SILVEREYE – The Implementation of Particle Swarm Optimization Algorithm in a Design Optimization Tool

Chapter · June 2017

DOI: 10.1007/978-981-10-5197-5_9

CITATIONS
25

READS
3,556

4 authors:

Judyta Cichocka
Saule Technology
14 PUBLICATIONS **73** CITATIONS
SEE PROFILE

Agata Migalska
Alphamoon
10 PUBLICATIONS **112** CITATIONS
SEE PROFILE

Will Neil Browne
Queensland University of Technology
197 PUBLICATIONS **4,789** CITATIONS
SEE PROFILE

Edgar Rodriguez Ramirez
Victoria University of Wellington
41 PUBLICATIONS **160** CITATIONS
SEE PROFILE

This is a post-print version for a chapter published in G. Çağdaş et al. (Eds.): CAAD Futures 2017, CCIS 724, pp. 151–169, 2017, © Springer Nature Singapore Pte Ltd. 2017. DOI: 10.1007/978-981-10-5197-5_9. The final publication is available at : https://link.springer.com/chapter/10.1007/978-981-10-5197-5_9.

SILVEREYE– the implementation of Particle Swarm Optimization algorithm in a design optimization tool.

Judyta M.Cichocka[1*[0000-0002-9798-5859]], Agata Migalska[2], Will N.Browne[3], Edgar Rodriguez[4]

[1] Wroclaw University of Technology, Faculty of Architecture, Wroclaw, Poland
judyta.cichocka@gmail.com
[2] Wroclaw University of Technology, Faculty of Electronics, Wroclaw, Poland
agata.migalska@pwr.edu.pl
[3] Victoria University of Wellington, School of Engineering and Computer Science, Wellington, New Zealand
will.browne@ecs.vuw.ac.nz
[4] Victoria University of Wellington, School of Architecture and Design, Wellington, New Zealand
edgar.rodriguez@vuw.ac.nz

Abstract. Engineers and architects are now turning to use computational aids in order to analyze and solve complex design problems. Most of these problems can be handled by techniques that exploit Evolutionary Computation (EC). However existing EC techniques are slow [8] and hard to understand, thus disengaging the user. Swarm Intelligence (SI) relies on social interaction, of which humans have a natural understanding, as opposed to the more abstract concept of evolutionary change. The main aim of this research is to introduce a new solver Silvereye, which implements Particle Swarm Optimization (PSO) in the Grasshopper framework, as the algorithm is hypothesized to be fast and intuitive. The second objective is to test if SI is able to solve complex design problems faster than EC-based solvers. Experimental results on a complex, single-objective high-dimensional benchmark problem of roof geometry optimization provide statistically significant evidence of computational inexpensiveness of the introduced tool.

Keywords: Architectural Design Optimization (ADO), Particle Swarm Optimization (PSO), Swarm Intelligence (SI), Evolutionary Computation (EC), Structural Optimization

1 Introduction

Design processes based on parametrically variable elements allow designers to explore thousands of potential solutions and can be enriched by computational optimization [1-4]. Thus, the subfield of engineering that uses optimization methods to aid designers in

studying and solving design problems plays a vital role in modern practice. Architectural engineering and design problems are usually classified as difficult problems, since they correspond to five reasons why problems might be difficult according to Michalewicz and Fogel [2]:

1. An exhaustive search for the best answer is forbidden due to the large number of possible solutions in the search space.
2. The optimization problem is complex and any simplification of it would lead to discovering useless solutions.
3. In many cases a series of solutions rather than a single one is required due to the fitness function's noisiness or variety in time.
4. Due to numerous constraints on the search space the construction of even one feasible solution is difficult and discovering an optimal result even more so.
5. The person solving the problem is not satisfactorily prepared or faces some mental barriers that prevents him/her from discovering a solution.

The complexity of design problems often results in non-convex fitness landscapes. Due to this non-convexity conventional approaches (e.g. gradient-based optimization) are commonly regarded as inappropriate for design problems [3]. Moreover, in complex problems, often instead of a single solution, a number of different solutions could be optimal. Therefore, although deterministic approaches can address this problem, they cannot provide the set of different feasible solutions as they would yield always the same result. In general, gradient methods are ill-defined for non-differentiable functions and may not be suited to non-linear, multi-modal or discontinuous objective functions that often define engineering optimization problems [4]. Therefore the algorithms from the fields of Computational Intelligence, Biologically Inspired Computing, and Metaheuristics have been applied to address such functions [5]. Moreover, often the high complexity of optimization problems might be simply due to errors in the optimization problem's statement that is common in the design space. An optimization procedure (problem formulation/fitness function) constructed by architects and designers need be considered as black-box as it is in metaheuristic methods [5]. Metaheuristics such as techniques rooted in Evolutionary Algorithms or Swarm Intelligence, can be applied to 'almost any kind of the optimization problem, regardless of the type (e.g. continuous/discrete, linear/nonlinear, convex/nonconvex) and number of variables' [3,5], , as opposed to more conventional approaches that require specific objective functions.

Thanks the universal use of metaheuristics, there has been noteworthy growth in the field of nature-inspired optimization for the last few decades. Natural processes like Darwinian evolution, foraging strategies or social group behavior are widely applied in difficult real world problems ranging from industry to commerce [6]. There are two main families of the algorithms that constitute this field today: Swarm Intelligence (SI) algorithms and Evolutionary Computation (EC) Algorithms. In this work we will refer to the following system of classification presented by Brownlee [5]:

"*Swarm Algorithms* (SI Techniques)*: Particle Swarm Optimization, Ant System, Ant Colony Optimization, Bees Algorithm and the Bacterial Foraging Optimization Algo-*

rithm. Evolutionary Algorithms (EC Techniques): *Genetic Algorithm, Genetic Programming, Evolution Strategies, Differential Evolution, Evolutionary Programming, Grammatical Evolution, Gene Expression Programming, Learning Classifier System, NSGA and SPEA*".

Currently, mostly the EC-based algorithms are exploited in the optimization tools for designers and architects. Therefore, it is unknown how the SI techniques can perform in solving design problems. Although, evolutionary solvers can address most of the design problems [3,7], they are very slow [3,8]. Thermal, solar, structural or acoustic analysis may take a minute/several minutes per iteration [9], thus 'a single process may run for days or even weeks' [8] and 'even with a powerful hardware set-up, the repetition of hundreds or even thousands of analyses might transform the optimization process into an extremely long task' [10]. In many cases the high computational cost of each optimizer run prevents designers from using optimization techniques [3], thus many revolutionary design solutions are not discovered. In this study, the recently developed technique PSO is hypothesized to be able to solve complex design optimization problems faster than existing EA-based tools. The PSO-optimizer prototype was developed on multiple 'toy' problems [11] and this paper develops a fully-working tool - SILVEREYE and tests its use for a real-world application.

