination with other objectives. It can concluded that one objective prevents the improvement of the other competing one. Weak conflicts seem to exists among f_{r1}, f_{-p2} since the values of both function reach very good values, better than in combination with other objectives. When completely contrary objectives are optimised, like for f_{p1}, f_{-p1}, f_{p2}, f_{-p2}, the Pareto front approximation is a line that show possible values of the functions.

8.2.2 Three-Objective Experiments

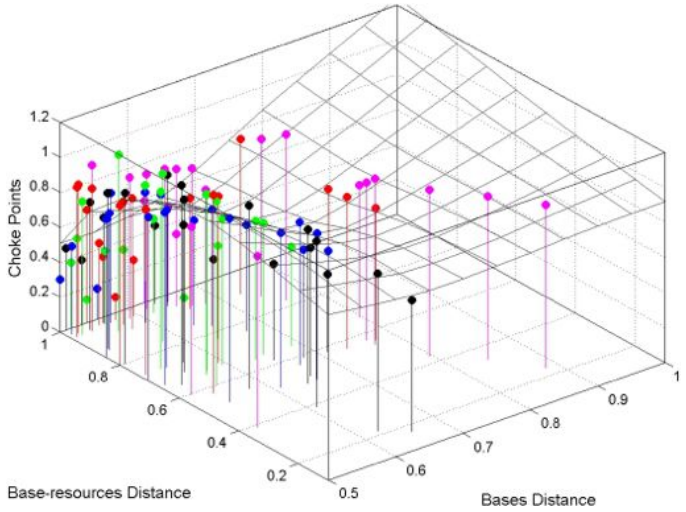
It would be infeasible to do an exhaustive study of the conflicts within all possible triples of objectives, both because of the computation time required to produce the results and the effort required to analyse them. Therefore, based on the results of the two-objective runs, we selected three interesting objective functions to be used in three-objective runs. We are here trying to find a triple of objectives where each objective partially conflicts with each of the other two, and where the three objectives are still relatively orthogonal to each other in terms of what they measure, reasoning that such combinations of objectives give rise to the most useful and meaningful Pareto fronts in design space.

As objective functions we chose the base distance f_{b1}, typed bases-resource distance f_{r1}, and choke points f_{p2}. Additionally, the base distance is also a constraint so that only maps with $f_{b1} \geq 0.5$ are valid, ensuring that the starting positions are relatively fair. These three objectives represent all 3 function groups and may be considered a good choice as the first two ensure that playable maps result and the third one strives for interesting maps. However, one may also exchange e.g. f_{r1} with f_{r2} or f_{p2} with f_{p1}.

Figure 5 depicts the resulting fronts of 5 independent runs (lower figure) and the non-dominated points attained from the composition of all fronts (upper figure). Values of $f_{b1} < 0.5$ would make a map invalid and are not part of the final front of any run. In the lower figure, single run fronts have different colours, showing that



(a) Non-dominated points of 5 runs



(b) The final fronts of the same 5 runs

Fig. 5: Optimisation in 3 dimensions: Base distance f_{b1}, bases-resource distance f_{r1}, and choke points f_{p2}. The upper figure shows only the non-dominated points of the aggregated set of the 5 fronts, the lower figure all points of the fronts. Note the slightly different scaling.

they are quite diverse. This is probably an effect of the very large search space, and similar to what has been obtained in a recent real-world multiple criteria optimisation problem investigation [33]. It seems that there are many ways to achieve the same objective function values, and that it depends on the starting population of each run to which area of the solution space it converges. A general treatment of such different

front realisations has been given in [34]. Nearly all runs ended with a front of size 20 (the population size), meaning that there is a considerable amount of conflict between the objectives.

Analysing the results, it seems that the choke points objective and the base-resources distance objective are strongly in conflict, regardless of base distance values. For base distance against base-resources distance, there seems to be a weaker conflict which interacts with the choke points objective, so that for certain choke point function values, a much better front becomes available. However, base distance and choke points seem to be in very weak conflict only as it is possible to reach near optimal solutions in both at the same time. In figures 6 and 7, we depict 10 example maps obtained from a single run, please see the next section for more details. However, we would like to note that the map style looks quite different from manually crafted maps: the maps are strongly asymmetric and feature large linear or areal blocks of impassable areas. Given that it is impossible to design perfectly symmetrical square 3-player maps, the lack of symmetry is not surprising.

