Towards multiobjective procedural map generation

Julian Togelius
IT University of Copenhagen
Rued Langgaards Vej 7
Copenhagen, Denmark
julian@togelius.com

Mike Preuss
TU Dortmund
Otto-Hahn-Str. 14
Dortmund, Germany
mike.preuss@cs.uni-dortmund.de

Georgios N. Yannakakis
IT University of Copenhagen
Rued Langgaards Vej 7
Copenhagen, Denmark
yannakakis@itu.dk

ABSTRACT

A search-based procedural content generation (SBPCG) algorithm for strategy game maps is proposed. Two representations for strategy game maps are devised, along with a number of objectives relating to predicted player experience. A multiobjective evolutionary algorithm is used for searching the space of maps for candidates that satisfy pairs of these objectives. As the objectives are inherently partially conflicting, the algorithm generates Pareto fronts showing how these objectives can be balanced. Such fronts are argued to be a valuable tool for designers looking to balance various design needs. Choosing appropriate points (manually or automatically) on the Pareto fronts, maps can be found that exhibit good map design according to specified criteria, and could either be used directly in e.g. an RTS game or form the basis for further human design.

1. INTRODUCTION

This paper presents a search-based approach to generating maps, both terrain height and base and resource placement, for strategy games. Search-based procedural content generation has a number of important differences to other types of procedural content generation (PCG) in games; in particular it offers the possibility of directly optimizing content for particular types of players and experiences. As it is hard to devise a single and unambiguous measure of quality or fitness for strategy game maps, but reasonably easy to come up with measures of particular aspects of map quality, we cast the problem of generating such maps as involving multiple objectives and use a multiobjective evolutionary algorithm at the core of our approach.

Thus, the contributions of this paper are twofold: we show how complete maps (as opposed to just height maps) can be generated in a search-based paradigm, and show how multiobjective optimization can be applied to procedural content generation. In both cases, we believe that this is the first time such an approach is published in the academic literature. Before diving into the particulars of map representation and fitness function design, we will provide some background on procedural generation of maps and terrains, on search-based procedural content generation and on multiobjective evolution.

PCGames 2010, June 18, Monterey, CA, USA

1.1 Procedural map and terrain generation

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

Most strategy games come with a set of hand-crafted maps, used both in single-player "campaign" mode and multi-player modes. 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 in multi-player matches by 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, the new map could play to the strengths of her particular playing style while being seemingly dissimilar to previous maps she has played. In a multi-player game, maps might be generated that balance out the strengths of playing styles of different players with differing levels of proficiency, without resorting to explicit handicapping in terms of game rules or units supplied. Such content generation places particular demands on models of player behaviour and preferences, as well as on what ways 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 optional 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 new randomly generated world map. No details have to the authors' best knowledge been released about Civilization's map generation algorithm, but the very short time to generate a map suggests a relatively uncomplicated algorithm. The parameters available to the user for map generation are relatively few, the most important one relating to size and connectedness of the islands forming the world's landmass.

The simplest way of generating maps similar to those used by Civilization is probably to seed the ocean with embryonal islands, and have them grown out in a random direction a predefined number of steps [1]. Certain features on land, such as forest areas, can be created in the same way, and simple constraints can easily be added such as not connecting certain land areas (so as not to fill in canals).

Slightly more advanced approaches involve using fractals, such as the diamond-square algorithm [12]. The diamond-square algorithm works by iteratively subdividing areas of space and offsetting the midpoint by random amounts. Such algorithms are most commonly used for height maps, where they can, for example, generate believable mountains. An advantage of this family of algorithms is that they are so fast that they can often be used for realtime terrain generation [13].

Recently, Doran and Parberry suggested the use of software agents for generating terrain [8]. In their approach, a large number of agents are let loose on an initially featureless piece of terrain and collectively shape 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.

The *roguelike* genre of games (the original *Rogue* game as well as countless successors, such as *Nethack*, *Moria* and *Diablo*) is unique in being based completely on random map generation. What is generated here is not terrain but dungeon layouts, including walls and placements of monsters, traps and treasure. Each time you play one of these games you are presented with a new randomly generated map, and this is usually part of the premise of the game. These often work either similarly to the 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 from the main path until the room is filled), or by gluing together a number of prefabricated segments [1].

