Natural Computing manuscript No.
(will be inserted by the editor)

On Balance and Dynamism in Procedural Content Generation with Self-Adaptive Evolutionary Algorithms

Raúl Lara-Cabrera · Carlos Cotta · Antonio J. Fernández-Leiva

Received: date / Accepted: date

Abstract We consider search-based procedural content generation in the context of *Planet Wars*, an RTS game. The objective of this work is to generate maps for the aforementioned game, that result in an interesting game-play. In order to characterize interestingness we focus on the properties of balance and dynamism. The former captures the fact that no player is overwhelmed by the opponent during the game, whereas the latter tries to model the fact that there is a lot of action during the game. To measure these properties on a given map, we conduct several games on them using top AI bots and collect statistics which are, in turn, used as inputs of a fuzzy rule base. This system is embedded within an evolutionary algorithm that features self-adaptation of mutation parameters as well as variable-length chromosomes (thus implying maps of different sizes). The experimentation focuses both on the optimization of balance and dynamism as stand-alone properties and in the analysis of the different tradeoffs attainable through them. To reach this goal a multi objective approach is used. We analyze both the usefulness of map-size self-adaptation in each scenario, as well as the properties of maps leading to different tradeoffs between dynamism and balance.

Keywords Procedural content generation · Game attractiveness · RTS · Self-adaptive evolutionary algorithm · Multiobjective optimization

1 Introduction

Videogames are becoming the most profitable component of the entertainment industry with a total consumer spend of 24.75 billion US dollars in 2011 [4]. The process of creating the game content, such as maps and models, is one of the main budget items due to the amount of work required and the number of professionals (graphic and game designers, software developers, etc.) involved, so any saving in the cost of manually creating this content is desirable.

R. Lara-Cabrera · C. Cotta · A. J. Fernández-Leiva
Department of "Lenguajes y Ciencias de la Computación"
ETSI Informática, University of Málaga, Campus de Teatinos, 29071 Málaga – Spain
E-mail: {raul,ccottap,afdez}@lcc.uma.es

Procedural content generation (PCG) comprises useful techniques to create game content automatically through algorithmic means which is useful for the videogame industry for several reasons, such as a reduced memory consumption of the games, the possibility to create endless videogames that change every time a new game is started and the reduced expense of creating the aforementioned game content, just to name a few.

The industry already recognizes these benefits so there are several examples of the use of PCG techniques in the development of a commercial videogame and even during its game-play. The first-person shooter (FPS) *Borderlands* [10] uses a PCG system that creates weapons and items with different features and special bonuses –such as rate of fire, accuracy and ammo regeneration– and even the creation of random enemies. *Minecraft* [21] is a sandbox building game that uses PCG to expand dynamically infinite maps where the game takes place. The music of *Spore* [20] (a god game simulation) is procedurally generated, and has a dynamically generated universe.

From an academic point of view, PCG is an active field of the computational intelligence in the field of games, as shown by the large number of papers on this topic. For example, Togelius et al. [25] designed a PCG system capable of generating tracks for a simple racing game from a parameter vector using a deterministic genotype-to-phenotype mapping. Frade et al. introduced the use of genetic programming to evolve maps for videogames (namely terrain programming), using either subjective human-based feedback [7,8] or automated quality measures such as accessibility [6] or edge-length [9]. Togelius et al. [27,26] linked several structural properties of maps, in an imaginary RTS, to game properties such as fairness, aesthetics, playability or interestingness, and used a multi-objective evolutionary algorithm to analyze the conflict between pairs of them. Mahlmann et al. [18] described another search-based map generator for an abstract version of the game *Dune*. Low resolution matrices represented map genotypes that were then converted to higher resolution maps by means of cellular automata.

Real-time strategy (RTS) games are a genre of video-games that usually consist of resource gathering, base building, in-game technological development and indirect control of units. They are games for two or more players who have to deal with incomplete information during the game like, for example, the map being covered by "fog of war". Precisely because of these features, RTS games are a great tool for computational intelligence research. Although academic game artificial intelligence (AI) has usually been linked to non player character (NPC) behavior and path-finding, there are new research areas that have recently dealt with other game development challenges such as procedural content generation (PCG), player experience modeling (PEM) and large scale game data mining [30].

Based on the taxonomy proposed by Togelius et al. [28], this paper presents an offline PCG method that generates necessary content using random seeds, deterministic generation and which follows a generate-and-test schema. This method generates maps for the RTS game *Planet Wars* that exhibit interesting properties such as balance (in terms of players not having any advantage over their opponents) and dynamism (promoting active games in which fights and fleet movements take place) using evolutionary algorithms (EAs). It also compares different strategies when generating maps (single and multiple objective optimization) as well as different parameters of the evolutionary algorithm.

2 Background

This section describes the game for which the maps are generated, specifically the game *Planet Wars* (Sect. 2.1). Next, there is a brief description of what defines unsuccessful videogames and what makes games more appealing to players (Sect. 2.2).

2.1 Planet Wars

Planet Wars is a real-time strategy game based on *Galcon* and used in the *Google AI Challenge 2010*. It takes place on a map, on which a number of planets of different sizes are scattered. The objective is to take over all the planets on the map or eliminate all of your opponents' ships. Initially each player has a home planet, and the rest of planets are neutral. The sizes of the planets are related to how many ships are produced every turn on each planet, so bigger planets are more appealing to the players (neutral planets do not produce any ships though). In order to conquer a planet a player has to move ships to it so as to outnumber the number of ships already placed on the planet under dispute (these ships may belong to another player, or represent an initial neutral number of ships to be taken over if the planet has not been conquered by either player up to that point). The game finishes when a certain number of turns is reached (so the player with the highest number of ships is the winner) or when all the players except the winner (the player that still has ships) have lost all their ships.

During a turn, the player can send fleets of ships from a planet that they own to another planet. If the player owns the target planet the number of the fleet's ships is added to the number of ships on that planet, otherwise a battle takes place on the target planet: ships from both sides destroy each other so the player with the greatest number of ships remaining, owns the planet (with the number of ships determined by the difference with the initial number of ships). The fleets take a number of turns to reach their destination planets depending on how far the planet is, and they can not be redirected while in flight. At the end of each turn, owned planets produce a number of new ships proportional to their size. Although the players issue their orders by a turn-by-turn scheme, they give these orders at the same time, so we can treat this game as a real-time one.