8.2.3 Testing in the Wild

Some of the evolved maps were used for the first StarCraft AI competition [35] at the 2010 IEEE Conference on Computational Intelligence and Games. In this competition, competitors pit their best StarCraft AIs against each other and the objective is simply to write the AI code that survives longest. Though technical issues prevented a conclusive winner from being found, the results of using evolved maps were encouraging.

9 User study

After investigating the search space induced by our representation and evaluation functions, the partial conflict between our objectives and the feasibility of evolving complete maps, we needed to investigate whether the objective functions actually corresponded to perceived qualities of map design. The best judges of the existence of such qualities in maps ought to be experienced StarCraft players, who have seen and played a large variety of maps, presumably of different quality. In order to reach out to a sizable number of experienced StarCraft players, and in order to be able to conduct the survey within reasonable time, we would need to use an Internet-based survey. (Using local students would mean a population of less experienced players, and therefore presumably less reliable answers.) Due to the technical, legal and other problems with letting survey participants play the maps in the actual StarCraft game engine, we opted to simply let participants view images of the generated maps. Overall, our study attracted 147 participants of which 7 (roughly 5%) were female. The vast majority (133) of participants were between 16 and 30 years old, and 134 (91%) saw themselves as experienced StarCraft gamers.

While the user study was designed to investigate the correlation between objectives and specific traits of the maps, as a side effect we obtained the opinions of

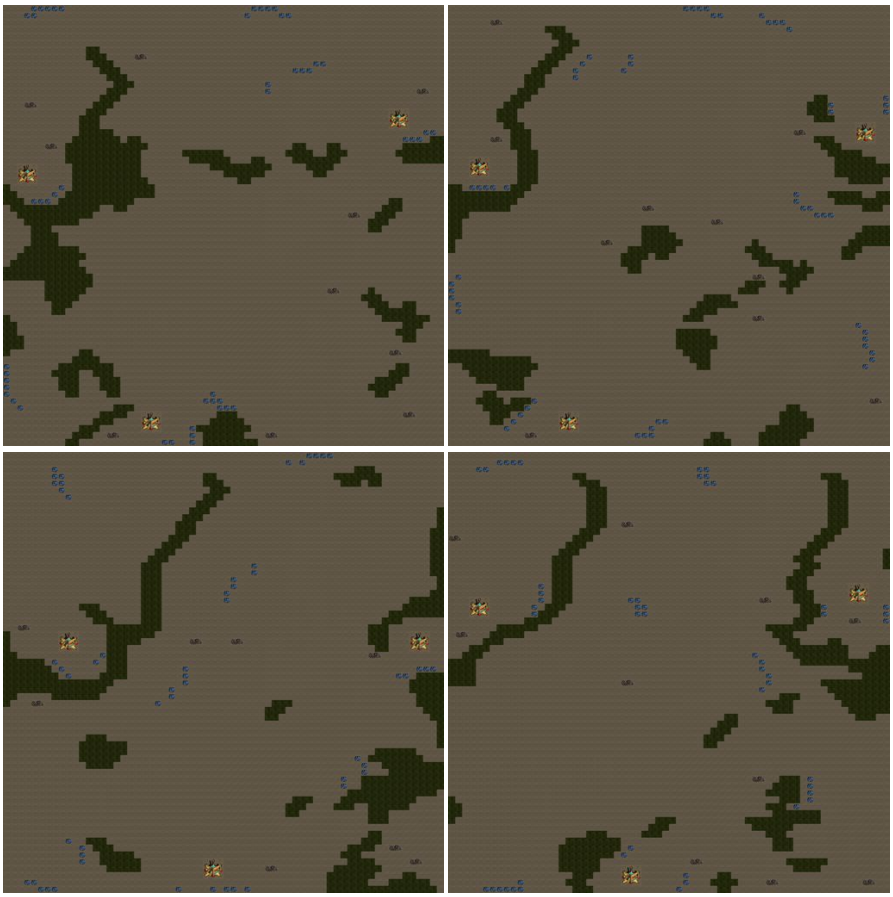


Fig. 6: First 4 of 10 example maps generated from the Pareto front of one 3D run (see table 3 for function values); these maps were in the user study

experienced players about the overall quality of the maps, as related to the (presumably high quality) maps usually enjoyed by the participants. The reader is advised to bear in mind that the objective of our research is not at this stage to rival the capacities and performance of professional human map designers, even though we hope that a system built on the methods we explore here will one day be able to do so.