1.2 Search-based procedural content generation

All of 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 (or other types of generated content) that pass some sort of test. One example is Tarn Adams' ambitious game *Dwarf Fortress*, in which initial fractal map generation is usually repeated a couple of times, and the user is shown screenshots of "failed" maps along with explanations of which playability tests the maps failed to meet.

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 some sort of search or optimization algorithm is used to find better content based on the evaluations of already generated content.

Usually, some sort of evolutionary computation (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 each generation the best candidates are selected (according to some *fitness function*) and the worst candidates are replaced with new candidates generated through mutation (random perturbation) and/or recombination from the best candidates. Core concerns when devising an SBPCG solution to some content generation task is how to represent the content and how to devise the fitness function. A recent overview of SBPCG can be found in [19].

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 realtime PCG – but this depends on the fitness function and the search space, and choosing these carefully can allow content to be generated in a fraction of a second.

There have been a few previous attempts to use evolutionary algorithms to generate height maps for terrains before. Frade et al. used genetic programming to evolve terrains, with the evolved expression tree mapping coordinates on a grid to desired elevation at that point. The fitness function was based on "accessibility" meaning that all flat areas should be connected while no individual flat area grows too big. Only the height map was evolved, no other features of the map [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 intraversible. The fitness function is simply the length from start to finish, and the only constraint that the path should be connected [16]. Similarly, Ashlock et al. evolved path-planning problems in which the objective was to maximize distance from start to finish by placing walls at various positions and angles [3].

In the above examples, only parts of game environments (e.g. height maps and walls) are evolved – not complete, playable levels with e.g. items, monsters, resources. This is probably why the fitness functions are only tangentially related to actual game playability and entertainment; path

length and accessibility do not alone make for a well-designed level.

In contrast, some recent SBPCG papers have explicitly been based on notions of player entertainment. For example Togelius et al. evolved racing game tracks based on objectives inspired by Malone's entertainment dimensions [17]; Pedersen et al. evolved levels for Super Mario Bros based on a data-driven model of player affect [14]; Hastings et al. evolved weapons for a shooter game based player activity in the game [10]; and Browne evolved board game levels based on measures derived from studies of commercially successful board games [6]. However, none of these studies concerned maps or terrains.

1.3 Multiobjective evolution

In standard evolutionary computation, a single fitness function is used, meaning that candidate solutions are evaluated to a single numerical dimension. However, for many problems it is hard to devise a single objective measure; for example we want a car to be cheap, fast and safe, meaning that we need to optimize in three fitness dimensions. Furthermore, often these objectives are partially conflicting, meaning that in general, a faster car is less cheap.

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

Multiobjective evolutionary algorithms (MOEA) were invented to solve this problem, and are now a major research direction within evolutionary computation as well as common in industrial application. A MOEA presumes at least two fitness functions that are partially conflicting, and proceeds to search not for an optimal solution, but a *Pareto front* of nondominated solutions. A nondominated solution is any solution for which there is no other solution that is equal or better in all dimensions. In other words, a nondominated solution is either best according to some objective, or at least as good as other nondominated solutions in a unique combination of dimensions.

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

[1]More than three objectives are usually hard to handle for any MOEA, as the number of incomparable solutions—better in one objective, but worse in another—grows exponentially with the number of objectives and the Pareto front coverage gets very sparse. The Pareto front is always a set that consists of one dimension less than the number of objectives, e.g. a four-dimensional set for five objectives.

Optimizing some aspect of a game for playability is inherently a multiobjective problem, as it is very hard to formulate a single-dimensional automatic measure of how entertaining a game is; it is indeed not trivial to formulate partial measures of game enjoyability. When designing game content, it would seem invaluable for a designer to be able to conveniently visualize the tradeoffs inherent in a design problem; when automatically generating game content tailored to particular players, it would also seem ideal to first generate a selection of candidate content from which appropriate game content for the particular player could then be chosen, based on her previous playing style and experience model. Despite this seemingly perfect fit, we have not seen any examples of MOEAs used for PCG; the closest we can find are examples of multiobjective evolution of NPC behaviour [2].