2.2 Game Attractiveness and Player Satisfaction

In the past, the success of a video game was directly associated with its graphical quality, but in the last decade this has changed and having good graphics does not necessarily ensure high sales of a video game. Existing players demand video games that show more than just a nice graphical skeleton and other issues (such as music, the story, or the atmosphere of the game for instance) influence the decision of a player to own a specific game. The question of what it is that attracts the attention of players in a game is easy to answer: fun. However, how to obtain fun games and whether we can predict if the game will be of interest to players are not so easily answered.

We find a number of theories in the literature, on why we play games and what makes video games fun [13] and, according to [1], the success of a game might be deduced by measuring in advance the quality of the game (which seems however to be a difficult task). The notion of fun is difficult to measure as this depends on each player but it is naturally associated with the notion of player satisfaction: the greater the satisfaction, the greater the fun. Two research trends for measuring the level of player's satisfaction were categorized in [29]: the qualitative approach [2, 19] and the quantitative model [22]. Whilst the first approach is closer to psychology, the second one suggests that the game is adapted in response to player's needs. Certainly, the latter can be handled via the generation of game contents that adjust to the player's skills, and this is currently automated via computational intelligence (CI) techniques.

The most classic application of CI in this sense is the automatic generation of strategies to govern the non-player characters (NPCs); in fact this is a natural consequence of trying to satisfy the player's demand to have opponents exhibiting intelligent behavior and thus researchers try to obtain more intelligent NPCs. However, this is not the primary goal of commercial games and game developers focus on the attainment of NPCs that offer a challenge that is fun [16]. A key principle is to let players survive a long time (e.g., reaching the highest possible number of levels) while preventing players suffering a severe defeat (NPCs that are too intelligent are therefore not wanted) and, at the same time, making sure that too easy victory cannot be obtained (so, NPCs that are too stupid are not appropriate either). The correct tradeoff between intelligence and stupidity is not easy to achieve though and one can find in the literature many proposals for generating NPCs the behaviors of which self-adapt to player skills [24, 11].

However, CI has also been applied to many other aspects of game development such as computational narratives, player modeling, learning in games, intelligent camera control, and procedure content generation (PCG), [17]. This paper centers on the capacity of PCG to engage the player (as commercial games demand) by maintaining, during a match, an adequate tradeoff between the dynamism of the game and the balance between players that in all probability have distinct skills. The work described here is related with self-adjustment techniques and can be considered another form of respecting the key principles of game development as described above.

3 Evolutionary Map Design for Planet Wars

As stated in the previous sections, we aim to design maps for the RTS game under consideration, focusing on the properties that a priori make it entertaining and appealing to play, in other words, ensuring that the games are balanced (i.e., no player thoroughly dominates the other) and dynamic (i.e., there is action during the game, and a player who is at a disadvantage at a certain point can regain their position and start to dominate the game at a later stage). In fact we discovered that these two properties are partially conflicting, thus suggesting the need for a multiobjective approach. This will be dealt with later on. Firstly, let us consider how solutions are represented and evaluated, as well as the evolutionary engine used to design maps.

3.1 Representation and Evaluation

A map for *Planet Wars* can be defined on the basis of a number n_p of planets located on a 2D plane. Besides its coordinates (x_i, y_i), each planet is characterized by two additional properties: its size s_i and a number of ships w_i. The first parameter (s_i) captures the rate at which this planet will produce new ships once it is captured by one of the players. As for the second parameter (w_i), it indicates the initial number of ships required to conquer that planet. Hence, we can denote a map as a list $[\boldsymbol{\rho}_1, \boldsymbol{\rho}_2, \cdots, \boldsymbol{\rho}_{n_p}]$, where each $\boldsymbol{\rho}_i$ is a tuple $\langle x_i, y_i, s_i, w_i \rangle$. In order to be playable, a map also needs to specify the home planets of the players. In order to keep things simple, we have taken the first two planets $\boldsymbol{\rho}_1$ and $\boldsymbol{\rho}_2$ as the home planets. Note that the number of planets n_p need not be fixed, and can range between 15 and 30 (as per specifications of the *Google AI Challenge 2010*). In fact, one of the features of the evolutionary approach discussed later on is the ability to self-adapt to not only search parameters but also to the complexity (i.e., number of planets) of the map.

In order to evaluate the playability features of a map, we use a tournament system. We run a set of *Planet Wars* games between an arbitrary number of non-player characters (NPC). Each NPC plays a game against another. The tournament system evaluates every game, analyzing some statistics gathered from each of them. Such statistics capture properties of a given game and are later used to quantify the extent to which a game can be said to have been balanced or dynamic. More precisely, the system collects the following information from the i-th game (out of the total number of N_g games played in the tournament):

- Territorial imbalance: Let τ_i be the number of turns played in the current game, and let $\pi_{ij}^{(a)}$ be the percentage of planets (out of the total number of planets n_p) owned by player a (where $a \in \{1, 2\}$), in the j-th turn of this i-th game. Then, we define

$$\Pi_i = \frac{1}{\tau_i} \sum_{j=1}^{\tau_i} \left| \pi_{ij}^{(1)} - \pi_{ij}^{(2)} \right| \tag{1}$$

  i.e., the average imbalance in conquered planets throughout the game. This variable can take values from 0 to 1.
- Growth imbalance: planets have different sizes and hence produce new ships at different rates. We can thus define the imbalance in the capacity for producing new ships analogously to Eq. (1), i.e.:

$$\Gamma_i = \frac{1}{\tau_i} \sum_{j=1}^{\tau_i} \left| \gamma_{ij}^{(1)} - \gamma_{ij}^{(2)} \right| , \tag{2}$$

  where $\gamma_{ij}^{(a)}$ is the percentage of the total growth capacity of the map accumulated by player a in the j-th turn of the i-th game. While obviously correlated to territorial imbalance, this variable is not identical to the latter and provides a different perspective on whether forces are balanced or not. Again, it is easy to see that this variable ranges from 0 to 1.