The main aim is to introduce a new optimization solver called Silvereye that implements a bespoke PSO algorithm in the Grasshopper framework. PSO is selected as demonstrator of SI optimization processes in design tools, as it constitutes the foundation for the other SI algorithms [5]. The second objective is to evaluate its computational speed and accuracy in solving design problems based on a comparative complex benchmark real-world problem. These initial benchmark tests were conducted on minimizing displacement of shell roof structure inspired by the Crematorium of Kakamigahara – Meiso no mori – Toyo Ito. Experiments have confirmed that Silvereye is faster in single-objective high-dimensional optimization problem as it outperforms existing EA-based alternatives (Galapagos, Octopus) in the search time criteria. Therefore, SI techniques are considered to be competitive to existing EA-based optimizers in solving complex design problems.

At the time when 'the importance of effective problem solving has never been greater' [2] facilitation of an extremely compact algorithm is anticipated to speed up the optimization process and improve the human-machine interaction. This could develop widespread adoption of solvers, popularize them among practitioners and thus enable discoveries of high performing solutions in design space.

2 Background

2.1 Solvers in design software – state of art

Nature-inspired intelligent computation has become recently an essential part of the interdisciplinary design process. Solvers based on EA obtainable in design systems reflect the principles of the evolution found in nature as they have an implicit parallel interactive search for solutions. The most commonly used of the current design platforms that enable visual programming with the embedded optimization solvers (if applicable) are presented in Table 1. The implemented optimization algorithms confirm that the design optimization field is dominated by EC techniques. Currently, in the most popular application enabling visual programming, Grasshopper for Rhino3D modelling environment [1,12,20] there are 4 optimization solvers available: Galapagos, Goat, Opossum and Octopus. The second most popular visual programming design application is Dynamo for Autodesk's Revit Architecture. Optimo – a plug-in for Dynamo [12] is based on the Non-dominated Sorting Genetic Algorithm II (NSGA – II). The other parametric platforms do not have generic solvers developed.

Table 1. Selection of design application with facilitated visual programming techniques.

PRODUCER - SOFTWARE	VISUAL PROGRAMMING/ PARAMETRIC PLUG-IN	% USERS [12]	OPTIMIZATION PLUG -IN /OPTIMIZER	OPTIMIZATION ALGORITHMS	CATEGORY	AUTHOR AND DATE OF IMPLEMENTATION
Gehry Technologies: Digital Project	n/a	3%	n/a	n/a	n/a	n/a
Bentley Systems: Mictostation	Generative Components	0%	Prototype Design Evolution	GA	EC	Bentley Applied Research, 2011
Autodesk: Revit Architecture	Dynamo (2011)	7%	Optimo (2014)	MOEAD	EC	Mohammad Rahmani Asl and Dr. Wei Yan, 2014
McNeel & Associates : Rhinoceros3D	Grasshopper (2008)	90%	Galapagos	GA	EC	David Rutten, 2008
				SA	other[1]	David Rutten, 2011
			Goat	COBYLA,BOBYQA,Sbplx, DIRECT,CRS2	other[2]	Simon Flory, 2010-2015
			Opossum	Surrogate models	Modelbased	Thomas Wortmann, 2016
			Octopus	SPEA- 2 and HypE	EC	Robert Vierlinger, 2013
Nemetschek North America: Vectorworks	Marionette (2015)	<1%	n/a	n/a	n/a	n/a

2.2 The PSO algorithm

PSO is a sub-field of Computational Intelligence. It belongs to the field of Swarm Intelligence and Collective Intelligence [5]. PSO has both ties with Artificial Life and Evolutionary Computation and conceptually 'seems to lie somewhere between genetic

[1] Simulated Annealing belongs to the class of Physical Algorithms
[2] Goat interfaces NLopt, a collection of mathematical optimization libraries

algorithms and evolutionary programming' [13]. In terms of computational expensiveness the PSO algorithm lies 'between evolutionary search, which requires eons, and neutral processing, which occurs on the order of milliseconds' [13]. PSO is a type of biological system, where the collective behaviors of individuals interacting with each other and their environment form the optimization process. PSO was inspired by bird flocking and fish schooling [13] and its simulation was motivated by the need to model 'human social behaviour' [13]. The interaction of agents within the PSO method is familiar to humans, since humans share with them some behavioral patterns (e.g. avoiding physical collision) [13].

3 Method

3.1 Method selection

As shown in the study conducted by Zhang, Wang and Ji [14] 'the number of publications per year related to PSO is the highest among all seven SI-based algorithms. This suggests PSO is the most prevalent SI-based optimization algorithms'. None of SI algorithms has been implemented in any design software application so far, thus PSO is recommended as it is a baseline algorithm for many variations of SI algorithms [5]. Particle Swarm Optimization is the apt representation of the SI family of algorithms as it corresponds to the basic five principles of swarm intelligence [13-15]:

1. Proximity principle – the group should be able to carry out '*elementary space and time computations*'[15].
2. Quality principle - besides the time and space considerations, the population should be able to act in response to the quality factors.
3. Principle of diverse response – the population should seek to distribute its activities along many modes.
4. Principle of stability - the group should not change its behavioral mode upon every environmental fluctuation, unless it is worth the computational price.
5. Principle of adaptability - the population should alternate its behavior, when it is worth the computational price.

It is worth noticing that above rules resemble numerous '*economic decision making principles or folk maxims*' [15] like time is money, only buy the best, don't put all your eggs in one basket, better safe than sorry, a bird in the hand is worth two in the bush, invest for the future, etc. [16] that may indicate the search rules are similar to human common sense. PSO is extremely simple and it has already demonstrated its computational inexpensiveness in feature selection [17] and robustness in the areas of the biological, chemical, medicine, mechanical and civil engineering, fuel and energy, automatic control, and others [14]. Moreover, the PSO algorithm has been already shown to be a promising method for solving complex design problems [11]. Therefore the initial version of PSO presented by Kennedy and Eberhart in 1995 with introduced internal weight factor [18] is selected for the first implementation of SI- based algorithm in a design optimizer.