1.4 Motivations for this paper

This paper intends to fill in the gaps highlighted by the above literature analysis in the following ways;

1. We generate complete maps (not just height maps or wall configurations) that could form complete levels for a simple strategy game; apart from terrain elevation, we also generate placement for bases and two types of resources.

2. We represent these maps in a way which is suitable for both strategy game engines and global optimizers such as evolutionary algorithms. Each map feature has its own real-valued parameter set, meaning that local changes in the genome has local effects in the generated maps.

3. We devise objectives that are directly motivated by optimizing the entertainment to be had from playing on these maps.

4. We use multiobjective evolutionary algorithms for exploring the tradeoffs between these partially conflicting objectives.

2. MAP REPRESENTATION

In these experiments we will evolve maps for an imaginary strategy game, containing some of the most common elements for RTS game maps. We take this to be locations for bases and for two types of resources (cf. minerals and vespene gas in *Starcraft*). Of course, the maps also include elevation differences.

We use two different representations of the map — an indirect representation used for searching (the *genotype*), and a direct representation for fitness testing and visualization (the *phenotype*). Each time fitnesses are calculated, a phenotype is created from each genotype.

The genotype (indirect) representation is a fixed-length array of real values between 0 and 1. The length of the array is decided by the number and types of map elements. Four types of elements are possible, with parameters as follows:

- **Base:** x and y coordinates of each base
- **Resource1:** x and y coordinates of each resource type 1.

- **Resource2:** x and y coordinates of each resource type 2.
- **Mountain:** 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.

For our experiments, we generated maps with three bases, four resources of each type and ten mountains, leading to genomes of length $3 * 2 + 4 * 2 + 4 * 2 + 10 * 5 = 72$.

This map representation has the advantage that it can be efficiently searched by many common global optimization algorithms, such as evolution strategies and particle swarm optimization. In particular, many of these algorithms assume a real-valued representation, and that local changes in the genotype have local effects in the phenotype. For example, when changing the x 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 size invariant, a map of any discrete size can be created out of it, as fits the preferences of the level designer.

The phenotype (direct) representation is similar to how the map would be represented in an actual game, and is designed to be easy to base fitness calculations on. This representation consists of a heightmap in the form of a $100 * 100$ grid where each cell can take on a discrete number between 0 and 99 representing elevation at that point, and three lists of x and y coordinates of bases, resource type 1 and resource type 2 respectively. The lists of resource sites are populated from the corresponding lists in the genotype representation by simply multiplying each x and y coordinate by 100.

The coordinate for a base can be generated using one of two different methods. The Cartesian method is the same as for resource placement – the two parameters are simply treated as x and y values and multiplied and discretized. Alternatively, in the Constrained Polar method, the coordinates are treated as angle and length of an axis extending from the center of the map, at the end of which the base is placed. Additionally, the representation is constrained so that each base is forced to be within its own arc of the circle, meaning that for three bases each base is placed within its own 120 degree arc; the length of the axis is constrained to between 1/2 and 1 of the radius of the map, meaning that bases cannot be place too close to the center of the map.

All cells of the heightmap are initially set to elevation zero. As already mentioned, 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 this paper). 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).

3. FITNESS FUNCTIONS

In SBPCG, there is a distinction among three types of fitness functions: *interactive*, *simulation-based* and *direct* [19]. Interactive fitness functions rely on human game players and provide direct or indirect feedback about the quality of the game content. While in a sense the ultimate type of fitness function, interactive fitness functions may require massive amounts of player input and might be only possible in game genres which provide sufficient game-player interaction, such as ongoing massively multiplayer games [10]. Simulation-based fitness functions assess content automatically through algorithmically playing the game or some aspect of the game using the candidate content. Such evaluations can potentially be accurate predictors of player enjoyment, but require both artificial intelligence capable of playing the game competently in a human-like manner and often substantial computation time [17, 18]. Direct fitness functions base their fitness calculations directly on the phenotype representation of the content. Such fitness functions are obviously much easier to implement and faster to compute than simulation-based functions, but it is hard to devise direct fitness functions that accurately predict key aspects of player experience (except when basing them on data-driven player models built from extensive user studies [14]).

