Chaos, Solitons & Fractals 73 (2015) 29–35

Contents lists available at ScienceDirect

Chaos, Solitons & Fractals

Nonlinear Science, and Nonequilibrium and Complex Phenomena

journal homepage: www.elsevier.com/locate/chaos

Particle swarm optimization for automatic creation of complex graphic characters

Iztok Fister Jr. [a], Matjaž Perc [b,c,*], Karin Ljubič [d], Salahuddin M. Kamal [c], Andres Iglesias [e,f], Iztok Fister [a]

[a] Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, SI-2000 Maribor, Slovenia
[b] Faculty of Natural Sciences and Mathematics, University of Maribor, Koroška cesta 160, SI-2000 Maribor, Slovenia
[c] Faculty of Science, King Abdulaziz University, Jeddah, Saudi Arabia
[d] Faculty of Medicine, University of Maribor, Taborska 8, SI-2000 Maribor, Slovenia
[e] Department of Applied Mathematics and Computational Sciences, University of Cantabria, Avda. de los Castros, 39005 Santander, Spain
[f] Department of Information Science, Faculty of Sciences, Toho University, 2-2-1 Miyama, 274-8510 Funabashi, Japan

ARTICLE INFO

Article history:
Received 25 November 2014
Accepted 22 December 2014

ABSTRACT

Nature-inspired algorithms are a very promising tool for solving the hardest problems in computer sciences and mathematics. These algorithms are typically inspired by the fascinating behavior at display in biological systems, such as bee swarms or fish schools. So far, these algorithms have been applied in many practical applications. In this paper, we present a simple particle swarm optimization, which allows automatic creation of complex two-dimensional graphic characters. The method involves constructing the base characters, optimizing the modifications of the base characters with the particle swarm optimization algorithm, and finally generating the graphic characters from the solution. We demonstrate the effectiveness of our approach with the creation of simple snowman, but we also outline in detail how more complex characters can be created.

1. Introduction

Computer games are one of the most profitable products of the interactive entertainment industry. Worldwide, the number of active users is staggering, and this largely regardless of age, gender and ethnicity. The positive trend in the number of user since the 80’s has been spurred on further with the advent of massive online multi-player games. The demand on the technology to deliver ever more authentic and exhilarating user experience is of course also increasing, and this creates a continuous need for innovation and improvement at many different levels of the industry. In addition to the obvious importance of hardware improvement, the development of new software, new interaction possibilities, as well as a firm positioning in the market all demand expertise and skill. At the juncture of these different aspects of development, theoretical research in computer science, applied mathematics and complex systems also plays a key role. And it is to the latter subject in particular that we aim to contribute with this paper.

In the past decades, the methods and techniques employed first in computer games have been applied with notable success in computational agent-based system research [21]. Many studies have also looked at video games as an art form on its own right [8]. At some universities and research institutes, computer games have become the subject on which one could do a masters or a PhD. Most notably, however, video games have had the most significant impact on the development of personal computers. Due to popular demand, personal computers had to improve steadily in the amount of RAM, processor

* Corresponding author at: Faculty of Natural Sciences and Mathematics, University of Maribor, Koroška cesta 160, SI-2000 Maribor, Slovenia.
E-mail address: matjaz.perc@um.si (M. Perc).

http://dx.doi.org/10.1016/j.chaos.2014.12.019
0960-0779/© 2015 Elsevier Ltd. All rights reserved.

speed, display resolution and graphic card capabilities to keep with the pace of more and more sophisticated games.

Simultaneously with the development of hardware, software methods have also arisen for solving the hardest real-world problems. As many of these problems are NP-hard [19], many heuristical algorithms have been proposed for solving such problems approximately. The majority of these heuristics take inspiration from nature. Two branches of algorithms are particularly known for their close ties with nature, namely the evolutionary algorithms (EA) [9] and swarm intelligence algorithms (SI) [4]. The former are inspired by the Darwinian principle of "only the fittest survive" [26], while the later take from social insects, swarms of birds and flocks of fish to execute complex actions based on elementary and indeed very simple interactions among individuals [38]. An intersection of both is the emergence of collective behavior in evolutionary games [28].