The study took the following form: participants were first presented with a page of instructions, including a legend of the maps. They were then presented with a brief demographic questionnaire, including questions about whether and how often they played strategy games. The main part of the survey consisted of a single web page including ten different maps, so that the participants could easily compare the maps with each other, and a number of questions concerning each map. The same ten maps were used for all participants, but their order was randomised between participants. The number ten is a compromise between the need for statistical significance (more

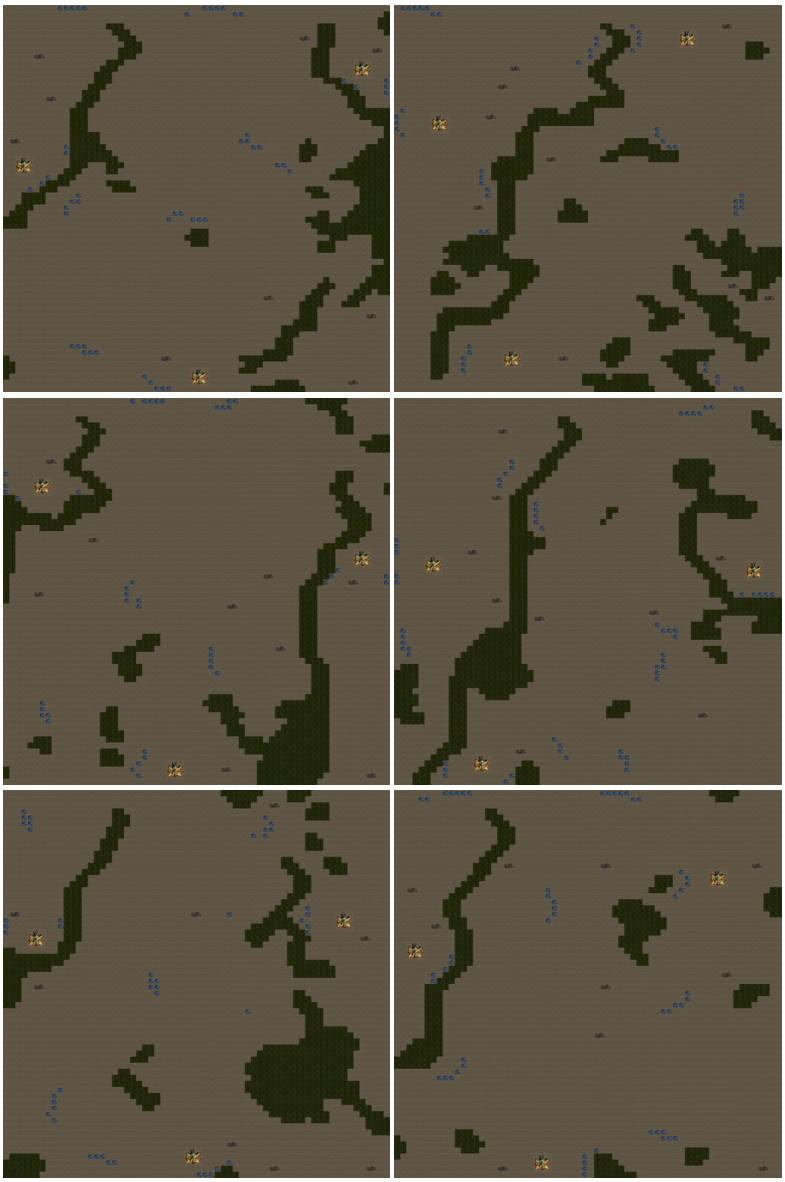


Fig. 7: Maps 5 to 10 of the 10 example maps used in the user study

maps) and user fatigue (less maps). All employed maps were taken from the last population of one 3-objective optimisation run using the same objectives as for our presented 3-objective study (sec. 8.2.2), namely base distance f_{b1}, typed bases resource distance f_{r1}, and choke points f_{p2}. We chose the ten most extreme maps in order to

allow the users to recognise the differences easily. (These were not necessarily the maps that "looked best" to us.)

For each map, three forced-choice questions were asked, ten tags could be applied and the participant was given the option to write a free-text comment on the map. The main questions were whether the participant agreed with the statements "the map has a fair resource distribution", "the map has a fair base starting point distribution" and "the map has choke points"; each of these questions had to be answered with "yes", "no" or "I don't know" for each of the maps before the user could submit the survey form (following the recommendations in [36]). The ten tags were simple check boxes, which the participant could choose to check none or as many as they wanted for each map; the tags were "interesting", "fun", "good gameplay", "engaging", "immersive", "boring", "frustrating", "challenging", "fair" and "symmetrical".