In this paper, we do not have access to testing our maps on a complete RTS game, and we certainly do not have the luxury of having human players sit through countless hours to test the tens of thousands of candidate maps the evolutionary algorithm generates. However, we can simulate one key aspect of RTS gameplay: moving between two points along the fastest possible path. Our fitness functions are therefore a combination of direct fitness functions, measuring various properties of the height map and site distributions, with somewhat simulation-based measures of the *weighted distance* between sites. Weighted distance is calculated using the $A*$ pathfinding algorithm, where the transition between any two cells has a cost of 5 plus the difference in elevation between the two cells.

As discussed above, devising a single non-interactive fitness function that accurately captures the game entertainment notion of all players is a rather hard and complicated task; that complication drives our motivation for using multiobjective evolution. In this paper, we investigate the interplay of the following five fitness functions. Please note that any particular evolutionary run will only use two of these functions, for reasons discussed above. All functions are approximately normalized to the $[0, 1]$ range.

- 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 not be 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 strategically chosen cells and their counterparts on the opposite half of the grid in both x and y axes.
  Motivation: Aesthetic considerations and 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 rely on the same feature (base, mountain or resource) being 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. The number of meta-cells c considered in this study is 9.
  Motivation: Interestingness. Maps where resources are clustered together ($f_4 \approx 1$) motivates some players to explore more, and gives them more surprises.

In the experiments presented in this paper, only two objectives will be tested together in any single run of the algorithm. We will explore how these various objectives partially conflict and how these conflicts can be used for map design.

Note, for example, that a map where fitness function f_4 reached its maximum value would probably be rather boring to play. But this is the power of the multiobjective approach: as this objective is almost certain to conflict with objective f_3 (at least in maps where not all bases are clustered together) it will be easy to avoid such maps, and pick maps where a reasonable level of clustering can be balanced with fairness.

Despite the naive implementation of $A*$ we use, calculating all five objectives for a given map takes a small fraction of a second on a modern laptop computer.

4. MULTIOBJECTIVE EVOLUTIONARY ALGORITHM

The most popular multiobjective evolutionary algorithm is undoubtedly NSGA-II [7], which has shown to be competent in many benchmark and real-world applications. However, the last years have seen several newly developed algorithms, usually obtaining well received performance improvements. Of these, we employ the SMS-EMOA [4] (where SMS stands for *S-metric selection*) which is known as a quite greedy and fast descendant of the NSGA-II.

However, the general working scheme of most of these algorithms is relatively similar. A population of search points (called individuals with reference to the evolutionary roots) 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 about existing points (recombination) and performing undirected steps with a defined expected length (mutation). Selection chooses the best of the old and new individuals to survive and deletes the others. This working principle is common to many evolutionary algorithms as well as some other algorithms. It has its advantages in the minimal knowledge of the optimized problem necessary (black box, no algebraic form or gradients needed) and is on the other hand somewhat slower than classical optimization algorithms on convex/very simple problems.

The SMS-EMOA is greedy as it generates only one new individual per cycle and removes the individual with the smallest hypervolume contribution, that is the individual that dominates the least space in the Pareto plot (objective 1 over objective 2 for 2 objectives). It has been recently found that the greediness is well in place as it is highly unlikely that the SMS-EMOA is diverted from the real Pareto front [5], that is, the Pareto front is reachable for the algorithm in almost all cases.

To acknowledge 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 standard recombination/mutation operators SBX and PM, and set the run lengths after some testing to 30,000 evaluations. This is largely sufficient for the problems treated here as found by experiment. Improvements are getting very rare after doing so many cycles. In all experiments, we use populations of 50 individuals. According to our parameter tests, the population size has little effect on convergence speed, thus it only limits the number of individuals on the final Pareto front approximation.