– Ship imbalance: Much as before, we can measure the extent to which the actual fleets owned by the players are balanced or not. This is done by computing:

$$\Xi_i = \frac{1}{\tau_i} \sum_{j=1}^{\tau_i} \left| \xi_{ij}^{(1)} - \xi_{ij}^{(2)} \right| , \tag{3}$$

where $\xi_{ij}^{(a)}$ is the percentage of the total number of ships the j-th turn of the i-th game that are owned by player a. Note that a player can achieve territorial balance at the cost of getting involved in numerous battles and seeing their number of ships reduced, so once again this variable is correlated but not qualitatively identical to the previous variables. It ranges from 0 to 1 as Pi_i and Γ_i.

– Game length: this is just the percentage of the maximum number of turns allowed $\tau_{\max}$ that have been played in the current game:

$$T_i = \tau_i / \tau_{\max} \tag{4}$$

Ranging from 0 to 1, this variable simply measures the duration of the game.

– Conquering rate: this variable is used to account for the percentage of planets which are not neutral (i.e., they have been conquered by either player) throughout the game. It can be easily computed using values $\pi_{i\tau_{\max}}^{(a)}$ as

$$K_i = \pi_{i\tau_{\max}}^{(1)} + \pi_{i\tau_{\max}}^{(2)} \tag{5}$$

and is obviously bounded between 0 and 1.

– Reconquering rate: this variable is a measure of how often planets change ownership. Let ζ_{ij} be the number of planets that were owned by a player in turn $j-1$ and conquered by the other player in turn j (regardless of who was the original owner, i.e., we simple count how many planets have changed hands in each turn). Then:

$$Z_i = \frac{1}{\tau_i} \sum_{j=1}^{\tau_i} \frac{\zeta_{ij}}{n_p} \tag{6}$$

While this variable can theoretically range from 0 to 1, in practice it is more likely to take values closer to the lower end.

– Peak difference: this is actually a family of variables, each of them measuring the maximal amplitude of the variation in any of the resources accounted for, in this case planets, combined size and ships. More precisely, if $\phi_{ij}^{(a)}$ is the number of resources ϕ (which can be π, γ or ξ) owned by player a in the j-th turn of the i-th game, we record the two points in which the relative difference is best for one player compared to the other one and add up both quantities, i.e.,

$$\Delta_i^{\phi} = \max_{1 \leqslant j \leqslant \tau_i} \left\{ \frac{\phi_{ij}^{(1)} - \phi_{ij}^{(2)}}{\phi_{ij}^{(1)} + \phi_{ij}^{(2)}} \right\} - \min_{1 \leqslant j \leqslant \tau_i} \left\{ \frac{\phi_{ij}^{(1)} - \phi_{ij}^{(2)}}{\phi_{ij}^{(1)} + \phi_{ij}^{(2)}} \right\} \tag{7}$$

In this case, the three variables $\Delta_i^{\pi}, \Delta_i^{\gamma}$ and Δ_i^{ξ} range from 0 to 2.

Balance:
1. **if** Π **is** LO **and** Γ **is** LO **then** *bal* **is** HI
2. **if** Π **is** HI **and** Γ **is** LO **and** Ξ **is** LO **then** *bal* **is** MED
3. **if** (Π **is** LO **and** Γ **is** HI) **or** T **is** LO **then** *bal* **is** LO

Dynamism:
1. **if** K **is** HI **and** Z **is** HI **then** *dyn* **is** HI
2. **if** Δ^{π} **is** HI **and** Δ^{γ} **is** HI **and** Δ^{ξ} **is** HI **then** *dyn* **is** HI
3. **if** Δ^{π} **is** HI **and** (Δ^{γ} **is** LO **or** Δ^{ξ} **is** LO) **then** *dyn* **is** MED
4. **if** Δ^{γ} **is** HI **and** (Δ^{π} **is** LO **or** Δ^{ξ} **is** LO) **then** *dyn* **is** MED
5. **if** Δ^{ξ} **is** HI **and** (Δ^{γ} **is** LO **or** Δ^{π} **is** LO) **then** *dyn* **is** MED
6. **if** Δ^{π} **is** LO **and** Δ^{γ} **is** LO **and** Δ^{ξ} **is** LO **then** *dyn* **is** LO
7. **if** K **is** LO **or** Z **is** LO **or** T **is very** LO **then** *dyn* **is** LO

Fig. 1 Fuzzy rule base for balance and dynamism.

Each of the aforementioned variables is subsequently averaged for the N_g games played in each tournament in order to obtain aggregate values (for the sake of simplicity we use the same notation as above, dropping the subindex i to denote these averaged values, i.e., Π, Γ, etc.) that will be then used for evaluating balance and dynamism. For this purpose, we have considered the use of a fuzzy rule base in order to encapsulate the expert's knowledge of what makes a certain game balanced or dynamic. The motivation for this is manifold. In previous work where the focus was only on balance [14] a mathematical formula was used to aggregate some statistics into a single value. While this is common practice, it is also true that there is a certain degree of arbitrariness on how the values are combined and how they are weighted in order to produce a single number. Certainly, it can be argued that the specific functional form used is guided by experts' knowledge. Then again, it is preferable to have this knowledge expressed at a higher-level, making it more amenable to interpretation and tuning if necessary.

The fuzzy rules considered are shown in Fig. 1 and the underlying fuzzy sets are depicted in Fig. 2. We first focus on balance. The first rule (1) defines that a map will have a high balance if the difference between the number of planets of both players during the game (namely Π) is low and so is the difference between their cumulated size (i.e., Γ). If the latter is low (and thus neither of the players has had a big advantage in growth rate); balance is said to be medium in rule (2) if there is an imbalance in the number of planets and ships (and hence one of the players has had a material advantage). Finally, rule (3) states that there is a low balance if both players have controlled a similar number of planets but with disparate sizes, or if the game length has been short (and hence one of the players could win easily). Note that we do not have to exhaustively cover all possible combinations of input variables. For example, if there is an imbalance both in the number of planets and growth rate, we cannot make a statement about whether the game is globally balanced or imbalanced because this may depend on whether both imbalances fall on the same side or not. However, this does not cause a problem because the aforementioned rules can still be activated to some degree in this situation (there are two input fuzzy sets overlapping across the whole input domain), and hence defuzzification can be carried out (details on the methods of instantiating fuzzy inference and defuzzification are provided in Sect. 4) .