3.2 Method description

The unique concept of PSO in comparison to EC techniques is that potential solutions 'fly' through the hyperspace, when in evolutionary computation potential solutions are represent as locations in it [13]. In PSO every single solution is a particle ('bird') in the multi-dimensional hyper-volume of the solution search space. A velocity component enables the system to progress quickly in the direction of good solutions. In practice velocity defines how much each parameter (dimension) can be adjusted in the forth-coming step. The goal of the PSO algorithm is to direct all the particles to the optima in the hyperspace of parameters [5]. The process is initialized with a random population of particles, each representing a solution. In every time step the fitness value is sampled and the velocity of each particle is updated. The velocity is constructed based on the current velocity, the best known global position (the best result sampled by the whole population - gbest) and the best position discovered by the particle (pbest) [5]:

$$pbest(i,t) = \underset{k=1,\dots,t}{\arg\,min}\ \left[f\left(P_i(k)\right)\right],\ i \in \{1,2,\dots,N_p\},$$

$$gbest = \underset{\substack{i=1,\dots,N_p \\ k=1,\dots,t}}{\arg\,min}\ \left[f\left(P_i(k)\right)\right] \tag{1}$$

When these two values are known, the position (P) and velocity (V) of each particle is updated using the following equations:

$$V_i(t+1) = \omega V_i(t) + c_1 r_1\left(pbest(i,t) - P_i(t)\right) + c_2 r_2\left(gbest(t) - P_i(t)\right) \tag{2}$$

$$P_i(t+1) = P_i(t) + V_i(t+1) \tag{3}$$

Where,

V denotes the velocity for particle i, ω is the momentum coefficient for the whole swarm, $r1, r2$ are uniformly distributed random variables within range [0, 1], $c1,2$ are the positive constant parameters called '*acceleration coefficients*' or '*learning factors*'. Table 2 presents the commonly set internal tuning parameters of the PSO algorithm. These values have been determinate based on much empirical testing to ensure robust-ness across a wide range of problem domains [5,13,19] (Fig. 1).

Table 2. PSO – commonly set internal parameters of the PSO algorithm.

Internal parameters of the PSO algorithm:	
1. number of particles	low, around 20-40
2. the maximum speed a particle can move (max. change in its position per iteration) (*Vmax*.)	should be bounded as to the percentage of the size of the domain
3. learning factors (biases towards global and personal best positions) (*c1*), (*c2*)	between 0 and 4, typically 2; 2 in the PSO from 1995

Additional factors that can be introduced:	
4. a local bias factor (local neighbourhood)	only when neighbours are determined based on the Euclidean distance between particles
5. an inertia or momentum coefficient (ω)	0.3 [19]

The pseudo code of the implemented procedure is as follows:

Step 1. Initialization

For each particle i $= 1, \ldots, N_p$, do

(a) Initialize the particle's position with a uniformly distribution as $P_i(0) \sim U(LB, UP)$, where LB and UB represent the lower and upper bounds of the search space

(b) Initialize $pbest$ to its initial position $pbest(i, 0) = P_i(0)$.

(c) Initialize $gbest$ to the minimal value of the swarm: $gbest(0) = \arg\min f[P_i(0)]$.

(d) Initialize velocity: $V_i \sim U(-|UB - LB|, |UB - LB|)$.

Step 2. Repeat until a termination criterion is met

For each particle i $= 1, \ldots, N_p$, do

(a) Pick random numbers: $r_1 r_2 \sim U(0,1)$.

(b) Update particle's velocity. See formula (2).

(c) Update particle's position. See formula (3).

(d) If $f[P_i(t)] < f[pbest(i,t)]$, do

(i) Update the best known position of particle i: $pbest(i,t) = P_i(t)$.

(ii) If $f[P_i(t)] < f[gbest(t)]$, update the swarm's best known position: $gbest(t) = P_i(t)$.

(e) $t \leftarrow (t + 1)$;

Step 3. Output $gbest(t)$ that holds the best found solution

Fig. 1. Pseudocode of a standard PSO. Source: [14].

The implemented algorithm includes inertia weight (ω) introduced by Shi and Eberhart in 1998. ω is also called the momentum coefficient and is used to 'balance the global exploration and local exploitation' [14]. It was demonstrated by the authors, that introduction of ω improves the PSO performance [19]. In the implemented PSO algorithm stochastic factors $c1$, $c2$ are equal to two, as was suggested by Kennedy and Eberhart [13]. Computational expensiveness depends on the size of the population, compliance of the communication rules between different agents [17] and directness of behavioral response to environmental stimuli [15].The default swarm size is set to 20, as this number is the least expensive recommended swarm size [5]. It is common to set an upper bound for the velocity component. The maximum value of the velocity (Vmax) should be calculated based on the size of the domain, so it is individual to each design problem.

A comprehensive survey of studies deliberating on the adjusting the internal tuning parameters was done by Carlisle and Dozier [20]. Discussion of setting optimized values for the design problem domains in the Grasshopper framework is an issue of further research. Currently, the identified values are robust and proper across a wide range of problems, which simplifies the adaptation of the PSO technique into design practice such that it is straightforward.

4 Silvereye

The implementation of PSO in the framework of the most popular parametric modeling platform – Grasshopper [1,12,21] will enable direct comparison of the new tool based - Silvereye with existing EC –based solvers (Galapagos, Octopus). We implement our Particle Swarm Optimization solver in C#, which is one of the languages available for custom component scripting in Grasshopper as a Silvereye add-on component (Fig 2). The central part of the component is a ParticleSwarmOptimization.dll file, a shared library containing an implementation of the core version of the Particle Swarm Optimization algorithm. A Graphical User Interface (GUI) is implemented separately and shipped in a form of a Grasshopper Assembly file, Silvereye.gh. In our current implementation the input parameters, the fitness input and the sliders correspond to the input arguments of the objective function and internal tuning parameters. The output is a list comprising of the best solutions found at each iteration of the algorithm. In order to solve an optimization problem, the Silvereye component has to be placed on the Grasshopper canvas and connected to a number of input parameters, either to number sliders or to gene pool components. These N input parameters constitute an N-dimensional search space S, $\boldsymbol{S \subseteq R^N}$, of the optimization problem in question. The component is also connected to a number component from which the value of a fitness function is retrieved. As in the standard form of an optimization problem definition the optimization objective is to minimize a fitness function, however a user can change the objective to maximization. In a default scenario, once an optimal solution search is initiated a swarm of 20 particles is created with each particle located at a uniform random position in S. The swarm searches for a solution to a minimization problem in 60 iterations. The maximal velocity of each particle is the same in all N directions and is chosen as 0.2 of the maximal range of the input parameters (See: 8.1 Conclusions).

The Silvereye GUI presents a user with a form where the default values of the optimization settings are displayed. The optimization settings comprise of an optimization objective, a swarm size, the number of iterations and a maximal velocity. An execution of the solver is then terminated either after the time has elapsed or after the number of iterations is reached, whichever occurs first. Finally, a user can choose to use an initial position. If this option is selected, one of the particles takes on the current values of the input parameters.