Note that while the obtained points shall be near the real Pareto front, they do not necessarily have to be unique. Depending on the problem, any or many points on the front can have several pre-images, meaning that there are several ways to implement solutions with the same objective function values. Although this would be interesting to a game/level designer, we omit investigating this here due to space limitations. However, approaches to retrieve the different pre-images are available [15].

5. EXPERIMENTAL INVESTIGATION

The first experiments were concerned with finding which pairs of objectives exhibit partial conflicts. Therefore evolutionary runs were done with 12 pairs of the 5 objectives, and the resulting Pareto fronts exhibited. The standard cartesian mapping was used for base placement.

It was found that all of the runs that involve objective $f0$ (base distance) generated nontrivial Pareto fronts of 5-20 individuals, whereas all other runs (that did not involve f_0) generated Pareto fronts consisting of only one individual. A front resulting from a run involving f_0 and f_2 is depicted in figure 1.

We interpret this result as meaning that f_0 partially conflicts with all other objectives, whereas those other objectives probably do not conflict with each other (an alternative explanation would be that the evolutionary algorithm has not been successful in finding the region of search space

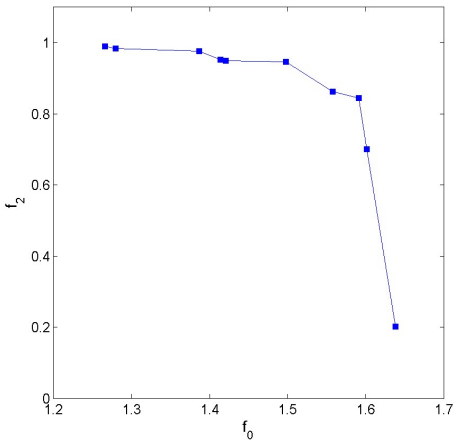

Figure 1: Pareto front for the objective pair f_0-f_2.

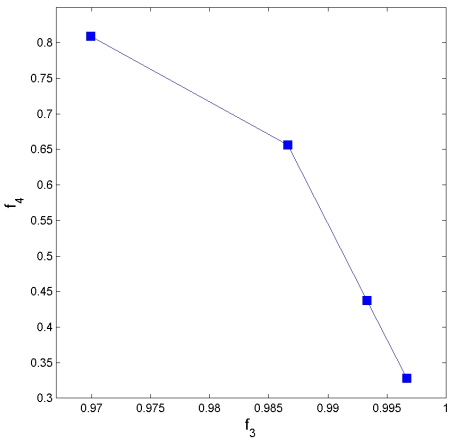

Figure 2: Pareto front for the objective pair f_3-f_4.

where conflicts occur).

This result surprised us, as we had assumed that f_1 (base on ground) and f_2 (asymmetry) would conflict, as would f_3 (resource distance) and f_4 (resource clustering). Upon closer inspection of the maps involving these two pairs of objectives, however, we found that in both cases the algorithm had avoided the conflict through placing all three bases very close to each other. This way, resource clustering could be maximized while keeping the same distance from all bases to nearest resource, and a very asymmetric map could be generated while keeping all the bases on the ground.

However, having all the bases very close to each other is clearly not good level design, as it would make for very short games with little use of most parts of the map. One alternative would be to optimize for three objectives simultaneously, and include f_0 along any other pair of objectives. As increasing the number of objectives generally greatly increases the computation time required for exploring regions that may not be of interest, we chose to go for another option: we changed genotype to phenotype mapping for base placement from the previous cartesian to constrained polar form. This representation prevents bases from being placed too close to each other. Additionally, a mechanism was implemented in the MOEA which removed any candidates with f_0 lower than 2.

Under these new conditions, partial conflicts (as indicated by substantial Pareto fronts) were found between objective pairs (f_1 and f_2), (f_1 and f_3), (f_2 and f_3) and (f_3 and f_4). The resulting Pareto front for objective pair f_3 and f_4 is shown in Figure 2. Four maps taken from that front are shown in Figure 3.

It is apparent from these pictures that the algorithm finds playable, varied and seemingly well-designed maps, which are interestingly different from one end of the Pareto front to the next.