In order to collect both experienced and inexperienced StarCraft players as participants in the survey, we advertised it widely using social networking sites (*Facebook*, *Twitter* and *Google Plus*), blog posts, emails to academic mailing lists and the message board of the StarCraft enthusiast site *Team Liquid*. In addition to survey participation from many highly experienced players, the Team Liquid post got 47 replies from players commenting on our maps in detail[4].

Most of the commenters on our experiments at Team Liquid appreciated the effort to automatically create StarCraft maps (claiming that it would be very useful for players if we succeeded) but said that the generated maps were not very good maps at all. In particular, a very common opinion was that the maps could not be balanced because they were not symmetrical; any balanced map, in the opinion of these players, must be symmetrical. Additionally, the decision to focus on three-player maps was frequently criticised, as it is very hard to make these maps symmetrical.

The results of the survey can be seen in tables 3 and 4, and the most important correlations from the statistical evaluation are depicted in table 5. We performed two different statistical tests. At first, it is important to know if the obtained user answers are at all different for the different maps in a statistical sense, or if the user answer variations can be explained by noise. We investigated this by means of the `prop.test` in the statistical software `R`, which "can be used for testing the null hypothesis that the proportions (probabilities of success) in several groups are the same"[5]. The null hypothesis is that all measured proportions (one for each map) concerning a specific property (e.g. base fairness) stem from variations of a single fixed probability. According to table 5, there is a clear influence for resource fairness and choke points (p-values around 10^{-6}, whether there is none for base fairness. Note that our sample size (10) is relatively small, so that differences must be relatively strong for the test to get significant. However, it is clear that base fairness is not perceived as being very different for the maps, and that tag values are not significantly different between maps.

The second statistical test is a non-parametric correlation test after Kendall (again we employed an `R` method, in this case `cor.test`) that compares the ranks induced

[4] The complete thread can be found at `http://www.teamliquid.net/forum/viewmessage.php?topic_id=245185\&currentpage=All`

[5] The manual is available e.g. here: `http://stat.ethz.ch/R-manual/R-patched/library/stats/html/prop.test.html`, the test goes back to a paper by Wilson [37]

Table 3: Function values and normalized user answers (from 147 users) to the main questions. The fBase, fRes, and fChoke values give the objective values measured while producing each map, namely for base fairness, resource fairness, and choke points. Note that we minimise here, where 1.00 is the attainable minimum and 2.00 is the maximum. The next 3 columns give the fraction of users that answered yes to the main questions as corresponding to the function values: fair base starting point distribution (*base*), fair resource starting point distribution (*res*), contains choke points (*choke*.)

Map	fBase	fRes	fChoke	base	res	choke
1	1.38	1.00	1.42	0.180	0.144	0.583
2	1.23	1.46	1.05	0.160	0.118	0.625
3	1.48	1.40	1.04	0.289	0.127	0.641
4	1.20	1.57	1.05	0.091	0.224	0.783
5	1.16	1.66	1.07	0.209	0.158	0.655
6	1.48	1.61	1.02	0.121	0.191	0.674
7	1.24	1.18	1.05	0.191	0.227	0.787
8	1.12	1.76	1.07	0.155	0.169	0.739
9	1.45	1.00	1.11	0.296	0.211	0.697
10	1.33	1.00	1.67	0.238	0.168	0.510

Table 4: Tag frequencies provided by the 147 users for the 10 example maps. Legend: interesting (*intr*), fun (*fun*), good gameplay (*ggp*), engaging (*eng*), immersive (*imm*), boring (*bor*), frustrating (*frus*), challenging (*chall*), fair (*fair*), symmetrical (*sym*). Note that some maps lacked some tag values for technical reasons.)

