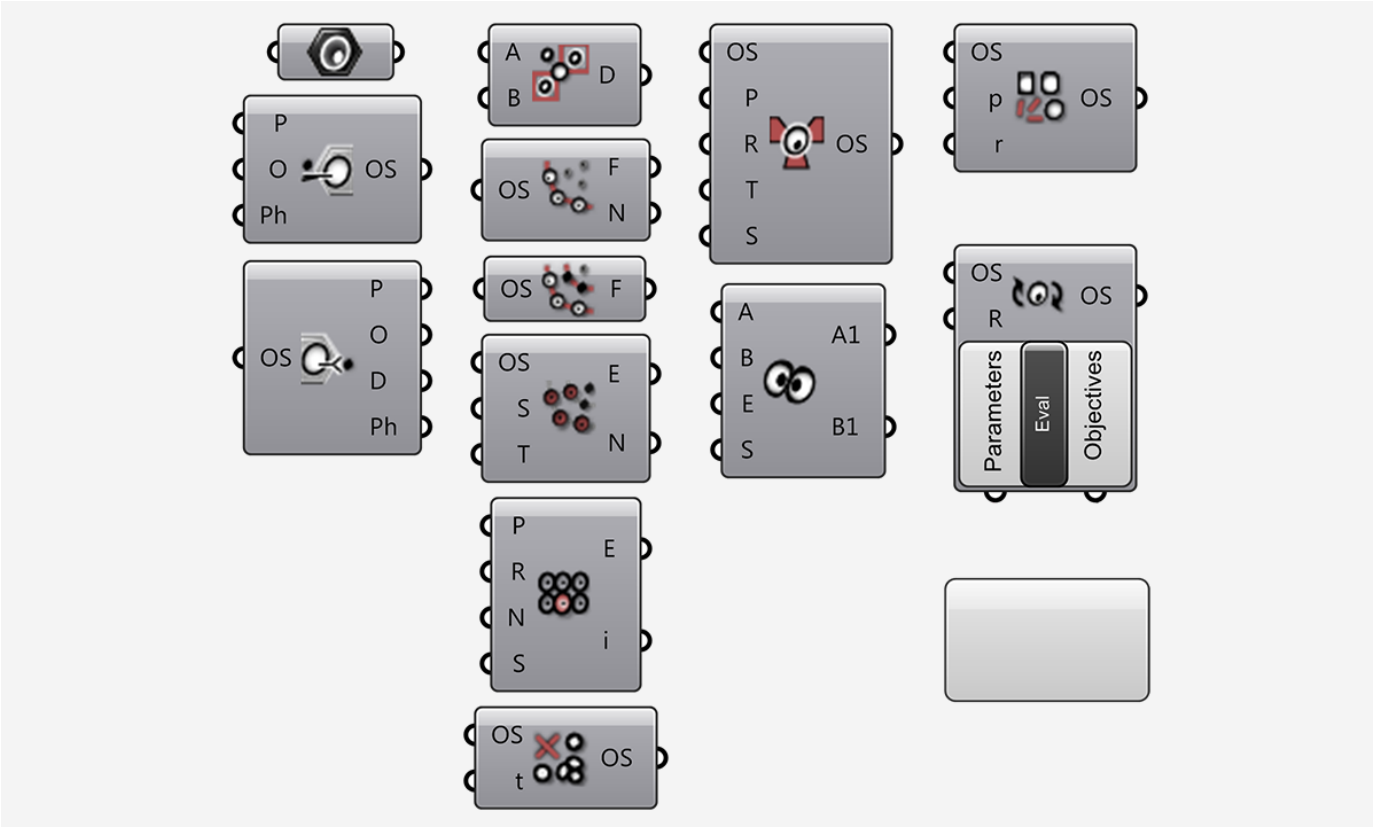
ACCOMMODATING CHANGE IN PARAMETRIC DESIGN

Robert Vierlinger
Klaus Bollinger
University of Applied Arts Vienna



1 Components for Custom Evolutionary Algorithms & Parallel Computation

ABSTRACT

Since the digital age, *how* we design has become as important as *what* we design. Never before have there been so many different techniques at our disposal, many of them with the ability to cross previously imagined frontiers. During the twentieth century, efforts to rethink the habitual practice of design are accompanying the process itself–the role of the model as an immediate source of physical and emotional feedback shifts from the analogue to the digital realm, and designing characterizes itself by manipulating digital systems rather than deterministic articulation. Today's computational resources advance the early pioneering approaches of digital emergence and complexity, while at the same time unveiling algorithmic design to non-programmers with intuitive interfaces. With respect to evolutionary search and optimization, this work investigates aspects of flexibility and performance on the levels of fast application and open representation. Firstly, a set of tools is introduced which allows parallelized implementation and customization of evolutionary algorithms on an existing parametric platform. Secondly, an approach for open-ended description of geometry is presented.