In general, it has been established that using advanced nature-inspired algorithms in computer graphics can be very useful. Applications to this domain include the EA in [33,34], where genetic algorithms were introduced to computer graphics. Later on, other evolutionary algorithms were also applied [14,2], while the most recent studies include also genetic algorithms [18] for polynomial B-spline surface reconstruction, particle swarm optimization [16] for the same problem and particle swarm optimization for Bézier surface reconstruction [15]. For example, Gálvez and Iglesias [17] applied the firefly algorithm [10,11] for polynomial Bézier surface parameterization, which is a well known problem in computer graphics. Further to these advances, creative 3D shape modeling has been proposed, where an initial population of 3D models is evolved to produce novel shapes in the next generation [40]. A new emerging field termed search-based procedural content generation is also growing rapidly, the aim of which is to use metaheuristics in automatic generation of content for games [36].

Since many games operate with a large number of different characters, we here address a very basic problem in a new way. In particular, we explore the usefulness of particle swarm optimization (PSO) for the automatic creation of complex graphic characters. We show how a modified particle swarm optimization algorithm can be used to create a gallery of 2D characters with different colors and shapes using a predefined set of parts. For the approach to work, the problem is defined as constraint satisfaction problem (CSP). Each candidate solution that satisfies all constraints is placed in a gallery from which the best characters can be selected by the game designer. As an example, we show the generation of 2D snowman, but we also outline the approach for the creation of 3D and more complex characters.

The remainder of this paper is structured as follows. Section 2 reviews the basics of the constraint satisfaction problem. In Section 3, we present particle swarm optimization algorithm for the automatic creation of graphic characters. The experiments and results are presented in Section 4. We conclude with a summary and directions for future development, as well as with a broader discussion about the relevance of nature-inspired algorithms for the emergence of intelligent design and cognition [29].

2. Constraint satisfaction problems

The majority of intractable (NP-hard) real-world problems are constrained [19]. That means, not all of candidate solutions obtained in EA and SI are valid after performing various variation operators. Usually, these operators act blindly according to constraints, i.e., these do not provide valid solutions. The constraint satisfaction problems (CSP) consist of three components $\langle \mathbf{x}, \mathbf{D}, \mathbf{C} \rangle$ [30], where.

- $\mathbf{x}$ is a vector of variables $\mathbf{x} = (x_1, \ldots, x_n)$, where n denotes a dimension of the problem,
- $\mathbf{D}$ is a set of domains $\mathbf{D} = \{D_1, \ldots, D_n\}$ for each variable of $\mathbf{x}$,
- $\mathbf{C}$ is a set of constraints $\mathbf{C} = \{C_1, \ldots, C_m\}$ that specify allowable combinations of values.

Each domain is determined by its lower and upper bounds $D_i \in [lb_i, ub_i]$. Each constraint C_i consists of a pair $\langle scope, relation \rangle$, where *scope* determines variables that participate in the constraint and *relation* defines an relation that needs to be valid in order to satisfy the so named feasibility condition ϕ. The feasibility condition for each candidate solution is composed of constraints and can be expressed as conjunction of constraints $\phi(\mathbf{x}) = \wedge_{i=1\ldots m} C_i(\mathbf{x})$. In fact, the feasibility condition is true if and only if all constraints are satisfied.

There are two main types of constraints handling in EA and SI [9]:

- Indirect, where constraints are transformed into optimization objectives,
- Direct, where the population based algorithms keep alone account about constraints during the run.

In this paper, indirect constraint handling was considered, where the proper solution is found, when the feasibility condition is satisfied. In this case, the value of fitness function is zero, i.e., $\phi(\mathbf{x}) = true$ if and only if $f(\mathbf{x}) = 0$.

3. Automatic creation of graphic characters

Particle swarm optimization (PSO) was one of the oldest swarm intelligence algorithms that was introduced at an International conference on Neural Networks by Kennedy and Eberhart in 1995 [23]. PSO is inspired by the social foraging behavior of some animals such as flocking behavior of birds and schooling behavior of fish. There are some individuals with better developed instinct for finding food, in both animal species. According these individuals, the whole swarm is directed into more promising regions in the fitness landscape.