Map	intr	fun	ggp	eng	imm	bor	frus	chall	fair	sym
1	15	4	2	2	0	30	46	12	2	0
2	14	6	1	1	3	27	46	20	1	0
3	19	0	2	2	1	27	51	21	3	0
4	19	4	3	1	0	30	48	19	1	1
5	15	1	1	2	0	35	56	18	1	2
6	22	7	4	3	2	28	44	17	2	2
7	19	5	3	7	1	30	43	17	2	1
8	27	4	1	2	0	26	49	17	3	0
9	19	2	3	2	1	25	38	20	2	2
10	11	3	2	5	0	36	52	20	2	1

by ordering after an objective value and one based on its corresponding user answer. We thus feed the test with the 10 values from each of the corresponding columns in table 3. The result is similar to the one above, namely that there is a relatively strong anti-correlation between objective values (which were formulated for minimisation) and user answers (questions posed for maximisation) for the same two cases resource fairness and choke points. As we have very few samples (10), the correlation tests themselves do not become significant, but they are not very far from doing so. The resulting correlations are near to what is usually regarded as strong anti-correlation (around -0.4) in psychology, thus the user feedback is in remarkably strong agreement to the measured objective function values.

Table 5: Relation of user feedback data to map specific objective function values. The column fBase/base stands for base fairness, fRes/res for resource fairness and fChoke/choke for the choke point objective. First line is the Kendall tau correlation, second gives the correlation test p-value. Third row consists of the p-values of an equal proportions test of the 10 answer ratios for each map property (failing this test means that user feedback shall be disregarded).

Property	fBase/base	fRes/res	fChoke/choke
User answer to map value correlation	-0.1111	-0.4140	-0.3492
P-value correlation test	0.7275	0.1022	0.1715
P-value equal proportions test	0.2037	7.373e-06	6.358e-06

10 Discussion

As we had hypothesised, many of the objective functions partially conflict with each other. One of the reasons why we expected this is that the game design considerations that these objectives intend to model partially conflict even when humans design. The most fair map is a completely symmetric and relatively uninteresting-looking one; the resource distribution which best assigns resources unequivocally to individual players does not cluster the resources as well; the requirement that all bases have enough space to grow gives less room for the type of easily defended entrances to bases that could constitute choke points, etc.

We believe that similar design tradeoffs between properties desirable from a gameplay perspective exist for many, perhaps most, game genres and types of game content. For example, it is hard to design a ruleset for a game that is both easy to grasp and hard to master, or to design an NPC personality which is both psychologically believable and acts in a way which fits with the storyline of a game. Therefore, it is plausible that the multiobjective technique introduced here could be used in other game genres and for other types of game content – subject to the development of appropriate content representations and objective functions.

Our three-objective experiment should be seen as a first step rather than a complete coverage of the matter. To our knowledge, the use of more than 2 objectives in PCG has not been studied before, and the main question is if the larger effort in setting up and understanding the results is rewarded by interesting findings. Doubtlessly, inspecting two-objective results in order to detecting conflicts is much easier. Thus, one should only resolve to larger numbers of objectives when a first impression of the nature and interaction of the single functions has been obtained. Note that in multiobjective optimisation in general, the interpretation of results regarding the detection of conflicts is an active research topic and there are no commonly-agreed upon techniques. We hope that we have been able to add to the growing understanding of that topic, as well as to PCG research, with our study.

10.1 How Can We Use Multiobjective Evolution as a Design Support Tool?

To make our ideas about how MOEAs can aid designers more concrete, we present a short fictive scenario.

A designer is at work on producing an extensive library of maps for a new strategy game. The plan is to be able to balance the gameplay by having ready-made maps that empower weaker players by catering to their particular strengths, and indirectly handicapping stronger players through presenting them with the sorts of challenges they are least good at. Presently, the designer is tasked with finding maps that work well when the weaker player (player 1) is adept at tactical combat, but bad at harvesting resources and building up an effective base defence, and the stronger player (player 2) has as an only weakness a tendency to only build very large and advanced bases that require a great many resources. The designer therefore specifies that, although players 1 and 2 should have the same number of resources in their general sphere of interest, player 1 should have "her" resources much closer to her initial base and clustered together, whereas player 2's resources are spread out over a large area. The designer also specifies that the bases should be relatively close to each other (so that player 1 could conceivably attack before player 2 has finished building), and that there should be only a single path between the bases and that path should contain a choke point close to player 1's base (so she can defend easily). After specifying these requirements, the designer runs a number of multiobjective runs and look at the resulting combined Pareto front. The tradeoffs are studied, and the designer decides to what degree each of the objectives will have to be compromised. A small number of solutions, taken from different evolutionary runs in order to ensure diversity, are selected for further inspection and editing. In the last phase, human judgment and aesthetic sensibility is used to ensure that the maps are indeed playable and to improve them through manual editing. The process proposed here has similarities to what has elsewhere been called *mixed-initiative PCG* [38]