INTRODUCTION

Emotion and efficiency are just two desires being negotiated during an architectural design process, where ultimate success can mean the evolution of good trade-offs between goals at odds. Economics introduced the Pareto-principle to describe an optimal distribution of limited resources, where one aspect can just be improved when degrading another. Pareto-optimization always yields a set of solutions, ideally spanning diversely from one extreme situation to the other. The final decision, which can address non-quantifiable measures such as aesthetics, is left to the designer. Modern parametric modeling allows the quick iteration of many design alternatives, while at the same time being able to supply a number of measures regarding their performance-character. Explorations and design by trial and error are again becoming more interesting for building-practice, as intuitive design environments and innovative design approaches enable a broad base of users to lever vast computational resources. Modern design platforms and their open systems allow a simple formulation of intricate generative processes, though a model's ability to adapt to changes is limited by its own complexity. There is a desire to be designing with systems of simple rules, which are able to temporarily stabilize but stay open to changes of any kind, to be capable of reflecting any unforeseen state. Ideally, the accommodation of profound change at any point of the design process should be as easy possible.

Daniel Davis concludes that a lack of smart modeling can be a major setback for an architectural project (Davis 2013). There are a number of different approaches to foster the development of parametric models that are more universal. Most of them rely on the user to design his or her parametric formulation in a responsible way, considering the uncertainties or certainties a planning process could have. Though, apart from the aspects of workflow, flexibility can be needed on another scale. When the model primarily serves the purpose of exploration, a digital representation defines the boundaries of the design space. A simple system with a little number of input variables, but a large spectrum of eventually unforeseeable solutions, is sought to be able to effectively and systematically explore the design domain. Whereas it is difficult to innovate in the field of CAD software on the scale of this work, it is the usage and the way of parameterization within existing open systems that is of interest here. When it comes to the dynamic representation, combination, and generation of pattern, shape, emotion, and aesthetics, there is a lot of capacity to unveil.

Stanley (2007) proposes the digital evolution of Compositional Pattern Producing Networks (*CPPNs*) to create fields of any complexity and resolution. Successfully used for controlling artificial neural networks and formation of multidimensional shapes, a transformation of the principle to the parameterization of design is investigated here. *CPPNs* use a set of canonical functions such as sine, cosine, etc. and combine them in a dynamically evolving tree, being able to describe almost any state–similar to the Fourier-approximation of a mathematical function. This includes the iterative fitting of the CPPN, training it over generations to exhibit certain qualities. In contemporary practice, it is already the case that grids, patterns, and shapes are successively refined by the designer or a genetic algorithm to fulfill a specific purpose.

Purpose can be of a performance-related or aesthetic character, where we do not want to touch on a general quantification of beauty here. Shajay Booshan, lead developer at *ZHA CoDe*, uses techniques of image-synthesis and ray-tracing to quantify his aesthetic desires in a setting of simulated evolution of shapes (Bhooshan 2014). Bittermann (2009) is grasping soft parameters like perception and cognition of an architectural design by using fuzzy logic and artificial neural trees. However, he ultimately states, "in order to boost the design generation component to deal with greater uncertainty regarding the potential solutions, it is significant to enhance the exploration capabilities of the algorithm."

In case the goal is clearly defined and numerically manifest, algorithms have been well probed to find good solutions. Professional simulation systems have become freely shared plug-ins for parametric design platforms, including particle-physics engines, structural finite element- and solar analysis as well as tool-path simulations. Digital physical models are accessible to anyone and can play a significant role in defining parametric design-elements. By their nature, they only yield numeric performance indicators, which might be less suited for interpretation by a human than by a machine, respective of a genetic algorithm. Especially in fields of engineering, the applicability of genetic algorithms to optimize performance problems has been shown (Kicinger 2005).

Still, as Manuel de Landa speaks of artistic use of genetic algorithms he states, "Only if virtual evolution can be used to explore a space rich enough so that all the possibilities cannot be considered in advance by the designer, only if what results shocks or at least surprises, can genetic algorithms be considered useful visualization tools" (DeLanda 2001).