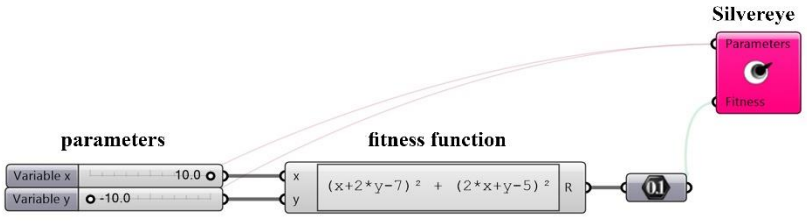

Fig. 2. Silvereye component on the GH canvas.

5 Initial tests / benchmarks - results

5.1 Problem formulation

In this section, the functionality of Silvereye is presented based on a structural optimization of a free-form roof structure benchmark problem inspired by a real architectural project (Fig. 3). As a case study we selected the Meiso no Mori Crematorium (from the Japanese words meaning Forest of Meditation), designed by Toyo Ito and Mutsuro Sasaki consultants at Kakamigahara City, in Gifu Japan. This building is located in the park-like cemetery between wooded hills and a small pond and it replaced the previous demolished crematorium [22]. Toyo Ito says, that his design is '*not as a conventional massive crematorium but as architecture of a spacious roof floating above the site like slowly drifting clouds, creating a soft field*' [23]. The 20 cm- thick roof of white reinforced concrete is made up of concave and convex forms and rises up to 11.5m in parts. The roof is supported by twelve freely scattered cone columns dropping from the ceiling [24] and four structural cores [22]. Although the roof seems to be free in its form, it was designed through rigorous structural analysis. In the original project the optimal solution was discovered by modifying the initial shape defined by the architect through the optimization process on minimizing the total strain energy and deformation based on the Sensitivity Analysis method [25-26].

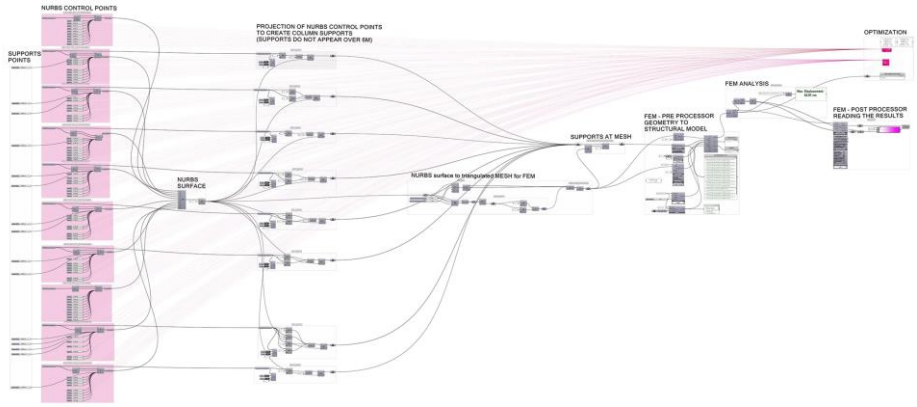

Fig. 3. Grasshopper definition of the optimization problem (adopted from the initial Alberto Pugnale's GH definition [28].

Contrary to the form improvement in the original procedure, in the problem studied by Pugnale and Sassone the optimization process was turned into a form exploration [10]. This case study was elaborated by Pugnale and Sassone in 2007 [27] and reanalysed in 2013 by Pugnale [28]. In 2007 the geometrical information and the optimization process with '*a specially developed Genetic Algorithm*' [27] were written in the programming environment embedded in Rhinoceros – RhinoScript [28]. In 2013 Pugnale reinterpreted the problem to the Grasshopper environment and used Karamba [29] for FEM

(Finite Element Method) analysis and Galapagos' Genetic Solver for the optimization procedure. In his exercise the roof geometry of the original Toyo Ito' s project was exemplified in rather a free manner [28], '*acting as design tool of investigation/exploration'* [27]. In the study we adopted the problem presented by Pugnale [28] in order to evaluate the performance of the Silvereye solver in comparison to the existing tools (Galapagos, Octopus). The crucial parameters for the structural efficiency evaluation are maximum nodal displacement (m), the mass (maximum total force of gravity) and strain energy. Minimizing all of them leads to more efficient structures [30]. Displacements are a vector field and in contrary to the strain energy it can reveal both local and global weaknesses [10]. In the presented problem, we chose to minimize total maximum nodal displacement because it is a factor minimizing effectively the strain energy in the whole model [30], and a parameter that can indicate the weakest points of the structure, as maximum displacement returns the values of the local peaks of the analyzed mesh. The aim of the optimization search is to find the shape of the roof within the boundary conditions (Fig. 4) (Table 3) that will have the smallest possible maximal displacement. In the experiment we are preserving the dimensions of the real free-curved surface: length - 80m, width - 60m and thickness - 200mm [26]. The roof structure is described by the means of the third degree NURBS surface (i.e. a rational polynomial continuous function, defined by a set of control points) with a grid of 10x10 control points in a plan projection restricted to the boundary presented in Fig. 4.

Fig. 3. Boundary conditions in plan and elevation. Source: [31].

The z-coordinates of the control points of the NURBS surface defining the roof geometry have been set as the parameters of the problem to be optimized. Their vertical domain is limited to the range -10.000 and 10.000m with 20 000 degrees of freedom (as the number slider component's accuracy was set to 3 decimal places). For the FEM analysis the continuous geometry is translated to the triangular discrete finite element mesh – quadratic grid of size 35x20 with complementary diagonals. The dead-load and mesh load 6kN/m² were applied to the 20cm thick concrete shell (C25/30 defined by Eurocode3) as it is considered as sufficient approximation of real load situation for the initial design stage. The height and position of 15 pillars, the planar projection of the roof boundary and thickness of the shell are the boundary conditions for the elaborated problem. The formulations of the design problem in our tests and in the previous studies are described in detail in Table 3.

The problem is interesting both from an architectural and a scientific point of view. The size of a set of solutions can be calculated as: X^N, where X is number of possible states, and N is a number of design variables (parameters), therefore in this case it is equal to 20 000^85 possible solutions. The calculation of the entire set of solutions, under the assumption that the calculation of one solution takes one microsecond, would take about 1.23 * 10352years! Simplification of the problem would lead to discovery of useless solutions. Thus this problem is constrained with both hard-constraints, such as x,y coordinates of the position of the columns or the roof outline, and with a penalty method included in the fitness function that prevents the tapered pillars becoming too high. Moreover, a number of solutions could be optimal and discovering even one feasible solution might be difficult. Therefore, this problem is considered a difficult optimization problem.