6. CONCLUSIONS

In this paper, we have demonstrated a search-based approach to procedural generation of strategy games, using a multiobjective evolutionary algorithm. We defined five suitable objectives for the algorithm motivated by player experience considerations. The results of our experiments show that there exist a number of partial conflicts between these objectives, which makes the algorithm useful as a design support tool: designers can visualize the necessary tradeoffs as a Pareto front and choose maps and make an informed aesthetic judgement. But the fronts can also form the basis for completely automatic generation of content. Importantly, the generated maps (when selected from the middle of Pareto fronts) look like plausible and playable RTS maps.

However, the research here is meant as a first step towards multiobjective search-based PCG capable of realiably generating quality maps for real strategy games. A number of issues remain to explore.

6.1 Future work

While the phenotype representation used in this paper is intentionally made similar to typical representations used in strategy games, it is probably not identical to the representation in any particular existing game. An important next step for us is to adapt the representation (possibly including the types of elements on the maps, and the fitness function) to fit some existing commercial-quality RTS game, and interface our algorithm to that game.

Another important next step is to investigate the use of more than two objectives at once. Can three-dimensional Pareto fronts be efficiently generated in this context, and can they be effectively used for design support, or for automated content selection?

Alternative fitness functions and more objectives need to be considered in future studies. For instance, one could investigate the impact of Malone's [11] factors for the design of engaging gameplay (challenge, curiosity and fantasy) on map generation. The three factors can already be quantified via the fitness functions available (e.g. curiosity could be represented by the resource unpredictability function f_4). However, new quantitative measures of those qualitative factors will be required, encapsulating Malone's principles better, and feed the MOEA algorithm with more meaningful objectives.

After these investigations are completed our intention is to cross-validate our conclusions regarding map generation against actual players. It would be investigated whether maps designed to lead to short, aggressive games actually did