Regarding dynamism, the rationale behind the rules presented is to attribute a high dynamism to games in which many planets are conquered and change hands often – rule (1). Likewise, if there are high peak differences in all resources

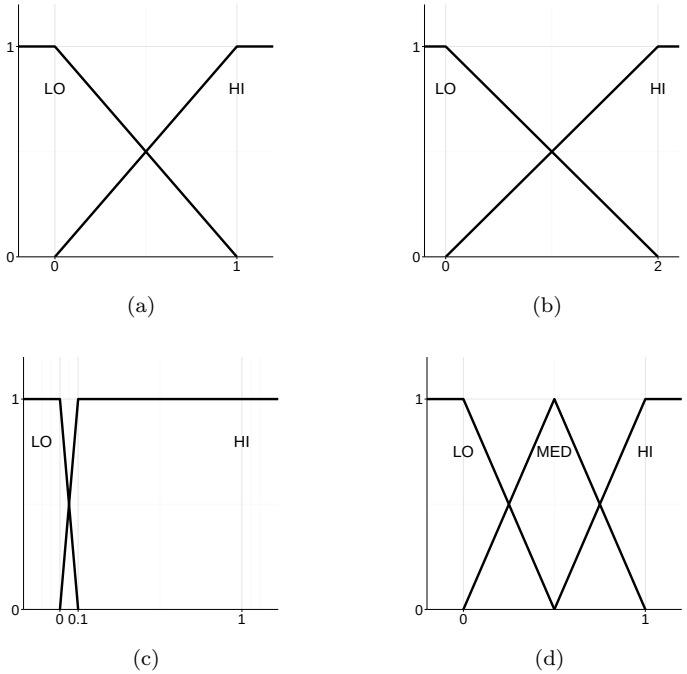


Fig. 2 Membership functions of the fuzzy sets. They correspond to variables (a) Π, Γ, Ξ, T, K (b) $\Delta^{\pi}, \Delta^{\gamma}, \Delta^{\xi}$ (c) Z and (d) *bal* and *dyn*.

accounted for, the game is also dynamic since one the players has made a comeback from a difficult position – rule (2). If the peak difference is high in one of the resources but not high in at least one of the remaining ones, then the game is considered dynamic at a medium level – rules (3)-(5). Along these lines, if all peak differences are low, dynamism is also considered to be low. However this rule (6) would not cover a scenario in which both sides frantically fight each other but always remain with balanced forces. That being said, this situation is somewhat extreme and is nevertheless mitigated by the fact that rule (1) would then enter into action, raising the dynamism. Finally, rule (7) states that if few planets are conquered or these rarely change hands or the game is very short, then the game is not dynamic.

Clearly, the rules defined above are not the only way in which balance and dynamism can be characterized. An expert in the game could think of alternative rules, or even of alternative statistics to be collected from the games. Be that as it may, these rules represent a reasonable approach for the properties we are interested in. Furthermore, they provide a modular way to evaluate tentative maps, thus easing the incorporation of new rules or the modification of existing ones.

3.2 The Evolutionary Approach

The procedural map generator makes use of a self-adaptive evolutionary approach which tries to optimize either balance or dynamism. Solutions are encoded as mixed real-integer vectors as planet coordinates (x_i, y_i) are real-valued numbers but sizes s_i and initial number of ships w_i are positive integers. Thus, a hybrid mutation operator is considered, using different mutation methods for parameters of either type: for real-valued parameters, Gaussian mutation is used; as for integer variables, it considers a method that generates suitable integer mutations [15,23] – see also [14]. It is similar to the mutation of real values but it uses the difference of two geometrically distributed random variables to generate the perturbation instead of the normal distributed random variables used for real values. The parameters governing mutation in either case are also a part of the solutions, thus providing the means for self-adapting them. More precisely, in the case of real-valued parameters $\langle r_1, ..., r_n \rangle$ they are extended with n step sizes, one for each parameter, resulting in $\langle r_1, ..., r_n, \sigma_1, ..., \sigma_n \rangle$. The mutation mechanism is specified as follows:

$$\sigma_i' = \sigma_i \cdot e^{\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)}$$
$$r_i' = r_i + \sigma_i \cdot N_i(0,1)$$

where $\tau' \propto 1/\sqrt{2n}$, and $\tau \propto 1/\sqrt{2\sqrt{n}}$. A boundary rule is applied to step-sizes to prevent standard deviations very close to zero: $\sigma_i' < \epsilon_0 \Rightarrow \sigma_i' = \epsilon_0$ (in this algorithm, σ_0 represents 1% of the parameter's range). Regarding integer-valued parameters $\langle z_1, ..., z_m \rangle$ they are extended in a similar way as are real-valued parameters, resulting in $\langle z_1, ..., z_m, \varsigma_1, ..., \varsigma_m \rangle$. The mutation mechanism is specified as follows:

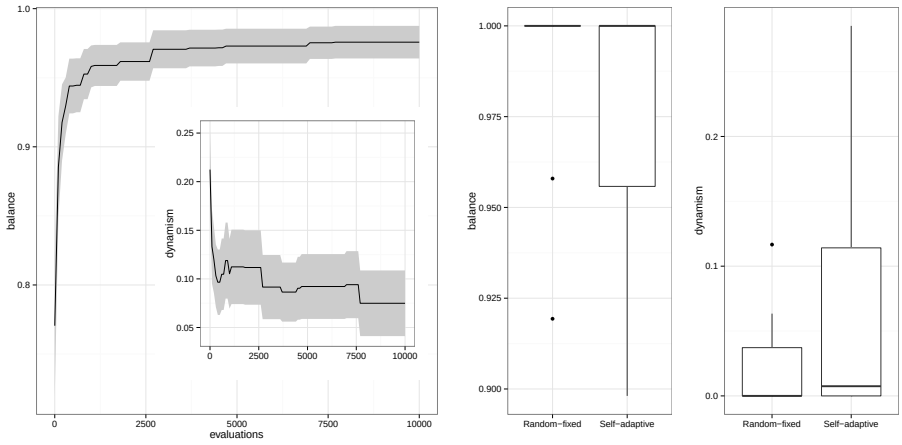
$$\varsigma_i' = \max(1, \varsigma_i \cdot e^{\tau \cdot N(0,1) + \tau' \cdot N(0,1)})$$
$$\psi_i = 1 - (\varsigma_i'/m) \left(1 + \sqrt{1 + \left(\frac{\varsigma_i'}{m} \right)^2} \right)^{-1}$$
$$z_i' = z_i + \left\lfloor \frac{ln(1 - U(0,1))}{ln(1 - \psi_i)} \right\rfloor - \left\lfloor \frac{ln(1 - U(0,1))}{ln(1 - \psi_i)} \right\rfloor$$

where $\tau = 1/\sqrt{2m}$ and $\tau' = 1/\sqrt{2\sqrt{m}}$. As described, the main difference between the two methods is the distribution used to generate the perturbation.