For this idea he quotes Gilles Deleuze as the first to bring three imperative forms of philosophical thinking together: population, intensive, and topological thinking. The application of topological principles hereby is the most important. It defines the mapping of how an abstract genome is turned into an actual solution. It is also the biggest quest to the exploratory parametric designer

to find a relationship between genotype and phenotype as generic as possible, yet precise enough to have realistic chances of finding usable solutions.

The discourse about the nature of 'parametric modeling' lasts since its beginnings with the rise of CAD software applications, such as Ivan Sutherlands 'Sketchpad' from 1963. His novel computer program enabled the user to draw on screen with a light-pen, the shape could then be altered by a simple 'parametric engine' that was built in. The vision of the computer as an uber-human design aid already existed at that time. John von Neumann, the inventor of modern chip architecture, in 1951 published 'General and Logical Theory of Automata' (Neumann 1951). Similar to Stephen Wolfram's 'a new kind of science' from 2002, Neumann foresees the potential of computational strategies that rely on simple rules for an intricate output.

Genetic algorithms, representative for stochastic evolutionary algorithms here, rely on simple and intuitive rules derived from biological development. There are a variety of specializations just as in nature, but the basic principles of selection, mating, crossover, mutation, and evaluation mostly outline the central mechanisms. Yet they perform a manifold of tasks that, if they were to be solved analytically, usually would require significantly more elaboration. Theoretically they are universal problem solvers, although practically limited by the relation of the problem's degrees of freedom and the runtime needed to arrive at satisfactory solutions. Still, the prototypic building industry and its conditions of economic production allow the success of evolutionary algorithms in architecture and engineering. Further, contemporary visual scripting environments that grow from a pool of shared add-ons provide extensive possibilities to set up the definition of a complex problem relatively fast.

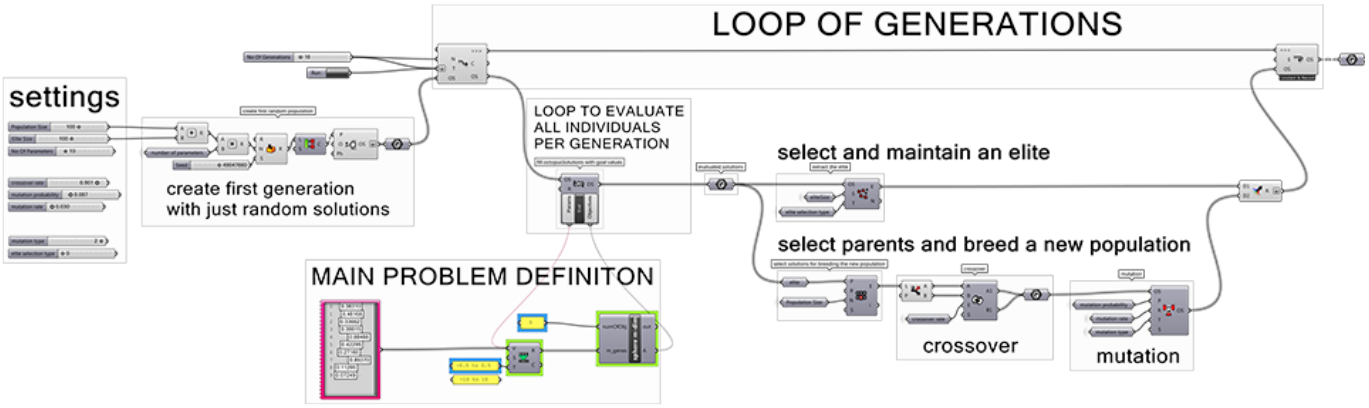
CUSTOM EVOLUTION

One major advantage of visual scripting environments is their simplicity in usage compared to lines of code, though there is a threshold when writing code again becomes more efficient for advanced tasks. Today, a few generic evolutionary solvers are publicly available for parametric modeling frameworks. A tool for multi-objective evolutionary search has been introduced recently, which is geared for flexibility during a search process in regards to changing parameters, objectives, and user choices to eventually become a parametric design assistant instead of a singular event of optimization (Vierlinger 2013). Still, neither this one released as 'octopus', nor another public tool for parametric modeling frameworks so far enables the user to customize the actual search algorithm.