Table 3. Problem formulation and design tools used in the previous and current analysis.

	Original Project 2004-2006 [26]	**Study 2007** [27]	**Study 2013** [28]	**Study 2017**
design tools				
Geometry modeler	CAD/CAM	Rhinoscript in Rhinoceros	Grasshopper	Grasshopper build 0.9.0076
FE solver	NASTRAN	Ansys	Karamba	Karamba ver. 1.1.0
Optimization tool/ algorithm	Iterative gradient-based optimization technique -shape design method used sensitivity analysis	GA scripted in RhinoScript	Galapagos (GA)	Silvereye (PSO) build 0.0.1
problem formulation				
Shell Geometry	"Triangulated shell-elements following the one-meter grid compose', 3600 elements 20cm--thick	Mesh – triangular grid 50x50 2500 elements 15cm-thick	Mesh – triangular grid 35×20 700 elements 25cm--thick	Mesh – triangular grid 35×20 700 elements 20cm--thick
Solution domain	Small, local search from predefined by architect shape	"1/5 of the edge length of the roof plane projection" [31]	Sliders in range: -10m and 10m	Sliders in range: -10m and 10m

[3]E = 3100[kN/cm²], G = 1291.67 [kN/cm²], gamma =25 [kN/m³], alphaT = 1.0E-5[1/Cº], fy = 1.67 [kN/cm²]

Loads	Gravity load Dominant load – 6.0 kN/m²[26]	Load 1kN/m² [10]	Self-weight and live load (6kN/m²)	Self-weight and mesh load (6kN/m²)
Constraints	Columns as roller supports withstand only vertical load Walls and other columns withstand horizontal load	'*Horizontal coordinates of NURBS control points*' and '*vertical coordinates of control points that represent the pillar-slab joint*' [31]	x,y coordinates of the pillars are fixed; projection of the roof outline is fixed; penalty function as the restriction of the max. height of columns	x,y coordinates of the pillars are fixed; projection of the roof outline is fixed; penalty function as the restriction of the max. height of columns
Variables	z-coordinates of the NURBS surface	z-coordinates of the NURBS surface	z-coordinates of the NURBS surface: 85 parameters	z-coordinates of the NURBS surface: 85 parameters
Fitness function	Strain energy	Vertical displacement	Total displacement	Total Displacement

5.2 Hardware and software

Tests were conducted with the use of a personal computer, which is typically available for modern architectural and design practices: Dell XPS 12 (Processor: Inter® Core (TM) i5-4200U CPU @, 2.30GHz, installed memory: 4.0 GB, Graphics: Intel (R) HD Graphics 4000).

6 Results

The introduced tool, Silvereye, and the existing EC-based alternatives (Galapagos and Octopus) were tested on the presented complex design problem (See section: 5.1 Problem formulation) in two runtime categories. Firstly, tests with a 15-minute runtime limit to investigate the software applications' abilities to discover different high quality solutions when a user only has short runtime available. The 8-hour tests were conducted only with the best performing in short runs set of settings for each solver in order to examine solvers' capability of finding the best possible solution if the longer time was available for testing. Silvereye was tested for sensitivity to the velocity parameter, with different maximum velocities ranging from 0.5 to 20, as it needs to be robust for ease of use. Twenty particles were initialized in each test. Galapagos was tested on two sets of settings: Set 1-settings used by Pugnale in his original definition of the problem [28]: population = 50, initial boost = x5, maintain = 3%, inbreeding = +60% and with default settings: Set 2:population = 50, initial boost = x2, maintain = 5%, inbreeding = +75%. For test purposes, Octopus was tuned with default internal parameters: Elitism = 0.500, Mut. Probability = 0.100, Mutation Rate = 0.500, Crossover Rate = 0.800, Population size = 100. All results for 30 runs of each tested solver are presented in Table 4. In the 15-minutes tests Silvereye with implemented PSO on average discovered solutions that are about 2.5 times better than those found by Galapagos and about two times better in comparison to the results achieved by Octopus.

The maximum deflection (d) of any structure should be such that people using feel safe and comfortable. As a rough rule of thumb it should be limited to d <= L/300 [30], where d is the maximum deflection and L is a span between the supports. The minimum span between columns in the benchmark problem is 8.82m, consequently for initial

design stage the acceptable value for the maximum deflection, understood as the degree to which a roof structure is displaced under the loads, is about 3cm. In almost all tested runs Silvereye discovered solution under 3cm, when neither Galapagos nor Octopus managed to find a feasible solution in thirty runs of the 15-minute tests (results underlined in Table 4). The local nodal displacements exceeding 3cm are marked in the Fig. 5.

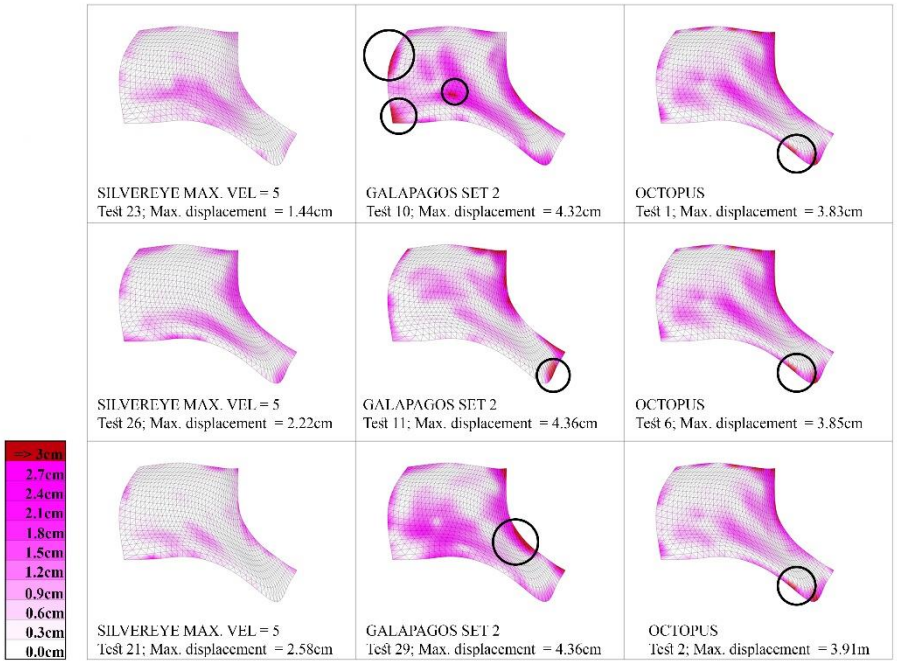

Fig. 4. Different solutions discovered during the optimization process during 15-minute tests.