Regarding the recombination operator, we consider a "cut and splice" operator that recombines two individuals by swapping cut pieces with different sizes. The operator selects one cut point for each individual and then swaps these pieces, obtaining two new individuals with a different number of planets in relation to their parents. This endows the algorithm with further self-adaptation capabilities, thereby affecting the complexity of the maps, i.e., the number of planets in the solutions.

4 Results

The self-adaptive EA described in Sect. 3.2 has been implemented on the DEAP library [5], using a population size of 100 individuals and a $(\mu + \lambda)$ generational



(a) (b)

Fig. 3 Optimization of balance. (a) Mean balance attained in the self-adaptive EA. The inset shows the evolution of the dynamism of the corresponding high-balance solutions. In both cases the shaded area indicates the standard error of the mean. (b) Boxplot of the final values of balance and dynamism.

scheme, with $\mu = 10$, $\lambda = 100$. In order to analyze the effectiveness of the self-adaptation of the number of planets, we have also considered a version of the algorithm in which the length of solutions is decided at random at the beginning of each run and kept fixed for all solutions during that run. The cut-and-splice recombination is in this case substituted by a two-point crossover. On each given optimization scenario we have conducted a series of 10 runs both algorithms.

The players of the tournament system used to assess the quality of the maps during the evaluation phase were three bots submitted to the *Google AI Challenge 2010*, namely *Manwe*[1], *Flagscapper's bot*[2] and *fglider's bot*[3]. All of them ranked in the top 100 (there were over 4600 participants) and who have their source code available. The maximum number of turns per game $\tau_{\max}$ is 400 turns. As for the evaluation of fuzzy rules, we have used the min t-norm, the max t-conorm, the x^2 function as a realization of the fuzzy modifier *very*, and the center of mass as the defuzzification method.

4.1 Balance vs. Dynamism

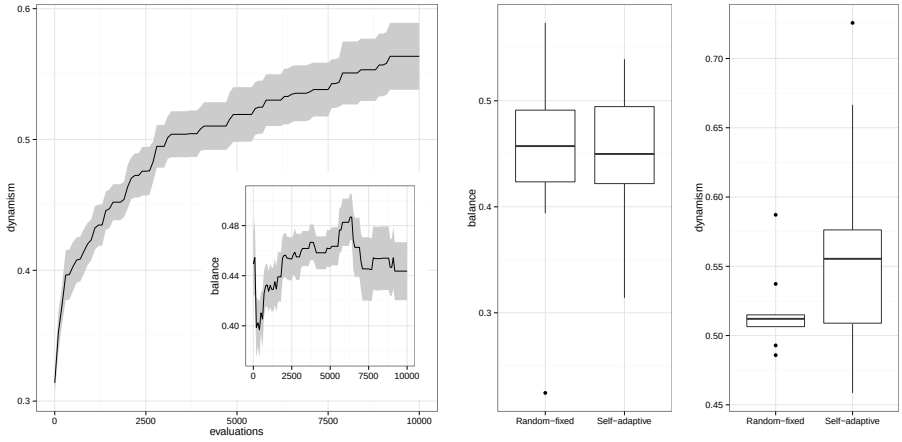
The first batch of experiments focused on analyzing the behavior of the algorithms when optimizing either balance or dynamism. The results are depicted in Fig. 3.

As can be seen, the self-adaptive EA was quite successful in finding highly balanced solutions. In fact it found solutions with perfect (1.0) balance as a median result. The fixed-random variant was also quite successful at this, achieving the

[1] `https://github.com/Manwe56/Manwe56-ai-contest-planet-wars`

[2] `http://flagcapper.com/?c1`

[3] `http://planetwars.aichallenge.org/profile.php?user_id=8490`



(a) (b)

Fig. 4 Optimization of dynamism. (a) Mean dynamism attained in the self-adaptive EA. The inset shows the evolution of balance in the corresponding high-dynamism solutions. In both cases the shaded area indicates the standard error of the mean. (b) Boxplot of the final values of balance and dynamism.

same median result and a slightly better mean result (but with no statistical significance according to a one-sided wilcoxon test, $\alpha = .1$). This can be attributed to the fact that finding highly-balanced maps is quite easy (actually, they are commonly found very early on the run) and hence the use of self-adaptation in the number of planets adds an unnecessary overhead. In fact, the self-adaptive method can be regarded as slightly more exploratory as indicated by the distribution of the dynamism values: as expected, dynamism decreases as balance increases but the final values for the self-adaptive EA were somewhat higher than those of the fixed-random variant.

A salient feature of many of the highly-balanced maps found is the fact that balance was achieved at the expense of complete inaction: both players sit on their home planets and do not attempt to conquer other planets, let alone engage in combat with the opponent. Such solutions are clearly unattractive and underline the inability of balance –as a stand-alone property– to characterize interesting games. We thus turn our attention to the optimization of dynamism. Fig. 4 show the results in this case.

Several qualitative differences with respect to balance optimization are evident. Firstly, the convergence of the algorithm is slower and suggests there is room for improvement if longer runs are allowed. Also, the average fitness of solutions places then close to medium level or half-way between medium and high level. Taken together, this hints at a great difficulty in finding highly-dynamic maps. It should also be noted that the trajectory of balance does not markedly decrease (as was the case of dynamism in balance optimization) but rather it oscillates in a narrow margin. This can partly be due to the fact that our definition of dynamism implicitly incorporates a component of balance via the peak-difference variables: if the game is very imbalanced and one player thoroughly dominates the other, these

variables may take lower values than in games in which the dominated player makes a comeback and thus the game goes through an intermediate stage of balance. The fact that in such highly unbalanced games a victory of one of the players may happen early on the game, also penalizes their dynamism – remember rule (7). Regarding the comparison of the fully self-adaptive EA with the fixed-random variant, Fig. 4b shows that while both algorithms provide maps, the balance of which is almost identically distributed, the fully self-adaptive EA provides more dynamic solutions than its fixed-random counterpart (with statistical significance at $\alpha = .1$ level). The self-adaptability of solution length seems to be helpful in this more difficult optimization scenario.