The PSO is a population-based algorithm that consists of particles $\mathbf{x}_i^{(t)} = (x_{i,1}, \ldots, x_{i,n})^T$ for $i = 1 \ldots Np$ representing their position in a n-dimensional search space. Thus, the variable Np limits the number of particles in the population. The particles move across the search space with velocity $\mathbf{v}_i^{(t)} = (v_{i,1}, \ldots, v_{i,n})^T$ according to the position of the best particle $\mathbf{x}_{best}^{(t)}$ towards the more promising regions in the search space. However, this movement is also dependent on the local best position of each particle $\mathbf{p}_i^{(t)}$ and can mathematically be expressed, as follows:

$$\mathbf{v}_i^{(t+1)} = \mathbf{v}_i^{(t)} + c_1 r_1^{(t)} \left(\mathbf{p}_i^{(t)} - \mathbf{x}_i^{(t)}\right) + c_2 r_2^{(t)} \left(\mathbf{x}_{best}^{(t)} - \mathbf{x}_i^{(t)}\right). \tag{1}$$

Then, the new particle position is calculated according to the expression

$$\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \mathbf{v}_i^{(t)}. \tag{2}$$

Pseudo-code of the PSO algorithm is illustrated in Algorithm 1.

Algorithm 1. Pseudo code of the PSO algorithm

```
Input: PSO population of particles $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,n})^T$
    for $i = 1 \ldots Np$.
Output: The best solution $\mathbf{x}_{best}$ and its corresponding
    value $f_{min} = \min(f(\mathbf{x}))$.
1: init_particles;
2: $eval = 0$;
3: While termination_condition_not_meet do
4:    for $i = 1$ to $Np$ do
5:       $f_i$ = evaluate_the_new_solution ($\mathbf{x}_i$);
6:       $eval = eval + 1$;
7:       if $f_i \leqslant pBest_i$ then
8:          $\mathbf{p}_i = \mathbf{x}_i$; $pBest_i = f_i$;  // save the local best
    solution
9:       end if
10:      if $f_i \leqslant f_{min}$ then
11:         $\mathbf{x}_{best} = \mathbf{x}_i$; $f_{min} = f_i$;  // save the global best
    solution
12:      end if
13:      $\mathbf{x}_i$ = generate_new_solution ($\mathbf{x}_i$);
14:   end for
15: end while
```

After finishing the initialization in function 'init_particles' (Algorithm 1), the main loop starts in which the evaluation of fitness function takes place ('evaluate_the_new_solution' function) and the new position of particle is calculated according to Eqs. (1) and (2) ('generate_new_solution' function). The former determines the quality of solution, while the latter moves the particle to a new possible better region of the search space [32,31]. However, between both functions the code for saving the local best as well as global best solution is executed. Note that the main loop is terminated when the termination condition (in line 3) is satisfied. Usually, the number of generation *MAX_GEN* is used for these purposes. Over the past years, also some PSO variants were proposed [12,44,42,24,41,22,6,27].

Particle swarm optimization was also used in many practical and real-world applications. It was used in electromagnetics applications [7], spam detection [43], electric power systems [1] and many more. On the other hand, PSO was also a good candidate for solving problem of traveling salesman person [39] which is a problem of discrete optimization.

3.1. Applying particle swarm optimization

The generation of characters is defined as a constraint satisfaction problem (CSP), where the proper solution is found, when the conjunction of constraints $\phi(\mathbf{x}) = \wedge_{i=1\ldots m} C_i(\mathbf{x})$ is true. Here, the graphic characters are divided into integral parts that are saved into a set of parts. Each part can be colored with different colors and can include various strokes, i.e., a shape outline with stroke width, line style and color. The former has an impact on characters look, while the later on a its shape. Generally, a development of automatic generation of graphic characters can be divided into three phases:

- Construction of the base character,
- Optimizing the modifications of the base character with the PSO algorithm,
- Generation of graphic characters from the solutions.

As an example, an automatic generation of snowman graphics was applied in this paper. The motivation for generation snowman graphics was that these graphics are relatively easy to construct, while all problems with which a designer is confronted by this generation can be identified. In the remainder of the paper generation of snowman graphics in the light of the mentioned problems are illustrated in details.

3.1.1. Construction of the base character

In the first phase, the base character is constructed that is then divided into integral parts. These parts that can be a subject of modification constitutes a set of parts and share the same characteristics (e.g., that are colored with the same color). However, the size of this set depends on the complexity of graphic characters. More complex the graphic character, greater the number of parts and more complex the shape.

For example, a snowman in Fig. 1 consists of 20 parts and 4 strokes, as follows: the lower part, the middle part, the upper part, the head, the hat, the left and right hand. The greater parts, like the lower, middle, upper parts and the hat are outlined with a stroke of various thickness. Each hand completes with the palm. The interior of the middle part of the snowman hull is filled with three buttons, while the head consists of two eyes, nose and mouth. The eyes and the nose are trimmed with strokes of fixed thickness, while the mouth are fixed.