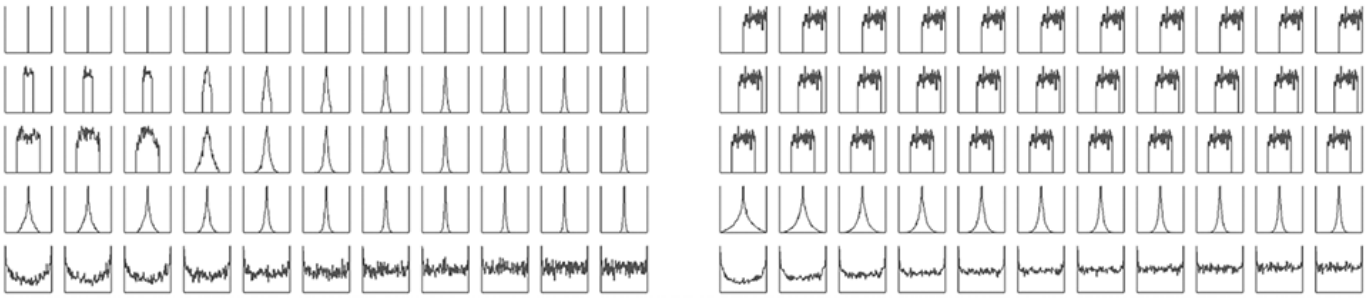
A set of functional components called *octopus.Explicit* is proposed, which can be combined, adapted, and replaced for different specialized purpose of evolutionary search (Figure 1). The essential groups of components are:

- Solutions: Data objects containing any characteristic of a solution such as parameter values, objective values, phenotypic representations, etc.; include components to construct and deconstruct solutions from and into native data types.
- Selectors: To select one or more solutions out of a pool, according to different criteria
  - Pareto Tournament Selection
  - Partitioning into Pareto Front and subsequent Front Ranks
  - Elite Selection after different criteria such as the Hypervolume Contribution (Bader 2006) or the archiving routine of SPEA-2 (Zitzler 2001)
- Operators: Stochastic modification of solutions to find a better configuration of parameters
  - Mutation after different strategies such as Polynomial, Alternative Polynomial (Deb 1996), or HypE Mutation (Bader 2006)
  - Crossover after (Deb 1994) as Simulated Binary Crossover for continuous search spaces
- Evaluators: To compute the solution's objective values depending on their parameter values
  - Diversity of Parameters: To introduce genotypic diversity as an optimization goal
  - Diversity of Objectives: To introduce objective diversity as an optimization goal
  - Evaluator: To evaluate the actual problem and compute the objective values of a parametric representation; the component supports local multi-threading and distributed computing for improved performance.
  - Evaluation Receiver: To receive an evaluation-context on a remote machine, enabling distributed computation

There has been extensive research on strategies for general evolutionary optimization, each defining different strategies for the iterative processing of solutions. By explicitly formulating the search algorithm in the same environment as the problem is defined, a more flexible reaction to special requirements is possible. Components can be replaced or added either by a sub-system of



2 Basic Multi-Objective Genetic Algorithm Using Octopus. Explicit Components



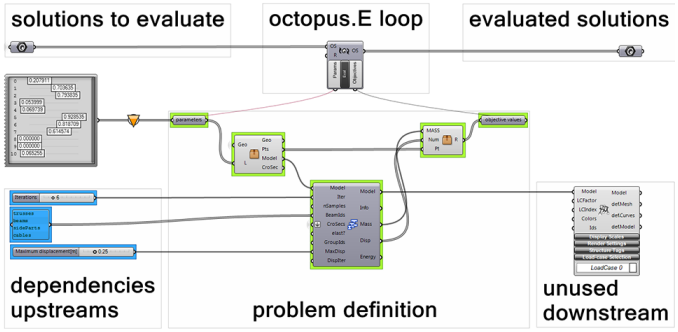
3 Statistical Comparison of Two Mutation-Operators Made in Grasshopper

the functional graph or an embedded piece of code, levering the vast possibilities of native and add-in components of parametric modeling software to compose a specialized optimization pipeline.

(Figure 2) shows the setup of a basic multi-objective evolutionary algorithm modeled with the explicit components. In the example, the sphere-problem as an academic benchmark of multi-objective optimization algorithms is used. The inner loop to evaluate the solutions is performed by the octopus.E loop component, which is explained later. The outer loop over the evolution's generations is facilitated with the third-party plug-in Anemone (Zwierzycki 2014), for reasons of read and usability, and because the performance needed by the Selector and Operator components is negligible. A benchmark between different looping tools is presented at the end of this work.

With the set of explicit components, many different extensions to a simple evolutionary algorithm are possible, some of them are:

- Elitism, where a percentage of solutions to modify by the Operator-components is selected not from the entire population but from the Elite.
- A biased Mutation-scheme, which takes into account analytical properties of the respective solution or relations of the solution to the rest of the generation. This way, an agent-like behavior can be introduced to form a hybrid between heuristic and deterministic optimization. (Figure 3) shows the statistical study of two different mutation operators (Polynomial Mutation and HypE Mutation), performed within *Grasshopper*. It exemplifies the advantage of developing, testing and applying within a single environment.