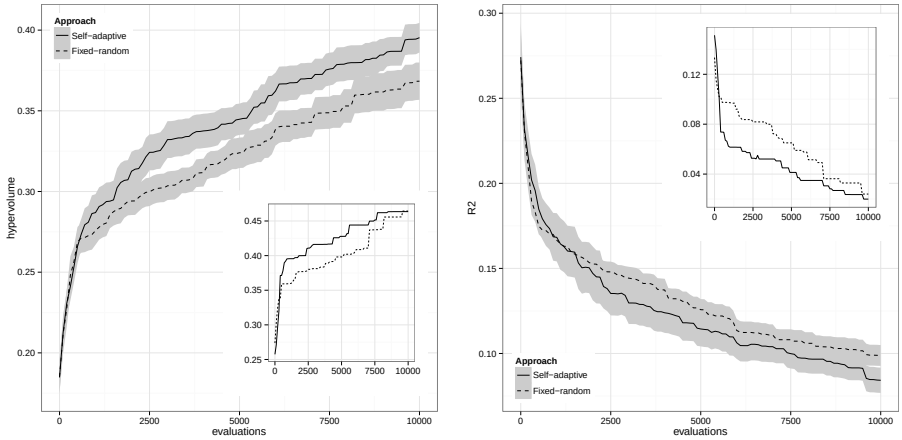
An inspection of these results, and more precisely of the qualitatively-different trajectories in fitness space of either property when the other is being optimized, raises interesting questions as to what exactly are the possible tradeoffs attainable between dynamism and balance. This requires approaching the problem from a multiobjective perspective and is tackled in the next section.

4.2 A Multiobjective Approach

In order to perform multi objective optimization of balance and dynamism used the Non-dominated Sorting Genetic Algorithm II (*NSGA-II*) [3]. As with the single-objective algorithm, we benchmarked the effectiveness of the self-adaptation of the number of planets against another version of the algorithm in which the length of solutions was decided at random at the beginning of each run. The comparison is therefore based on two performance indicators, namely hypervolume [31] and R_2 [12]. The former provides an indication of the region in the fitness space that is dominated by the front (and thus it is to be maximized). This requires the use of a reference point, which in our case is $(0, 0)$ (the multi-objective fitness of the worst hypothetical solution). As for the latter indicator, it estimates the extent to which a given front approximates another one. For this purpose, we constructed a non-dominated front by joining the fronts discovered in all runs of both algorithms and removing dominated solutions. This front was then used as the reference front to which the distance was measured (clearly, this distance is to be minimized).

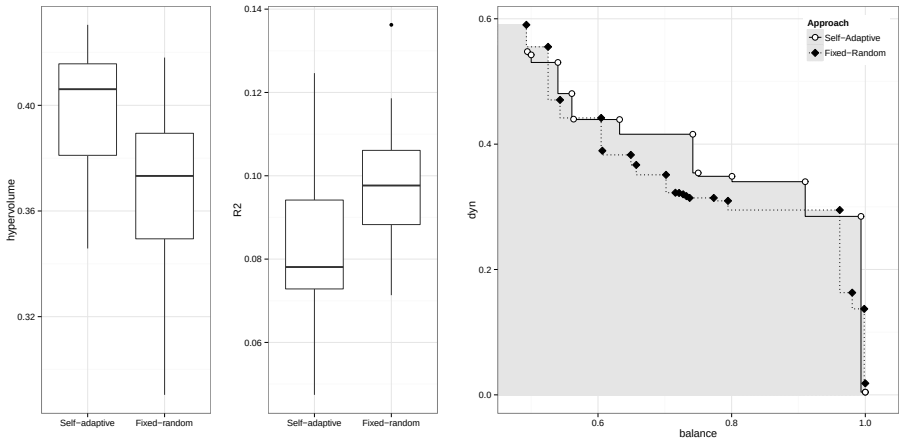
Fig. 5 shows the results. As can be seen, the self-adaptive algorithm had a faster convergence to better values of both indicators. The actual distribution of the final values of these indicators is shown in Fig. 6a. In both cases the difference is favorable to the self-adaptive algorithm (with statistical significance at the $\alpha = .1$ level). A comparison of the cumulated fronts found by each algorithm is shown in Fig. 6b. Quite interestingly, the random-fixed variant seems to perform better on the extremes of the front, whereas the self-adaptive EA is more successful in exploring the central part of the front, where a wide spectrum of different tradeoffs between balance and dynamism are located.

We now look in more detail at the solutions comprised in the non-dominated front and analyze how their properties map a different tradeoff between the two objectives considered. Firstly, consider the number of planets in each map – see Fig. 7a. In general, the number of planets tends to be larger in increasingly dynamic maps. As a matter of fact, the number is close to the upper limit of 30 planets in the left hand side of the front and actually hits the lower limit of 15 several times as we move towards the right. A natural interpretation of lies in the broader set



(a) (b)

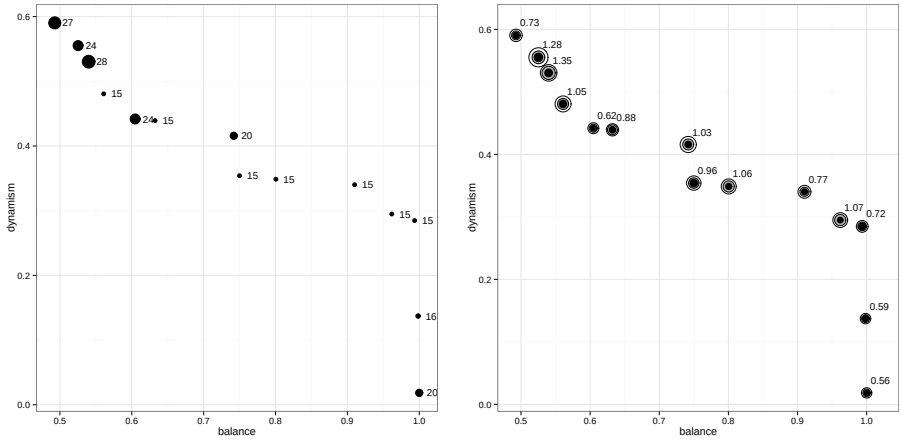
Fig. 5 Evolution of performance indicators of the non-dominated front in each run of the algorithms. The inset shows the same measure for the cumulated front of both algorithms. (a) hypervolume (b) R_2.