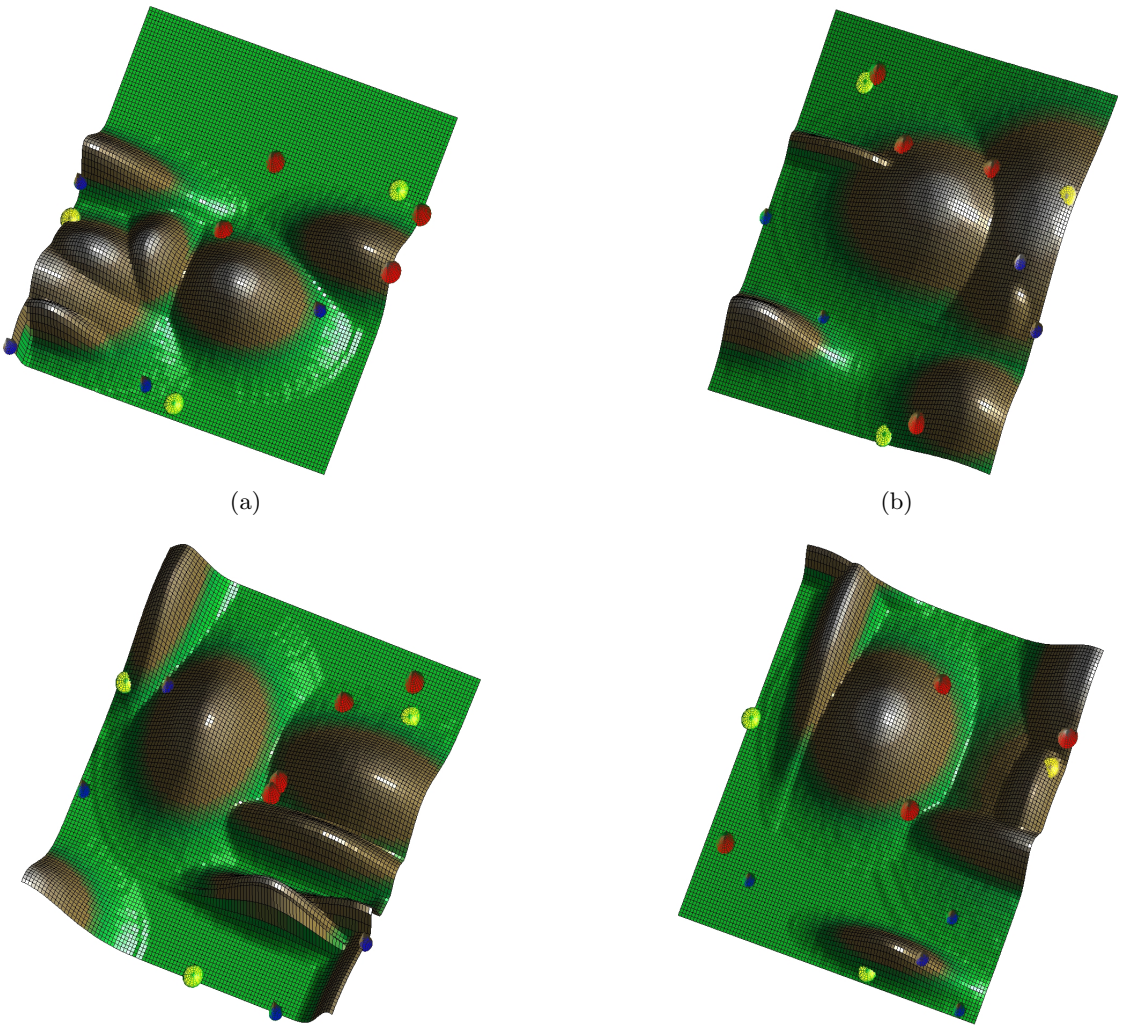

Figure 3: The four generated maps taken from the Pareto front of the objective pair f_3-f_4. Bases are illustrated as yellow spheres; resources are depicted as either red (type 1) or blue (type 2) cones.

that, and whether maps generated to balance the differing abilities of different players actually did that.

Eventually, a complete game could be based around this type of procedural map generation, where new maps are continually generated based on the previous performance and expressed preferences of the players.

7. ACKNOWLEDGMENTS

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

8. REFERENCES

[1] T. Adams. Re: Optimization-based versus "constructive" pcg (post to the "procedural content generation" google group).

[2] 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.

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

[4] N. Beume, B. Naujoks, and M. Emmerich. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653–1669, 2007.

[5] N. Beume, B. Naujoks, M. Preuss, G. Rudolph, and T. Wagner. Effects of 1-greedy-metric-selection on innumerably large pareto fronts. In M. E. et al., editor, *Evolutionary Multi-Criterion Optimization, 5th International Conference, EMO 2009, Nantes, France, April 7-10, 2009. Proceedings*, volume 5467 of *Lecture Notes in Computer Science*, pages 21–35. Springer, 2009.

[6] C. Browne. *Automatic generation and evaluation of*

recombination games. PhD thesis, Queensland University of Technology, 2008.

[7] K. Deb, A. Pratap, and S. Agarwal. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Trans. on Evolutionary Computation*, 6(8), 2002.

[8] J. Doran and I. Parberry. Controllable procedural terrain generation using software agents. *IEEE Transactions on Computational Intelligence and AI in Games*, 2010.

[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)*, volume 6024, pages 90–99. Springer LNCS, 2010.

[10] 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.

[11] T. W. Malone. What makes computer games fun? *Byte*, 6:258–277, 1981.

[12] G. S. P. Miller. The definition and rendering of terrain maps. In *Proceedings of SIGGRAPH*, volume 20, 1986.

[13] J. Olsen. Realtime procedural terrain generation. 2004.

[14] 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.

[15] G. Rudolph and M. Preuss. A multiobjective approach for finding equivalent inverse images of pareto-optimal objective vectors. In C. Coello Coello, P. Bonissone, and Y. Jin, editors, *2009 IEEE Symposium on Computational Intelligence in Multicriteria Decision-Making (IEEE MCDM 2009)*, pages 74–79, Piscataway (NJ), 2009. IEEE Press.

[16] 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)*, volume 6024, pages 130–139. Springer LNCS, 2010.

[17] 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.

[18] J. Togelius and J. Schmidhuber. An experiment in automatic game design. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2008.

[19] 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)*, volume 6024. Springer LNCS, 2010.