4 Use of octopus.E's Evaluator-component

- A Mutation scheme which considers the sensitivity of each gene, mutating parameters to different extents dependent on their level of influence on the solution.
- Lehmann and Stanley (2011) propose a general approach to evolutionary optimization which they call Novelty Search. It implies the abandoning of objectives, respectively the reward of innovation in a search process. Inspired by this, diversity in a search can be fostered by defining a distance measure of one solution to the rest of a generation, which can be in genotypic, phenotypic, or objective space. Defining this distance as an additional objective would impose a fitness pressure towards the evolution of a diverse set of optimized solutions. In this sense, Novelty Search could be called similar to diversity preservation techniques commonly used in multi-objective evolutionary algorithms.
- The type of parameters is not limited to numbers, and could be actual geometry. If the operators are modified or replaced accordingly, any type of data could serve as the genome of a solution.

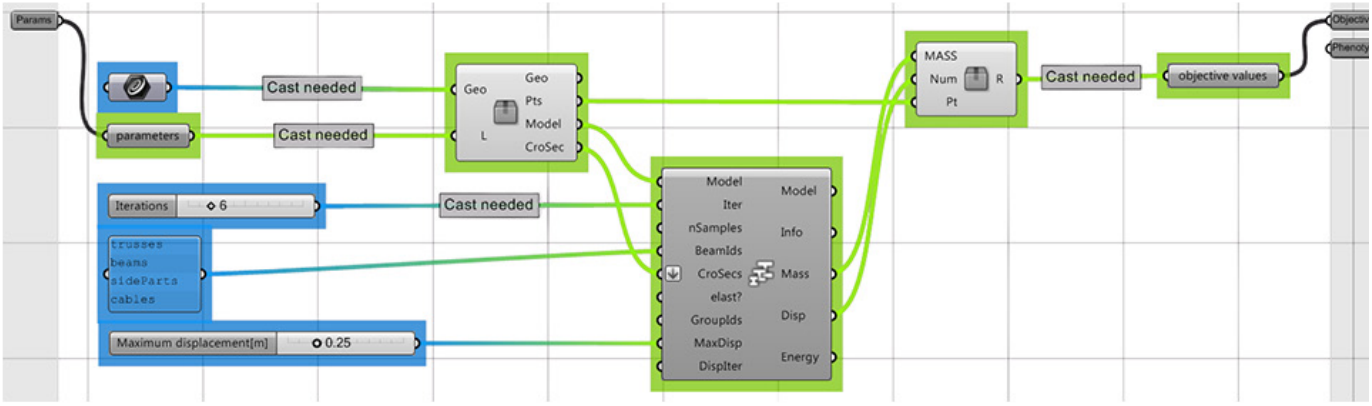
LOOPING AND PARALLELIZATION

The Evaluation-component works different from other approaches to iteration and looping in *Grasshopper*. Technically, it does not constitute a breach in *Grasshopper*'s policy to always keep the focus of computation on just one component. Similar to Galapagos (Rutten 2013), it connects to a part of the definition and evaluates it with specific parameters. Galapagos does this by using native processes: it writes the parameter values into the connected components and waits for all downstream elements to calculate until it finally reads the goal values. Hence it can never be part of the definition itself, since its own evaluation would require other components to compute. Our Evaluation-component analyzes the network it is connected to, and copies the network and its dependencies into a separate abstract document, called context. (Figure 4) shows the basic use of the Evaluation-component. It connects to a part of the definition and highlights the relevant parts in colored frames and symbols. Upstream connections on a connected input are not considered by the evaluation since the parameters will be chosen and supplied by the evolutionary algorithm; they are marked by an orange triangle. However, those connections can still be used for debugging or using the definition from another point within the setup. Upstream dependencies are highlighted in blue; the components which are part of the actual evaluation are marked in green. The context which is being copied by the evaluation-component can additionally be shown and edited in a separate window (Figure 5). This window also shows bottlenecks in the performance of the definition. For example, the flexibility which Grasshopper provides by automatic casts between data-types can have significant negative influence. Octopus.E's loop component does not evaluate the components on the canvas, but it is still using *Grasshopper*'s native methods to compute each component's solution. Since the constant updating of the user-interface is not necessary, this brings a significant gain in performance–still the evaluation would always work correctly. Code optimizations within the copied context, for example regarding expensive type-casts, are possible but have been discarded for the robustness of the component. A performance-benchmark with other looping plug-ins for *Grasshopper* can be found at the end of this paper.