(a) (b)

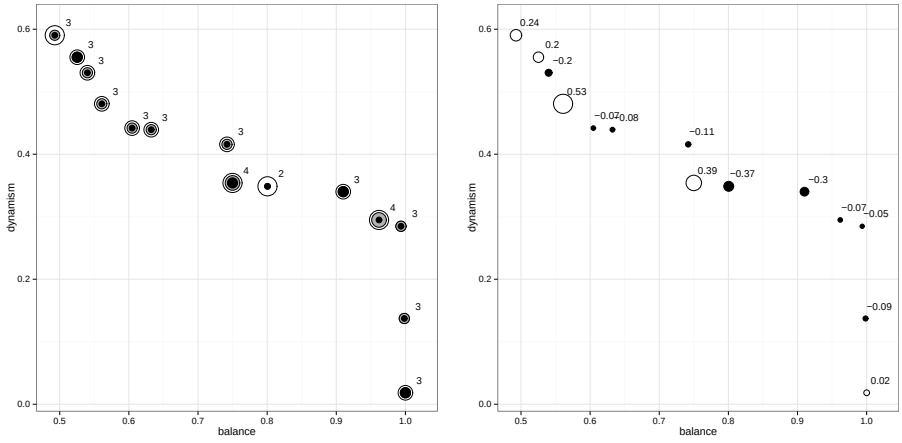
Fig. 6 (a) Boxplot of the values of performance indicators for both algorithms. (b) Comparison of the global non-dominated fronts for both algorithms.

of choices available for players in larger maps, and the possibility they offer for building larger empires or engaging in more fights for planet control. Besides the number of planets, their distribution on the map also differs according to the zone of the front where we are located as demonstrated in Fig. 7b. While the trend is not so marked as in the previous case, we can see that planets are, in general, farther apart from each other in highly-dynamic maps than in highly-balanced maps. Furthermore, their arrangement appears less structured (as indicated by a



(a) (b)

Fig. 7 (a) Number of planets in each solution of the non-dominated front. (b) Distribution of inter-planetary distances. The black circle denotes the 25% percentile, the gray circle the median and the white circle the 75% percentile.



(a) (b)

Fig. 8 (a) Distribution of planetary sizes. (b) Pearson correlation between planet sizes and their initial number of ships.

larger inter-quartile distance, i.e., with higher dispersion in the inner half of the distribution). This higher irregularity contributes to dynamism, likely due to the fact that it makes maps more anisotropic and hence can provide very different stimuli to each player, eliciting actions from either of them which in turn triggers further action by the opponent.

Regarding planetary sizes (Fig. 8a) the median size is analogous at the extremes of the front, but seems to have a greater dispersion towards the high-dynamism

end. Again, the irregularity effect could be at work, promoting dynamism. At any rate, the trend is not very strong. However, an interesting fact emerges when we try to correlate the size of each planet with the initial number of ships placed on them (Fig. 8b). In general this correlation tends to be positive at the high-dynamism end, while it is virtually zero (i.e., very close to it, from above or from below) near the other end. In this case, we may interpret this feature as a safeguard against highly imbalanced scenarios in which large planets can be conquered at a low cost, providing a decisive edge for one of the players in the game.

5 Conclusions

This paper compares several PCG methods for generating maps for *Planet Wars*, an RTS game. These maps should fulfill some desirable requirements, such as balance and dynamism, in order to obtain interesting and appealing games. We have observed that the first objective can be easily misused by the optimization engine, leading to maps that promote absolute inaction of the players (thus achieving perfect balance at the expense of extreme dullness). To the contrary, our definition of dynamism implicitly carries with it a component of balance as well, if only by negation of extreme imbalance (a very imbalanced game ends very early and/or is likely to exhibit less comebacks from one of the players). This way, medium-high dynamism is compatible with medium balance. A multiobjective optimization EA has been used to explore what other tradeoffs between these two objectives are attainable, and has shown a gentle degradation of dynamism as the balance is increased, followed by a sharp decrease of the former upon reaching the high end of balance. We have further studied the different properties of maps providing different balance/dynamism tradeoffs. In general, dynamic games seem to be related to maps featuring a larger number of planets, widely scattered on the map and whose sizes are positively correlated to the initial number of ships. We hypothesize that these features promote dynamism by providing ample stimuli to the players to expand their empires, eventually clashing with each other. Of course, a point of caution is that the particular bots used in the experimentation exert an influence on these results. The fact that we have considered three different, highly competitive bots may make the case for the validity of the argument though. Whatever the case, future work will allow more experimentation in this regard. Other points to be addressed in future work are the inclusion of additional criteria capturing interestingness and aesthetic appeal.

From an algorithmic point of view, we have presented an EA that optimizes maps by self-adapting mutation parameters and map size (i.e., number of planets). We have benchmarked this self-adaptive EA against a variant that features random fixed-length maps. In general the fully self-adaptive EA has been shown to outperform its counterpart both in the optimization of dynamism as a single objective (which is harder than optimizing balance) as in the multiobjective scenario (as measured by the hyper volume and R_2 indicators). We will try to confirm this result on different RTS games in the future. In this regard, the approach presented here can be easily generalized to other RTS scenarios since these often feature bases and resources to be conquered. Last but not least, further study will be dedicated to the actual characterization of balance and dynamic, and how adjusting these, influences the optimization process.

Acknowledgements This work is partially supported by Spanish MICINN under project ANYSELF[4] (TIN2011-28627-C04-01), by Junta de Andalucía under project P10-TIC-6083 (DNEMESIS[5]) and by Universidad de Málaga, Campus de Excelencia Internacional Andalucía Tech.