10.2 Making maps better

Our user study was designed to verify whether optimising for a particular objective yielded an increase in the map quality the fitness function was intended to model. The results are overall positive, both in that there were significant differences in terms of perceived qualities between maps optimised for different objective combinations, and that of the three main assessed qualities (base fairness, resource fairness, presence of choke points), at least the latter two were found strongly correlated with the corresponding objectives. The base fairness is problematic in that the human testers generally did not agree with this being a valuable objective for which different degrees of fulfilment would be reasonable (we do not fully agree with this especially when we think of balancing a game for two differently able players). We therefore consider the effectiveness of these fitness functions validated.

On the other hand, the maps we generated were not "good" maps, in the sense that an experienced player would enjoy one of our maps as much as a professionally designed map.

One interesting way of indirectly assessing the quality of generated maps would be to first create a predictor of map quality based on extracted features from existing maps. As StarCraft is a hugely popular game for which manual map editors have been available for a long time, several rich repositories of player-made maps exist. Some of these repositories feature rankings of popularity and/or perceived quality for their maps, based on the ratings of thousands of players. It would be eminently doable to calculate all objective scores for the functions defined above on each map. A model could then be trained (e.g. via neuroevolution) to reproduce the ranking observed on the repository, with only the objective vector as input. This would give us a predictor of map quality, which could be used to rate existing maps; it could also be used to create one or several new objective functions resulting from nonlinear combinations of these features. However, we have to be aware that the model would reflect the tastes of the players at that time; setting up maps with new, unseen features as 3-player/asymmetric maps would not be covered.

10.3 Symmetry in map design, and the purpose of PCG

The outcomes of our user study surprised us to some extent, in that so many of the commenters complained about the lack of symmetry in the maps our algorithms had generated. We had started out it with the assumption that symmetrical maps were boring because they were predictable, and considered the asymmetrical nature of our maps a feature rather than a bug (in fact, it would have been much easier to generate symmetrical maps, though in our opinion less interesting). However, most of the experienced StarCraft players that commented on our maps at Team Liquid disagreed with us. Many of them reasoned that as the maps were not symmetrical, they cannot be balanced. One explanation is that they have never seen an asymmetrical balanced map, and cannot imagine how such a map could be balanced, but would have liked the maps if they were shown to be balanced through extensive playthroughs. Another explanation is that such players value the symmetry and associated predictability *in itself*, in striking contrast to the authors. Certain players appreciate knowing their content very well, whether it be racing tracks or maps for FPS or RTS games, so they can perfect a strategy on a particular level; others value variation and novelty. When using procedural content generation to adapt gameplay such inter-player differences should be kept in mind; it is also important to test with the type of player population for which a particular PCG approach is targeted.

11 Conclusions

We have shown that complete and playable strategy game maps can be generated using multiobjective evolution. We have also introduced a generic indirect evolvable representation for such maps, together with its specialisation to two particular map spaces (StarCraft and a heightmap-based game). We have introduced more than a dozen evaluation functions related to gameplay experience, several of which partially conflict with each other; most of these functions could easily be generalised to other

strategy games. Finally, we have empirically demonstrated that some of our key evaluation functions correlate with perceived map qualities.

We believe that with relative little additional work, the method described here could be used as design a support tool for offline map generation in real games. For online content generation to be feasible, some work remains to be done, in particular the process needs to be sped up. As discussed above, a number of interesting research projects could be undertaken to further develop the technique introduced here.

12 Acknowledgments

This research was supported in part by the Danish Research Agency project *AGame-ComIn* (number 274-09-0083) and in part by the EU FP7 ICT project *SIREN* (number 258453). As stated in the introduction, this paper is based on two previously published papers [1,2]; the differences and additions with regard to those papers are detailed in the introduction.