In general, the practicability of evolutionary algorithms suffers from their immanently long runtime. Mostly reliant on stochastic methods, the number of solutions to be evaluated increases with the complexity of the problem. Planning tools in the prototypic architectural industry benefit from an immediate feedback as a true design tool. Therefore, we propose the distribution of computational load by:

- Local multithreading, unscheduled
- Local multithreading, scheduled
- Distributed computing within a local network or the Internet

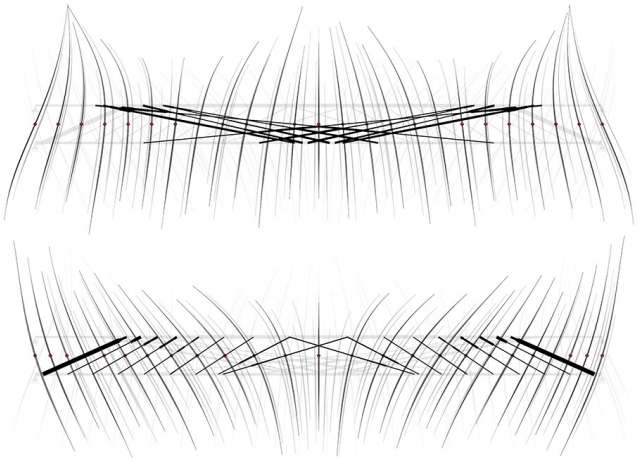
Local multithreading can be problematic with regards to the very core architecture of the underlying functions that are used by *Grasshopper* and *Rhinoceros*. They are not designed to be thread-safe, which means their concurrent execution on CPUs sharing the same memory can lead to unexpected results and crashes. A naive scheduling is implemented, which reflects the code of each component in the evaluation-context and makes sure no identical functions are executed at the same time. However, the performance gain by this method strongly depends on the problem definition,



5 Copied Evaluation Context

further the number of CPUs on the local machine mostly is limited to four or eight. There are other plug-ins enabling iteration and loops in *Grasshopper*, but aside their lack of parallelism they also are not as performant in single-threaded mode.

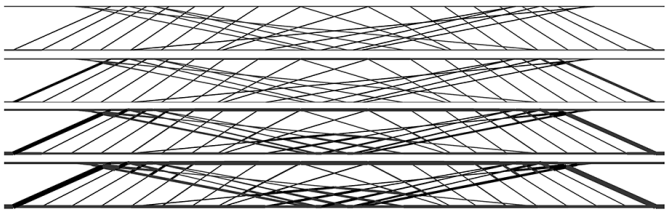
For this reason, a prototypic framework for distributed computation of our evaluation-units is introduced. The evaluation-receiver component (Figure 1) is placed on the canvas of a separate session of *Rhino* and *Grasshopper*, which has to be configured in the same way as the initial instance regarding the versions and plug-ins needed to run the definition. All evaluation-components register themselves on a server, which then manages the split and distribution of a pool of solutions to be evaluated. The distribution is naive, meaning the pool is split up uniformly without any load balancing approach. This means that the weakest machine determines the overall performance of the evaluation, since the master-component has to wait for all solutions to be computed. The outsourcing of an optimization's computational load also opens up the possibilities of a continuously accompanying design assistant which is acting in the background.