3.1.2. Optimization of modifications of the base character

This phase can be performed using different optimization algorithms. In this study, the PSO algorithm was applied, where each modifications of the base characters are represented as a vector $\mathbf{x}_i = \{x_{i,1}, \ldots, x_{i,n}\}$ for $i = 1 \ldots Np$ of length $n = 24$ with real-valued elements in the interval $x_{i,j} \in [0, 1]$. The elements $x_{i,1}, \ldots, x_{i,16}$ represent the color of defined part, and tuples $\langle x_{i,17}, x_{i,18} \rangle, \ldots, \langle x_{i,23}, x_{i,24} \rangle$ the strokes of four greater parts of snowman. Here, the even elements of tuples denote colors, while

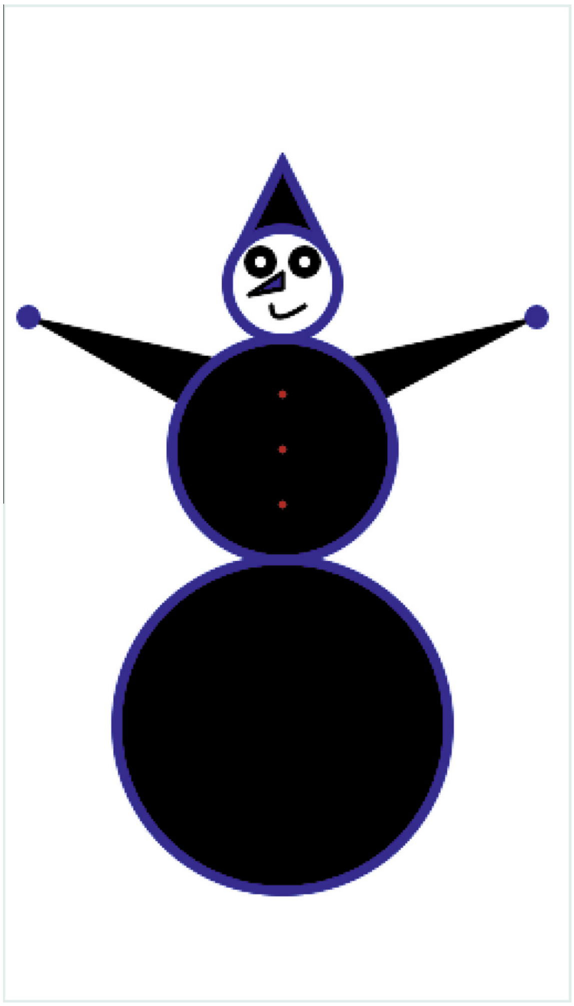

Fig. 1. A simple snowman, which we use as the benchmark for our theory. We have used the snowman for its simple shape to demonstrate the correctness of our method. Moreover, we have used neutral colors to highlight the parts that are subject to change.

Table 1
An example of a simple encoding scheme for colors and widths of the character.

Lower bound	Upper bound	Colors (Δ)	Widths
0.0	0.1	Orange	1
0.1	0.2	Black	2
0.2	0.3	Blue	3
0.3	0.4	Red	4
0.4	0.5	Yellow	5
0.5	0.6	Green	6
0.6	0.7	Magenta	7
0.7	0.8	Cyan	8
0.8	0.9	Aqua	9
0.9	1.0	Coral	10

odd widths. Mapping of the colors and widths is implemented according to encoding scheme as represented in Table 1. Note that as higher the number of widths, the thicker the stroke.

The automatic generation of graphic characters is the CSP problem, where the proper solution is found, when the feasibility condition $\phi(\mathbf{x}) = true$. Using the fitness function, the feasibility condition is expressed as

$$f(\mathbf{x}) = \sum_{i=1}^{m} \chi(C_i(\mathbf{x})), \tag{3}$$

where

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } C_i(\mathbf{x}) \text{ is false}, \\ 0 & \text{otherwise}. \end{cases} \tag{4}$$

In line with this, the fitness function counts the number of constraint violations. Normally, the constraints are defined by designer and depends on users demands and desires. In case of snowman characters, the following constraints were defined:

- The number of colors needed for coloring the snowman parts must be more or equal to 4. This constraint can be formally described as $|S_1| \geqslant 4$, where the set S_1 is expressed as
  $$S_1 = \{\forall x_i \forall x_j | \Delta x_i \neq \Delta x_j \text{ for } i = 1 \ldots 16 \wedge j = 1 \ldots 16 \wedge i \neq j\},$$
  where Δ denotes a color of the parts x_i and x_j as illustrated in Table 1.
- There must be at least 4 different strokes. This constraint can be formally described as $|S_2| \geqslant 4$, where the set S_2 is defined as
  $$S_2 = \{\forall x_i \forall x_j | \Delta x_i \neq \Delta x_j \text{ for } i = 17 \ldots 24 \wedge j = 17 \ldots 24 \wedge i \neq j \wedge (i \mod 2) = 1\}.$$
- At least one stroke must have a value more than 5. Mathematically, this constraint can be written as $|S_3| > 5$, where the set S_3 is defined as
  $$S_3 = \{\exists x_i | \Delta x_i > 5 \text{ for } i = 17 \ldots 24 \wedge (i \mod 2) = 1\}.$$
- At least one stroke must have a value less than 5. The constraint can mathematically be expressed as $|S_4| < 5$, where the set S_4 is defined as
  $$S_4 = \{\exists x_i | \Delta x_i < 5 \text{ for } i = 17 \ldots 24 \wedge (i \mod 2) = 1\}.$$
- The color of snowman eyes and the color of the snowman face cannot have the same color. This constraint can directly be written as a conjunction of constraints, in other words $\Delta x_{14} \neq \Delta x_6 \wedge \Delta x_{16} \neq \Delta x_6$, where x_{14} and x_{16} denotes eyes and x_6 the snowman face (the upper part of snowman torsion).

Using the described representation of solutions evaluated by the fitness function represented in Eq. (3), the original PSO algorithm as illustrated in Algorithm 1 can be applied to the problem for automatic generation of graphic characters.

3.1.3. Generation of graphic characters

The solution generated by the PSO algorithm describes how to combine the colors and strokes of the snowman parts to get the unique visual presentation of 2D graphic character. Unfortunately, this description has nothing to do with the real graphical presentation of 2D snowman.