Statistical test selection procedure included the check of the normality of the all distributions of the results with K-S., Lilliefors and Shapiro-Wilk tests and the Levene and Brown-Forysythe tests of Homogenity of Variances. The normality assumption ($p < 0,05$) is not satisfied in all tested cases. The tests of homogeneity of variances also decline the hypothesis zero. Therefore, for the comparison of the results the non-parametric corresponding tests are used.: Kruskal-Wallis ANOVA and Median Test. Both tests indicated on the statistically important differences. The multiple comparisons of p value indicate the statistical differences between the independent sets of results when p is below 0,05 (Table 5).

Table 4. Results achieved by Silvereye, Galapagos and Octopus: 15-minute tests on the benchmark problem.

RUNTIME LIMIT = 15 MINUTES									
	SILVEREYE						GALAPAGOS		OCTO-PUS
	result [cm]						result [cm]		result [cm]
Test	Vmax= 0.5	Vmax= 1	Vmax= 2	Vmax= 5	Vmax=10	Vmax=20	Set 1	Set 2	Set
1	2.93	1.73	1.47	1.92	2.83	2.74	4.73	5.53	3.83
2	2.59	1.64	1.72	2.13	2.52	1.72	6.27	4.03	3.91
3	2.94	1.98	1.50	1.90	1.97	2.77	5.38	4.40	3.85
4	1.77	1.48	1.53	1.47	1.85	2.33	5.66	4.59	3.85
5	2.08	1.89	1.62	1.86	2.73	2.67	5.93	4.65	3.83
6	2.51	1.93	2.57	1.79	1.78	1.86	4.04	5.06	3.85
7	2.48	1.44	2.30	1.66	2.00	1.90	5.25	4.75	3.83
8	1.68	1.99	1.91	2.36	2.15	2.29	4.79	4.91	4.43
9	2.31	2.13	1.80	1.15	1.82	2.36	5.48	4.43	4.43
10	2.13	1.27	1.96	2.07	1.63	1.48	5.51	4.32	4.43
11	2.48	2.09	1.51	1.63	2.40	2.31	5.85	4.36	4.33
12	2.43	1.97	2.06	1.85	1.73	2.32	5.85	4.91	4.33
13	2.99	2.52	1.25	1.51	2.23	2.82	5.49	5.23	4.33
14	2.74	1.72	1.59	1.51	1.73	1.69	5.43	4.87	4.33
15	3.48	2.69	1.58	1.52	2.22	2.02	6.07	4.80	4.33
16	2.55	1.74	2.60	1.53	2.02	2.83	5.91	3.55	4.33
17	2.76	1.54	1.24	1.76	3.24	2.68	6.68	5.42	4.33
18	1.73	1.63	1.71	1.98	2.29	2.73	3.22	5.18	4.33
19	5.01	1.47	1.48	1.33	1.86	1.88	4.78	4.31	4.33
20	2.33	2.20	1.98	1.69	1.72	2.45	6.50	4.90	4.43
21	3.40	2.34	2.66	2.58	1.49	2.04	6.29	4.73	4.33
22	2.42	1.84	1.62	1.82	2.95	2.87	5.16	3.76	4.33
23	2.43	1.63	1.52	1.44	1.98	1.61	5.05	4.56	4.33
24	2.66	2.01	1.78	1.78	1.74	1.98	4.50	6.47	4.33
25	3.19	2.80	1.76	2.02	2.53	1.92	6.21	4.56	4.33
26	3.58	1.86	1.65	2.22	2.38	2.78	5.35	4.10	4.33
27	3.26	2.92	1.99	1.54	1.89	2.31	4.85	5.04	4.26
28	2.95	1.27	2.28	1.64	2.15	2.39	4.74	5.23	4.33
29	3.38	2.16	1.81	1.57	1.61	3.76	4.15	4.36	4.33
30	2.42	4.52	1.82	2.35	2.11	2.21	5.73	5.18	4.81
Av.	**2.72**	**2.01**	**1.81**	**1.79**	**2.12**	**2.32**	**5.36**	**4.74**	**4.25**

Table 5. Multiple comparisons p values between all sets of the results of 15-minute tests on benchmark problem.

Depend.:	**Multiple Comparisons p values (2-tailed); Independent (grouping) variable: SetName Kruskal-Wallis test: H (8, N= 270) =205,4970 p =0,000**								
Set	**Silvereye 05**	**Silvereye 1**	**Silvereye 2**	**Silvereye 5**	**Silvereye 10**	**Silvereye 20**	**Galapagos 1**	**Galapagos 2**	**Octopus**
	R:141,97	R:79,000	R:59,000	R:58,267	R:94,700	R:113,83	R:244,93	R:226,07	R:201,73
Silvereye Vmax =0.5		0,064433	**0,001394**	**0,00119**	0,686148	1	**0,000012**	**0,001091**	0,109198
Silvereye Vmax = 1	0,064433		1	1	1	1	**0**	**0**	**0**
Silvereye Vmax = 2	**0,001394**	1		1	1	0,235258	**0**	**0**	**0**
Silvereye Vmax = 5	**0,00119**	1	1		1	0,210627	**0**	**0**	**0**
Silvereye Vmax = 10	0,686148	1	1	1		1	**0**	**0**	**0,000004**
SilvereyeVmax = 20	1	1	0,235258	0,210627	1		**0**	**0,000001**	**0,000469**
Galapagos Set 1	**0,000012**	**0**	**0**	**0**	**0**	**0**		1	1
Galapagos Set 2	**0,001091**	**0**	**0**	**0**	**0**	**0,000001**	1		1
Octopus	0,109198	**0**	**0**	**0**	**0,000004**	**0,000469**	1	1	

Non-parametric tests indicated that Silvereye with the Vmax between 1 and 20 performed statistically better than Galapagos with Set 1 and Set 2 and Octopus with default settings. They have also proven than PSO can perform statistically better or worse depending on tuning the Vmax internal parameter (Silvereye with Vmax = 0.5 performed statistically worse than Silvereye with Vmax = 2 and Vmax = 5)) (Figs. 6 and 7).

Results of the 15-minute tests revealed that Silvereye with the maximum velocity restricted to be between 1 and 10 discovers the best results. The maximum velocity may be understood as the '*maximum jump*' between values on the number sliders (design parameters included in the optimization process) in every iteration. It was observed that the most efficient maximum velocities should be in the range 1/20 and 10/20 of the variability of the design optimization parameters.