References

1. Browne, C., Maire, F.: Evolutionary game design. IEEE Trans. Comput. Intellig. and AI in Games **2**(1), 1–16 (2010)
2. Czikszentmihalyi, M.: Flow: The psychology of optimal experience (1990)
3. Deb, K., Agrawal, S., Pratab, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: M. Schoenauer, et al. (eds.) Parallel Problem Solving from Nature VI, *Lecture Notes in Computer Science*, vol. 1917, pp. 849–858. Springer-Verlag, Berlin Heidelberg (2000)
4. Entertainment Software Association: Essential facts about the computer and video game industry (2012). URL `http://www.theesa.com/facts/pdfs/esa_ef_2012.pdf`
5. Fortin, F.A., Rainville, F.M.D., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research **13**, 2171–2175 (2012)
6. Frade, M., de Vega, F., Cotta, C.: Evolution of artificial terrains for video games based on accessibility. In: C. Di Chio, et al. (eds.) Applications of Evolutionary Computation, *Lecture Notes in Computer Science*, vol. 6024, pp. 90–99. Springer-Verlag, Berlin Heidelberg (2010)
7. Frade, M., de Vega, F.F., Cotta, C.: Modelling video games' landscapes by means of genetic terrain programming - a new approach for improving users' experience. In: M. Giacobini, et al. (eds.) Applications of Evolutionary Computing, *Lecture Notes in Computer Science*, vol. 4974, pp. 485–490. Springer-Verlag, Berlin Heidelberg (2008)
8. Frade, M., de Vega, F.F., Cotta, C.: Breeding terrains with genetic terrain programming: The evolution of terrain generators. International Journal of Computer Games Technology **2009** (2009)
9. Frade, M., de Vega, F.F., Cotta, C.: Evolution of artificial terrains for video games based on obstacles edge length. In: IEEE Congress on Evolutionary Computation, pp. 1–8. IEEE (2010)
10. Gearbox Software: Borderlands. 2K Games (2009)
11. Gutiérrez, J.A.G., Cotta, C., Leiva, A.J.F.: Design of emergent and adaptive virtual players in a war RTS game. In: J.M. Ferrández, J.R.Á. Sánchez, F. de la Paz, F.J. Toledo (eds.) 4th International Work-Conference on the Interplay Between Natural and Artificial Computation (IWINAC 2011), *Lecture Notes in Computer Science*, vol. 6686, pp. 372–382. Springer-Verlag, La Palma, Canary Islands, Spain (2011)
12. Hansen, M., Jaszkiewicz, A.: Evaluating the quality of approximations to the nondominated set. Tech. Rep. IMM-REP-1998-7, Institute of Mathematical Modelling Technical University of Denmark (1998)
13. Koster, R.: A theory of fun for game design. Paraglyph Press (2004)
14. Lara-Cabrera, R., Cotta, C., Fernández-Leiva, A.J.: A procedural balanced map generator with self-adaptive complexity for the real-time strategy game planet wars. In: A. Esparcia-Alcázar, et al. (eds.) Applications of Evolutionary Computation 2013, *Lecture Notes in Computer Science*, vol. 7835, pp. 274–283. Springer-Verlag, Berlin Heidelberg (2013)
15. Li, R.: Mixed-integer evolution strategies for parameter optimization and their applications to medical image analysis. Ph.D. thesis, Leiden University (2009)
16. Lucas, S.M.: Computational intelligence and ai in games: A new ieee transactions. IEEE Trans. Comput. Intellig. and AI in Games **1**(1), 1–3 (2009)
17. Lucas, S.M., Mateas, M., Preuss, M., Spronck, P., Togelius, J.: Artificial and Computational Intelligence in Games (Dagstuhl Seminar 12191). Dagstuhl Reports **2**(5), 43–70 (2012)

[4] `http://anyself.wordpress.com/`

[5] `http://dnemesis.lcc.uma.es/wordpress/`

18. Mahlmann, T., Togelius, J., Yannakakis, G.N.: Spicing up map generation. In: C.D. Chio, et al. (eds.) Applications of Evolutionary Computation, *Lecture Notes in Computer Science*, vol. 7248, pp. 224–233. Springer-Verlag, Málaga, Spain (2012)
19. Malone, T.: What makes things fun to learn? heuristics for designing instructional computer games. In: Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems, vol. 162, pp. 162–169. ACM (1980)
20. Maxis: Spore. Electronic Arts (2008)
21. Mojang: Minecraft. Mojang (2011)
22. Nogueira, M., Cotta, C., Fernández-Leiva, A.J.: On modeling, evaluating and increasing players' satisfaction quantitatively: Steps towards a taxonomy. In: C.D. Chio, et al. (eds.) Applications of Evolutionary Computation, *Lecture Notes in Computer Science*, vol. 7248, pp. 245–254. Springer-Verlag, Málaga, Spain (2012)
23. Rudolph, G.: An evolutionary algorithm for integer programming. In: Y. Davidor, H.P. Schwefel, R. Männer (eds.) Parallel Problem Solving from Nature III, *Lecture Notes in Computer Science*, vol. 866, pp. 139–148. Springer-Verlag, Jerusalem, Israel (1994)
24. Szita, I., Ponsen, M.J.V., Spronck, P.: Effective and diverse adaptive game AI. IEEE Trans. Comput. Intellig. and AI in Games **1**(1), 16–27 (2009)
25. Togelius, J., De Nardi, R., Lucas, S.: Towards automatic personalised content creation for racing games. In: Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on, pp. 252–259 (2007)
26. Togelius, J., Preuss, M., Beume, N., Wessing, S., Hagelback, J., Yannakakis, G.: Multiobjective exploration of the starcraft map space. In: Computational Intelligence and Games (CIG), 2010 IEEE Symposium on, pp. 265 –272 (2010)
27. Togelius, J., Preuss, M., Yannakakis, G.N.: Towards multiobjective procedural map generation. In: Proceedings of the 2010 Workshop on Procedural Content Generation in Games, pp. 3:1–3:8 (2010)
28. Togelius, J., Yannakakis, G.N., Stanley, K.O., Browne, C.: Search-based procedural content generation: A taxonomy and survey. IEEE Transactions on Computational Intelligence and AI in Games **3**(3), 172–186 (2011)
29. Yannakakis, G.: How to model and augment player satisfaction: A review. In: Proceedings of the 1st Workshop on Child, Computer and Interaction, ACM Press (2008)
30. Yannakakis, G.N.: Game AI revisited. In: Proceedings of the 9th conference on Computing Frontiers, CF '12, pp. 285–292. ACM, New York, NY, USA (2012)
31. Zitzler, E., Thiele, L.: Multiobjective optimization using evolutionary algorithms - a comparative case study. In: A.E. Eiben, et al. (eds.) Parallel Problem Solving from Nature V, *Lecture Notes in Computer Science*, vol. 1498, pp. 292–301. Springer-Verlag, Berlin Heidelberg (1998)