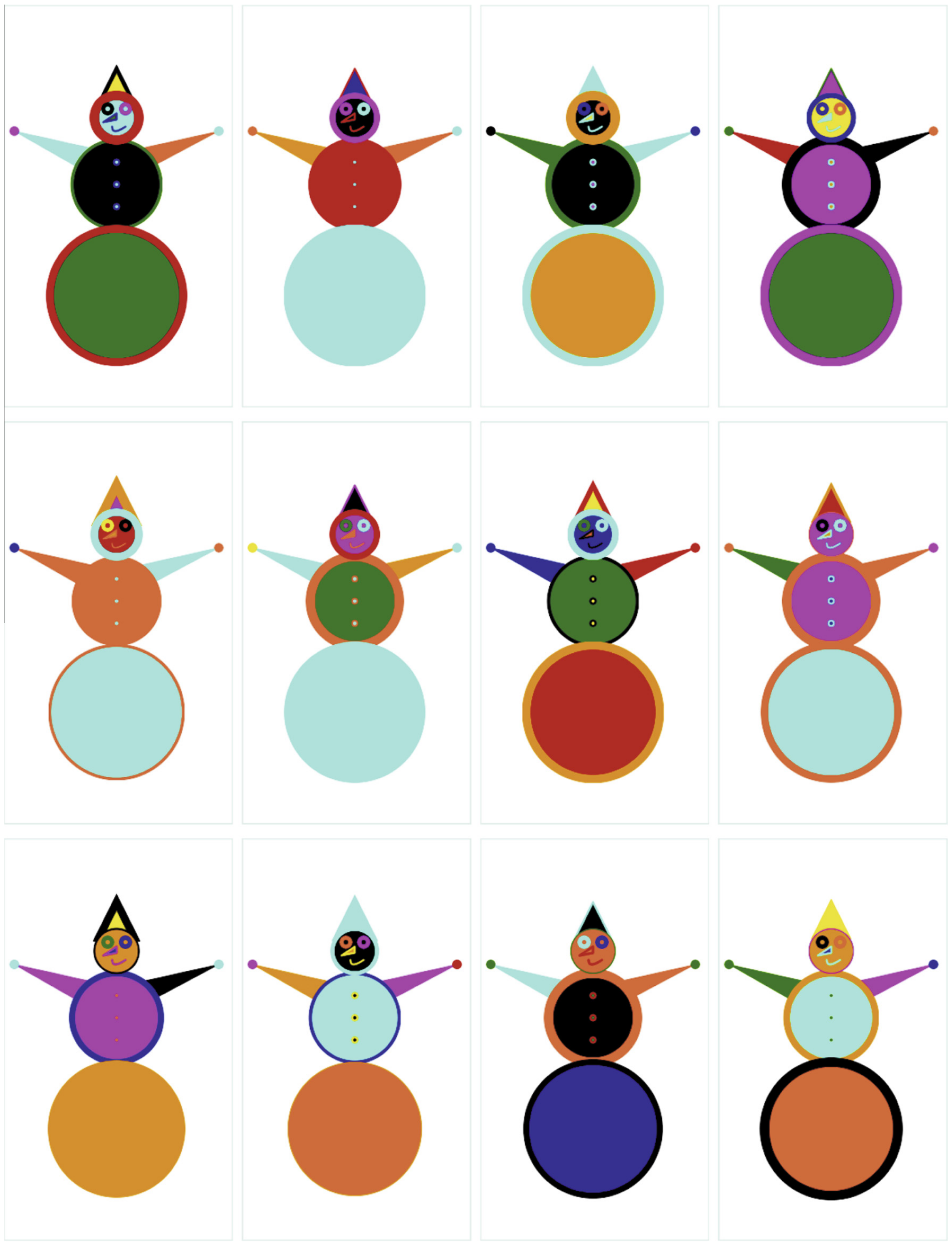

Fig. 2. A gallery of automatically created snowman, made with the particle swarm optimization algorithm. Only those snowman are depicted that satisfy all the criteria in the constraints set. The selection of the best graphic is, however, left to the judgment of the developer or user.

In order to plot the real 2D snowman graphic, an interpreter in Ruby programming language [13] was developed. Modifications can be directly transformed to call of graphical function from RMagick library [35,3]. Moreover, Ruby enables performing any operation tackling the snowman graphic characters within its script. Because the interpreter is easy to implement, the detailed description of it is left out of the scope of this paper.

4. Experiments and results

The goal of our experimental work was to prove that the proposed method for automatic generation of the graphic 2D characters with the PSO algorithm could be applied to real-world video game development as well. In line with this, the set of graphical snowman parts was defined. On that basis, the PSO algorithm was developed for solving the constraint satisfaction problem of automatic generation of snowman 2D graphics. The solution of the problem was found when all constraints were satisfied as posed by designer. Finally, the proper solutions were added to a gallery of snowman graphic characters.

In the experiments, an implementation of classical PSO algorithm in Ruby was taken from author's Github repository [5]. The parameter settings of the PSO algorithm was during experiments, as follows. The values of velocities were limited to the interval $v_i \in [-1.1]$, the population size was set to $Np = 10$, maximal velocity to $v_{\max} = 100$, and the constriction coefficients to $c_1 = c_2 = 2.0$. As a termination condition, the number of generation 100 was set in the case that the proper solution cannot be found. The small population size is applied because of the faster convergence of the PSO algorithm.

A gallery of properly generated snowman graphic characters is illustrated in Fig. 2 from which it can be observed that a lot of different solutions was obtained by the PSO algorithm. What is the best solution here? The running cycle of this PSO algorithm is slightly different from runs of the classical stochastic algorithms (like EA and SI), where normally the best solution is taken after satisfying the specific termination condition, (i.e., after the maximum number of generation, the maximum number of function evaluations, etc.). The best solutions in more independent runs are then evaluated according to statistical measures, as minimum, maximum, average, and standard deviation values.

The characteristics of this problem are that the solution can be found quickly. On the other hand, these solutions are not the best solutions per se. Each of the proper solutions can be viewed as the best depending on the observer's point of view. Unfortunately, this point of view can be very subjective. Therefore, all the proper solutions are aggregated into a gallery and evaluated later by ordinary users, i.e., video game developers. The video game developers are the only definitive point of reference for this assessment because they know also the context in which the generated characters are supposed to act.