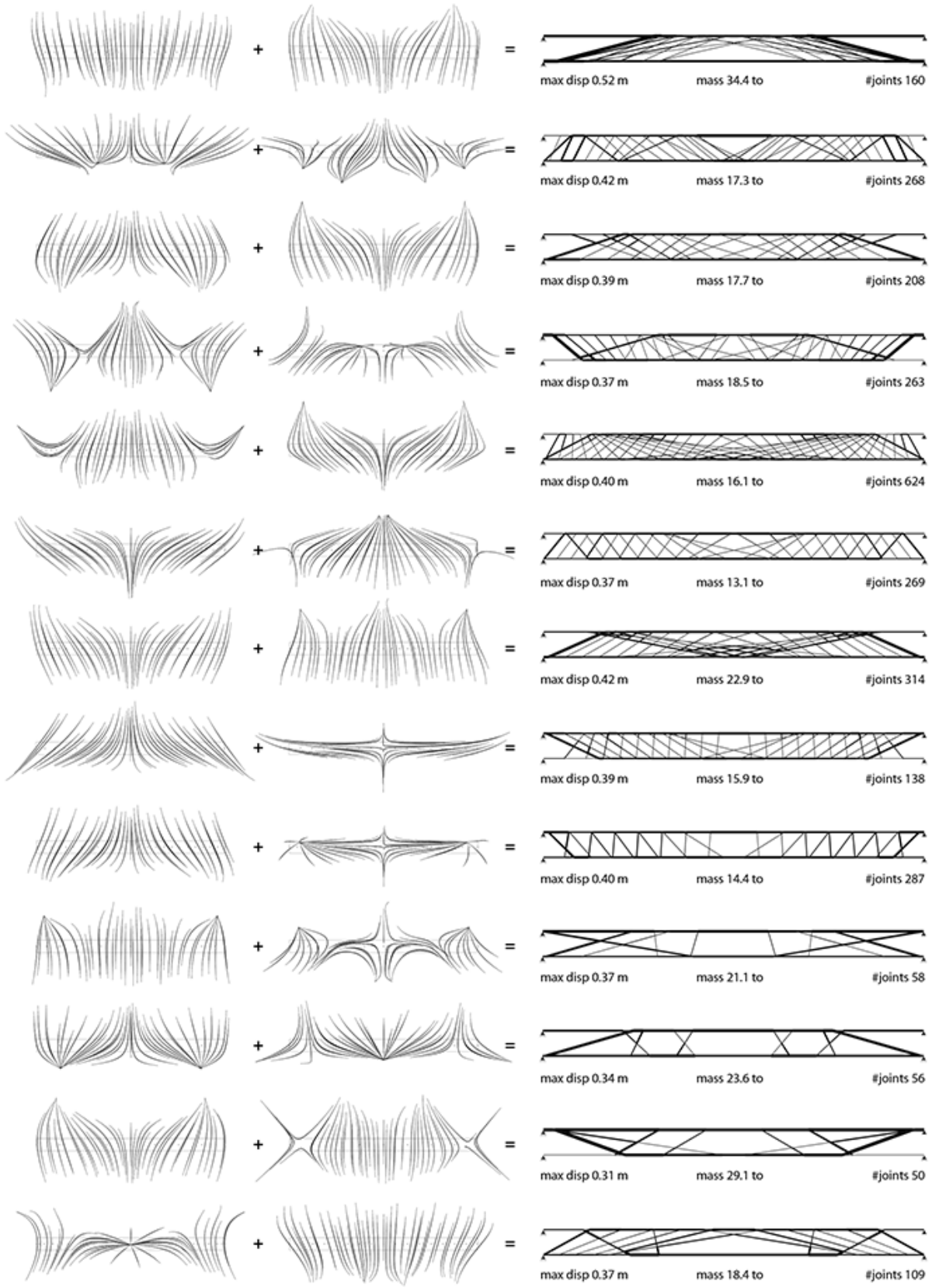
6 Translation of Two Magnetic Fields into Two Sets of Truss-Diagonals

FLEXIBLE PARAMETERIZATION

The optimization of a truss is used as an illustrative, yet simple example. Parametric finite element analysis using karamba for *Grasshopper* (Preisinger 2013) offers high-performance assessment of the trusses' load bearing capacity. Additionally, it is able to iteratively size the cross sections of its members according to their maximum allowable stress (Figure 7). Hence, the maximum displacement and the resulting mass are goal values to be minimized, as well as the number of joints as a measure for economic fabrication. Further, genetic diversity is imposed as a custom goal with octopus.E like described above.