References

1. J. Togelius, M. Preuss, and G. N.Yannakakis, "Towards multiobjective procedural map generation," in *Proceedings of the FDG Workshop on Procedural Content Generation*, 2010.
2. J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, and G. N. Yannakakis, "Multiobjective exploration of the starcraft map space," in *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2010.
3. T. Adams, "Re: Optimization-based versus "constructive" pcg (post to the "procedural content generation" google group)."
4. G. S. P. Miller, "The definition and rendering of terrain maps," in *Proceedings of SIGGRAPH*, vol. 20, 1986.
5. J. Olsen, "Realtime procedural terrain generation," University of Southern Denmark, Tech. Rep., 2004.
6. J. Doran and I. Parberry, "Controllable procedural terrain generation using software agents," *IEEE Transactions on Computational Intelligence and AI in Games*, 2010.
7. R. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra, "Integrating procedural generation and manual editing of virtual worlds," in *Proceedings of the FDG Workshop on Procedural Content Generation*, 2010.
8. J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: a taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. in print, 2011.
9. 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.
10. 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.
11. 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.
12. 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.
13. C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modeling player experience in super mario bros," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2009.
14. E. Hastings, R. Guha, and K. O. Stanley, "Evolving content in the galactic arms race video game," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2009.

15. J. Togelius and J. Schmidhuber, "An experiment in automatic game design," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2008.
16. C. Browne, "Automatic generation and evaluation of recombination games," Ph.D. dissertation, Queensland University of Technology, 2008.
17. I. Das and J. E. Dennis, "A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems," *Structural and Multidisciplinary Optimization*, vol. 14, pp. 63–69, 1997.
18. J. Koski, "Defectiveness of weighting method in multicriterion optimization of structures," *Communications in Applied Numerical Methods*, vol. 1, pp. 333–337, 1985.
19. A. Agapitos, J. Togelius, S. M. Lucas, J. Schmidhuber, and A. Konstantinides, "Generating diverse opponents with multiobjective evolution," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2008.
20. N. van Hoorn, J. Togelius, D. Wierstra, and J. Schmidhuber, "Robust player imitation with multiobjective evolution," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG)*, 2009.
21. J. Schrum and R. Miikkulainen, "Constructing complex npc behavior via multi-objective neuroevolution," in *Proceedings of the Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2008.
22. F. J. Gomez, J. Togelius, and J. Scmidhuber, "Measuring and optimizing behavioral complexity for evolutionary reinforcement learning," in *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, 2009.
23. J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation," in *Proceedings of the European Conference on Applications of Evolutionary Computation (EvoApplications)*, vol. 6024. Springer LNCS, 2010.
24. S. Papert, "Teaching children thinking," Massachusetts Institute of Technology AI Memos, Tech. Rep. 247, 1971.
25. T. W. Malone, "What makes computer games fun?" *Byte*, vol. 6, pp. 258–277, 1981.
26. M. Csikszentmihalyi, *Flow: The Psychology of Optimal Experience*. Harper Perennial, 1991.
27. R. Koster, *A Theory of Fun for Game Design*. Paraglyph press, 2005.
28. 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, 2002.
29. 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.
30. K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, pp. 115–148, 1995.
31. M. López-Ibáñez, L. Paquete, and T. Stützle, "EAF graphical tools," 2010. [Online]. Available: http://iridia.ulb.ac.be/~manuel/eaftools
32. V. G. d. Fonseca, C. M. Fonseca, and A. O. Hall, "Inferential performance assessment of stochastic optimisers and the attainment function," in *EMO '01: Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*. London, UK: Springer-Verlag, 2001, pp. 213–225.
33. M. Preuss, C. Kausch, C. Bouvy, and F. Henrich, "Decision space diversity can be essential for solving multiobjective real-world problems," in *MCDM for Sustainable Energy and Transportation Systems, EMO Track*, M. Ehrgott *et al.*, Eds. Berlin: Springer, 2008, pp. 367–377.
34. G. Rudolph, B. Naujoks, and M. Preuss, "Capabilities of emoa to detect and preserve equivalent pareto subsets," in *Evolutionary Multi-Criterion Optimization, 4th International Conference, EMO 2007, Proceedings*, ser. Lecture Notes in Computer Science, S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, and T. Murata, Eds., vol. 4403. Springer, 2007, pp. 36–50.
35. 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/.
36. G. N. Yannakakis, "How to Model and Augment Player Satisfaction: A Review," in *Proceedings of the 1st Workshop on Child, Computer and Interaction*. Chania, Crete: ACM Press, October 2008.
37. E. Wilson, "Probable inference, the law of succession, and statistical inference," *Journal of the American Statistical Association*, vol. 22, pp. 209–212, 1927.
38. G. Smith, J. Whitehead, and M. Mateas, "Tanagra: Reactive Planning and Constraint Solving for Mixed-Initiative Level Design," *Computational Intelligence and AI in Games, IEEE Transactions on*, no. 99, pp. 1–1, 2011.