5. Discussion

Computer games are at the heart of mass entertainment, driving not just hardware development, but pushing also the limits of software development. In fact, nowhere else are hardware and software development so closely interrelated. Video games are becoming increasingly sophisticated, and the need is clearly there to aid this development with nature-inspired algorithms. Clearly, no man-made system achieves the complexity seen in nature, so the next best thing is to learn from nature to aid artificial design.

The main contribution of our paper to this area is to help game designers facilitate the difficult task of designing large groups of characters that arise usually in historical, sports, and war-game settings. A new, highly efficient method for automatics generation of graphic characters has been proposed based on the particle swarm optimization algorithm, which is able to generate characters of arbitrary complexity. The results presented confirm that we are on the right path towards improving existing algorithms, as well as harvesting the knowledge of complexity science for use in the entertainment industry.

On a related note, which ought to be of interest to individuals working on complex systems, collective behavior and cognition, the question is if engineers looking for the best algorithms to design computer games benefit from the swarm intelligence, does it make sense to imagine that research work of the same type may help us to deepen our understanding of the origin of intelligence and cognition? [37,25,20]. While it is impossible at this point to make arguments precise, at least the conceptual link between particle swarm optimization and other nature-inspired algorithms and the emergence of intelligence is evidently there. In fact, the very use of these algorithms leads one to conclude that we are using intelligence to create new intelligence, and it may be along these lines that inspiring new discoveries could be made.

Acknowledgments

This project was funded by the Deanship of Scientific Research (DSR), King Abdulaziz University, under Grant No. (76-130-35-HiCi). The authors, therefore, acknowledge technical and financial support of KAU. Financial support from the Computer Science National Program of the Spanish Ministry of Economy and Competitiveness, Project Ref. TIN2012-30768 is also acknowledged.

References

[1] AlRashidi MR, El-Hawary ME. A survey of particle swarm optimization applications in electric power systems. IEEE Trans Evol Comput 2009;13:913–8.
[2] Bentley PJ. Exploring component-based representations-the secret of creativity by evolution? Evolutionary Design and Manufacture. Springer; 2000. p. 161–72.
[3] Berube D. Manipulating images with rmagick. Practical ruby gems 2007; p. 191–8.
[4] Blum C, Li X. Swarm intelligence in optimization. In: Blum C, Merkle D, editors. Swarm intelligence: introduction and applications. Berlin: Springer Verlag; 2008. p. 43–86.
[5] Brownlee J. Clever algorithms: nature-inspired programming recipes. Lulu; 2011.
[6] Chaturvedi KT, Pandit M, Srivastava L. Self-organizing hierarchical particle swarm optimization for nonconvex economic dispatch. IEEE Trans Power Syst 2008;23:1079–87.
[7] Ciuprina G, Ioan D, Munteanu I. Use of intelligent-particle swarm optimization in electromagnetics. IEEE Trans Magn 2002;38:1037–40.
[8] Crawford C. Art of computer game design: reflections of a master game designer. U.S.: Osborne/McGraw-Hill; 1984.