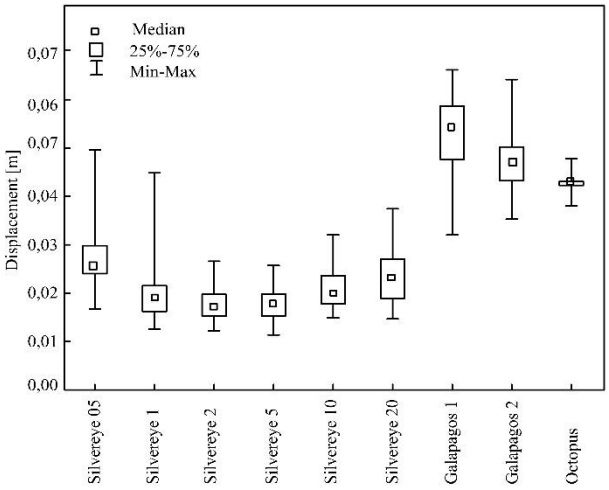

Fig. 5. Boxplot of the results of 15-minute tests on benchmark problem.

Table 6. Results achieved by Silvereye, Galapagos and Octopus - 8-hour tests on the benchmark problem.

RUNTIME LIMIT = 8 HOURS									
SILVEREYE							GALAPAGOS		OCTOPUS
result [cm]							result [cm]		result [cm]
Test	V_{max}= 0.5	Vmax= 1	Vmax= 2	Vmax= 5	Vmax= 10	Vmax= 20	Set 1	Set 2	Set
1	-	0.6	0.7	0.9	1.1	0.9	-	2.8	1.6

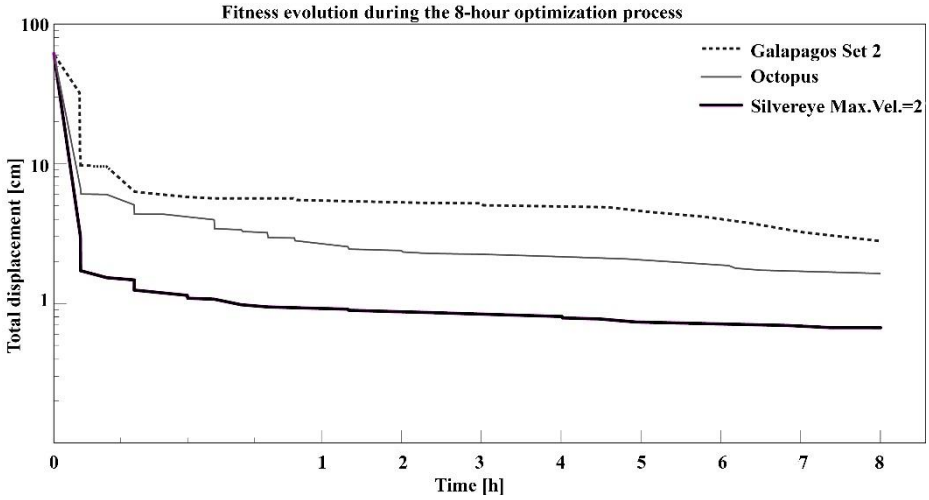

Fig. 6. Graph of the structural performance improvement during 8-hour tests. Source: [32].

7 DISCUSSION

Enhanced computational modelling is concluded as one of the seminal achievements of the 20thcentury [1]. Vast majority of design problems are of an engineering nature and modelling social behavior is an effective way to solve these optimization problems [13]. Facilitation of an extremely focused and fast optimization tool in the popular design platform may allow the most efficient solutions in the space of design and architecture to be discovered routinely, thus enabling a new quality in design practice. The introduced PSO-based solver has demonstrated that it has a potential to become this revolutionary tool. Silvereye demonstrated the significant prevalence over EA-based alternatives in the tested real-world benchmark problem. However, it has to be underlined, that every run of the test was not conducted in the identical conditions. The solvers'

performance was also influenced by the current computer's performance, which was affected by other processes running in the background and current temperature of the processor that was rising during the series of tests. During each run the optimization process was also slowing down due to the file storage.

However, throughout all experimentations, in a single 15-minute test Galapagos managed to produce 20-27 generations, whereas Octopus only between 8 and 11. It has to remarked upon that Octopus implements the SPEA-2 algorithm that belongs to the family of Multi-Objective Evolutionary Algorithms and is mostly dedicated to solve multi-criteria problems. Silvereye within the same time frame searched through 60-76 iterations. Despite differences in number of iterations or generations, each application within this time sampled the fitness landscape about the same amount of times ranging from 1200 to1600, which allows us to compare the results.

Although the introduced PSO-based solver outperformed the tested alternatives, there is no silver bullet for solving all design optimization problems and it should be acknowledged that '*A prerequisite to handling the real world as a multiplayer game is the ability to manipulate an arsenal of problem – solving techniques, namely, algorithms that have been developed for variety of conditions*' [2].

8 CONCLUSION AND FUTURE WORK

8.1 Conclusions

We introduced a new tool Silvereye that is a first implementation of Particle Swarm Optimization – an algorithm from the field of Swarm Intelligence in the most commonly used parametric design system. Unlike EA-based solvers Silvereye is intuitive as it is based on social behaviour rather than the concept of evolution. The PSO-based solver is shown to be applicable to complex, real-world design problems. The presented results are the evidence for its computational inexpensiveness and robustness in solving a high-dimensional design problem of roof-shape optimization. Silvereye discovered high-quality solutions faster than the tested alternatives based on Genetic Algorithm (Galapagos) and Strength Pareto Evolutionary Algorithm 2 (Octopus).

The quality of the solutions discovered by Silvereye depends on the maximum velocity parameter. The results indicate that the best performance of solver is achieved with the maximum velocity limited to the value in range between the 1/20 and 10/20 of the variability of the optimization parameters. The wide range of effective maximum velocities makes the tool easier to tune for practical implementations.

In almost every 15-minute test Silvereye discovered a high-quality solution that met the structural criterion in contrast to the tested alternatives. Therefore, it shown to be not only an efficient optimization solver, but also its potential to become a solution-exploration tool, as it can produce more feasible solutions in short run times than alternative approaches. This feature is especially appreciated in design space, where the incomputable aspects like aesthetics has a significant influence on the final solution chosen. Thus by providing an extremely fast tool the decision-making process can be enriched by the possibility to compare different high-quality solutions.