[9] Eiben AE, Smith JE. Introduction to evolutionary computing. Berlin: Springer-Verlag; 2003.
[10] Fister I, Fister Jr I, Yang X-S, Brest J. A comprehensive review of firefly algorithms. Swarm Evol Comput 2013;13:34–46.
[11] Fister Jr I, Perc M, Kamal SM, Fister I. A review of chaos-based firefly algorithms: perspectives and research challenges. Appl Math Comput 2015. http://dx.doi.org/10.1016/j.amc.2014.12.006.
[12] Fister Jr I, Yang X-S, Ljubič K, Fister D, Brest J, Fister I. Towards the novel reasoning among particles in pso by the use of RDF and SPARQL. The Sci World J 2014. Article ID: 121782.
[13] Flanagan D, Matsumoto Y. The ruby programming language. O'Reilly Media Inc; 2008.
[14] Funes P, Pollack J. Evolutionary body building: adaptive physical designs for robots. Artif Life 1998;4:337–57.
[15] Gálvez A, Cobo A, Puig-Pey J, Iglesias A. Particle swarm optimization for Bézier surface reconstruction. Computational science–ICCS 2008. Springer; 2008. p. 116–25.
[16] Gálvez A, Iglesias A. Particle swarm optimization for non-uniform rational b-spline surface reconstruction from clouds of 3d data points. Inf Sci 2012;192:174–92.
[17] Gálvez A, Iglesias A. Firefly algorithm for polynomial Bézier surface parameterization. J Appl Math 2013. Article ID: 237984.
[18] Gálvez A, Iglesias A, Puig-Pey J. Iterative two-step genetic-algorithm-based method for efficient polynomial b-spline surface reconstruction. Inf Sci 2012;182:56–76.
[19] Garey MR, Johnson DS. Computers and intractability: a guide to the theory of NP-completeness. New York, USA: W.H. Freeman & Co.; 1979.
[20] Grigolini P, Chialvo DR. Editorial: emergent critical brain dynamics. Chaos, Solitons Fractals 2013;55:1–2.
[21] Helbing D. Social self-organization: agent-based simulations and experiments to study emergent social behavior. Berlin: Springer-Verlag; 2012.
[22] Jiang Y, Hu T, Huang C, Wu X. An improved particle swarm optimization algorithm. Appl Math Comput 2007;193:231–9.
[23] Kennedy J, Eberhart R. Particle swarm optimization. In: Proceedings of IEEE international conference on neural networks. vol. 4. Australia: Perth; 1995. p. 1942–8.
[24] Liang J, Qin K, Suganthan PN, Baskar S. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Trans Evol Comput 2006;10:281–95.
[25] Lukovic M, Vanni F, Svenkeson A, Grigolini P. Transmission on information at criticality. Physica A 2014;416:430–8.
[26] Nowak MA. Evolutionary dynamics. Cambridge, MA: Harvard University Press; 2006.
[27] Panigrahi BK, Pandi VR, Das S. Adaptive particle swarm optimization approach for static and dynamic economic load dispatch. Energy Convers Manage 2008;49:1407–15.
[28] Perc M, Grigolini P. Collective behavior and evolutionary games – an introduction. Chaos, Solitons Fractals 2013;56:1–6.
[29] Rosenfeld S. Global consensus theorem and self-organized criticality: unifying principles for understanding self-organization, swarm intelligence and mechanisms of carcinogenesis. Gene Regul Syst Biol 2013;7:23–39.
[30] Russell S, Norvik P. Artificial intelligence a modern approach. Upper Saddle River, US: Prentice Hall; 2010.
[31] Shi Y, Eberhart R.A modified particle swarm optimizer. In IEEE World Congress on Computational Intelligence, pages 69–73. IEEE, 1998.
[32] Shi Y. Particle swarm optimization: developments, applications and resources. Proceedings of the congress on evolutionary computation 2001, vol. 1. IEEE; 2001. p. 81–6.
[33] Sims K. Artificial evolution for computer graphics, vol. 25. ACM; 1991.
[34] Sims K. Evolving virtual creatures. Proceedings of the 21st annual conference on computer graphics and interactive techniques. ACM; 1994. p. 15–22.
[35] Still M. Rmagick: imagemagick programming with ruby. The Definitive Guide to ImageMagick 2006; p. 301–10.
[36] Togelius J, Yannakakis GN, Stanley KO, Browne C. Search-based procedural content generation: a taxonomy and survey. IEEE Trans Comput Intell AI in Games 2011;3(3):172–86.
[37] Vanni F, Lukovic M, Grigolini P. Criticality and transmission of information in a swarm of cooperative units. Phys Rev Lett 2011;107:078103.
[38] Vicsek T, Zafeiris A. Collective motion. Phys Rep 2012;517:71–140.
[39] Wang KP, Huang L, Zhou CG, Pang W. Particle swarm optimization for traveling salesman problem. International conference on machine learning and cybernetics, vol. 3. IEEE; 2003. p. 1583–5.
[40] Xu K, Zhang H, Cohen-Or D, Chen B. Fit and diverse: set evolution for inspiring 3d shape galleries. ACM Trans Graphics 2012;31:57.
[41] Yang X-S, Deb S, Fong S. Accelerated particle swarm optimization and support vector machine for business optimization and applications. Networked digital technologies. Springer; 2011. p. 53–66.
[42] Zhang Y, Jun Y, Wei G, Wu L. Find multi-objective paths in stochastic networks via chaotic immune PSO. Expert Syst Appl 2010;37(3): 1911–9.
[43] Zhang Y, Wang S, Phillips P, Ji G. Binary pso with mutation operator for feature selection using decision tree applied to spam detection. Knowl Based Syst 2014;64:22–31.
[44] Zhang Y, Wu L. A robust hybrid restarted simulated annealing particle swarm optimization technique. Adv Comput Sci Appl 2012;1(1):5–8.