8.2 Future work

PSO has fewer tuning parameters than existing evolutionary solvers and the underlying fundamental rules exemplifying Swarm Intelligence may be regarded as more familiar to humans than principles of neo-Darwinian theory of evolution. However, the Silver-eye optimizer, as with the other EA-based solvers embedded in Grasshopper requires an '*aware*' user as the formulation of the fitness function and solution space can significantly influence the time of search and quality of solutions discovered. Although there are numerical benchmark tests available in literature, to the best of our knowledge, very few publications can be found that address the issue of intuitive control over optimization processes as a factor significantly influencing the achieved results. Like numerous problem studies, in the presented study it is avoided to '*immerse deeper into the issues of problem formulation and convergence behaviour*' [1], so this is an issue of further research.

Future work should explore more the intuitiveness of the tool, as this is a significant factor influencing its overall performance and user-friendliness. Further research should improve the machine-tool interaction and increase the control over the optimization process. This could be achieved by definition of the rules for the solver tuning in the different design problem domains in the Grasshopper framework and by the extension of the basic set of the tuning parameters.

Acknowledgements. This work is partially supported by the "THELXINOE: Erasmus Euro-Oceanian Smart City Network". This is an Erasmus Mundus Action-2 Strand-2 (EMA2/S2) project funded by the European Union.

References

1. Vierlinger R. Multi Objective Design Interface. Technischen Universitat Wien, 2013.
2. Michalewicz Z, Fogel BD. How to Solve It: Modern Heuristics. Second, Re. Springer; 2004.
3. Wortmann T, Costa A, Nannicini G, Schroepfer T. Advantages of surrogate models for architectural design optimization. Artif Intell Eng Des Anal Manuf 2015;29:471–81. doi:10.1017/S0890060415000451.
4. Wetter M, Polak E. A convergent optimization method using pattern search algorithms with adaptive precision simulation. Build Serv Eng Res Technol 2004;25:327.
5. Brownlee J. Clever Algorithms. Nature-Inspired Programming Recipes. First. LuLu; 2011.
6. Marques a I, Garcia V, Sanchez JS. A literature review on the application of evolutionary computing to credit scoring. J Oper Res Soc 2013;64:1384–99. doi:10.1057/jors.2012.145.
7. Rutten D. Navigating Multi-Dimensional Landscapes in Foggy Weather as an Analogy for Generic Problem Solving. 16th Int Conf Geom Graph 2014.
8. Rutten D. Evolutionary Principles applied to Problem Solving. Adv Archit Geom 2010 n.d.
9. Nguyen AT, Reiter S. Passive designs and strategies for low-cost housing using simulation-based optimization and different thermal comfort criteria. J Build Perform Simul 2013;7:68–81. doi:10.1080/19401493.2013.770067.
10. Pugnale A, Echengucia T, Sassone M. Computational morphogenesis. Design of freeform surfaces. In: Adriaenssens S, Block P, Veenendaal D, Williams C, editors. Shell Struct. Archit. Form-finding Optim., Routledge; 2014, p. 250–61.

11. Cichocka JM, Browne WN, Rodriguez E. Evolutionary Optimization Processes As Design Tools :, Proceedings of 31th International PLEA Conference ARCHITECTURE IN (R)EVOLUTION, Bologna 9-11 September 2015; 2015.
12. Cichocka JM, Browne WN, Rodriguez E. Optimization in the architectural practice - An international survey. In: P. Janssen, P. Loh, A. Raonic MAS, editor. Flows Glitches, Proc. 22nd Int. Conf. Assoc. Comput. Archit. Des. Res. Asia 2017, Pap. 155., Hong Kong: The Association for Computer-Aided Architectural Design Research in Asia (CAADRIA); 2017.
13. Kennedy J, Eberhart R. Particle swarm optimization. Neural Networks, 1995 Proceedings, IEEE Int Conf 1995;4:1942–8 vol.4. doi:10.1109/ICNN.1995.488968.
14. Zhang Y, Wang S, Ji G. A comprehensive survey on particle swarm optimization algorithm and its applications. Math Probl Eng 2015;2015. doi:10.1155/2015/931256.
15. Millonas MM. Swarms, Phase Transitions, and Collective Intelligence. St Fe Inst Stud Sci Complexity-Proceedings Vol 1994:30. doi:citeulike-article-id:5290209.
16. Geanakoplos JD, Gray L. When Seeing Further Is Not Seeing Better n.d.
17. Xue B, Zhang M, Browne WN. Particle Swarm Optimization for Feature Selection in Classification: A Multi-Objective Approach. IEEE Trans Cybern 2013;43:1656–71.
18. Shi Y, Eberhart RC. A modified particle swarm optimizer. Congr. Evol. Comput., 1998, p. 69–73.
19. Eberhart RC, Shi Y. Comparison between Genetic Algorithms and Particle Swarm Optimization. Ep'98 1998:611–6. doi:10.1007/BFb0040812.
20. Carlisle A, Dozier G. An Off-The-Shelf PSO. Proc Part Swarm Optim Work 2001;1:1–6.
21. Martyn D. Rhino Grasshopper. AEC Mag 2009;42.
22. Webb M. Organic Embrance. Archit Rev 2007;222:74–7.
23. Toyo Ito & Associates. Meiso no Mori Crematorium Gifu , Japan Toyo Ito & Associates Meiso no Mori Crematorium Gifu , Japan Toyo Ito & Associates n.d.
24. Municipal Funeral Hall in Kakamigahara. Detail 2008;48:786–90.
25. Sasaki M. Flux Structure. Toto Publishers; 2005.
26. Sakamoto, T., Ferre A. From Control to Design: Parametric/algorithmic Architecture. ACTAR; 2008.
27. Pugnale A, Sassone M. Morphogenesis and structural optimization of shell structures with the aid of a genetic algorithm. J Int Assoc Shell Spat Struct 2007;48:161–6.
28. Pugnale A. Computational Morphogenesis-with Karamba, Galapagos– Test on the Crematorium of Kakamigahara – Meiso no mori – Toyo Ito 2013. https://albertopugnale.wordpress.com/2013/03/25/computational-morphogenesis-with-karambagalapagos-test-on-the-crematorium-of-kakamigahara-meiso-no-mori-toyo-ito/ (accessed November 30, 2015).
29. Preisinger C. Linking Structure and Parametric Geometry. Archit Des 2013;83:110–3. doi:10.1002/ad.1564.
30. Preisinger C. Karamba User Manual (version 1.1.0). 2015.
31. Pugnale A. Engineering Architecture. Advances of a technological practice. Ph.D. Thesis. Politecnico di Torino, 2009. doi:10.1017/CBO9781107415324.004.
32. Cichocka JM, Particle Swarm Optimization for Architectural Design. Silvereye 1.0, Code of Space, Vienna 2016