7 Iterative Cross Section Optimization of the Truss Members

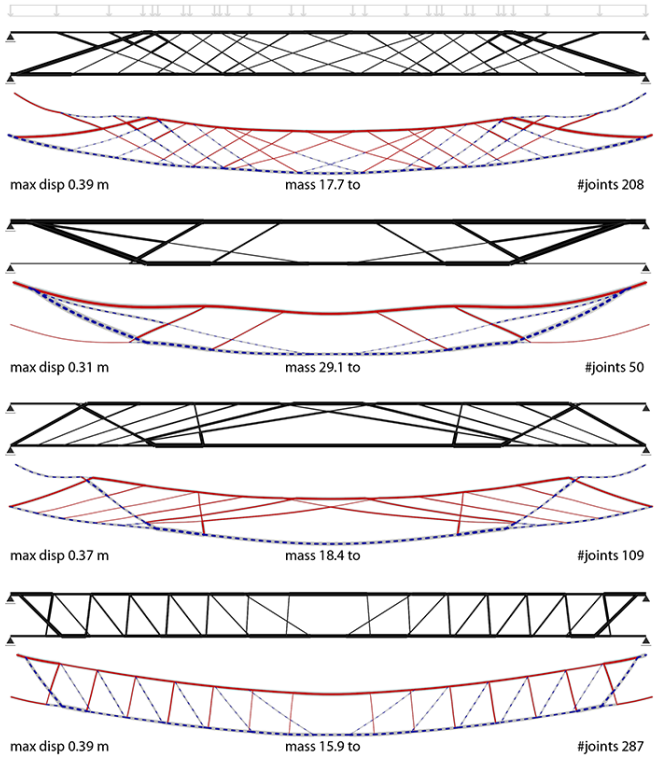


8 Selected Solutions after Three Generations of Size 200

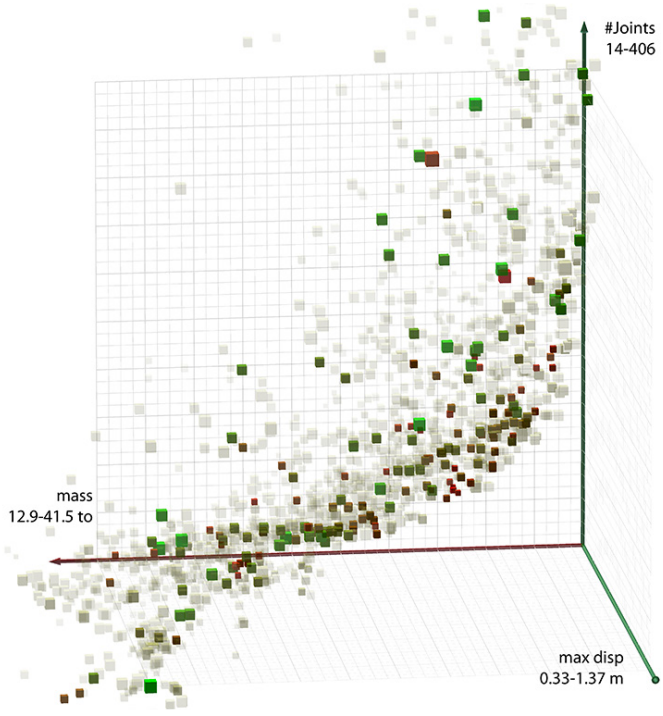
The geometry of the chords is fixed as straight lines and the variables of each solution are the positions of the diagonals. A simple parameterization would be the introduction of a parameter for the two ends of each strut. This straightforward way can have advantages for some applications, but for larger numbers of diagonals, the amount of parameters becomes too high for an efficient optimization. Also, a truss would mostly exhibit a certain degree of regularity, respectively enabling a reading of the underlying structural mechanisms. These would be hard to accomplish by the approach of an individual parameterization. Instead, we define two directional fields spanning throughout the bounding box of the structure. The fields each are parameterized by three points of simulated magnetic charge, adding to a total of twelve parameters. Three more parameters are used to determine the distribution-functions and offset of each set of diagonals. Along the middle axis of the truss, these distribution functions create points, for which the directional fields are evaluated. A point and a direction then define a diagonal (Figure 6). The resulting line geometry is turned into a structural finite-element model, and loaded by 1.35 times its dead weight and a line load of thirty kN/m on its upper chord. (Figure 8) shows selected results after three generations and a population size of 200, where bearing mechanisms already are clearly readable. The distribution of tension and compression within those mechanisms, as well as the deflected shape are further illustrated in (Figure 9). (Figure 10) shows a plot of the solutions in objective space forming the Pareto-front approximation, giving feedback on the best trade-offs that were found so far. The color and size in the diagram correspond to an abstract measure of diversity.

Hofmann (2011) uses a direct parameterization as described above, and needs significantly higher runtime to arrive at coherent solutions. This shows that encodings using superimpositions of fields and functions are at least promising when trying to digitally breed and evolve functionally meaningful patterns.

(Figure 11) shows the setup for a benchmark between different tools for looping a definition in *Grasshopper*. 'Hoopsnake' (Chatzikonstantinou 2014) and 'Anemone' (Zwierzycki 2014) are compared to the looping-tool introduced in this work. The resulting runtimes are written in (Table 1). It is shown that also in single-threaded mode octopus.E performs best due to savings at user-interface updates. Hoopsnake exhibits a function to deactivate the previews during execution of a loop, but still is almost twice as slow as octopus.E. In local multithreading mode, octopus.E is slower than single-threaded due to the use of karamba in the truss example, which already uses all the cores available. The resulting overhead in management of the different threads then



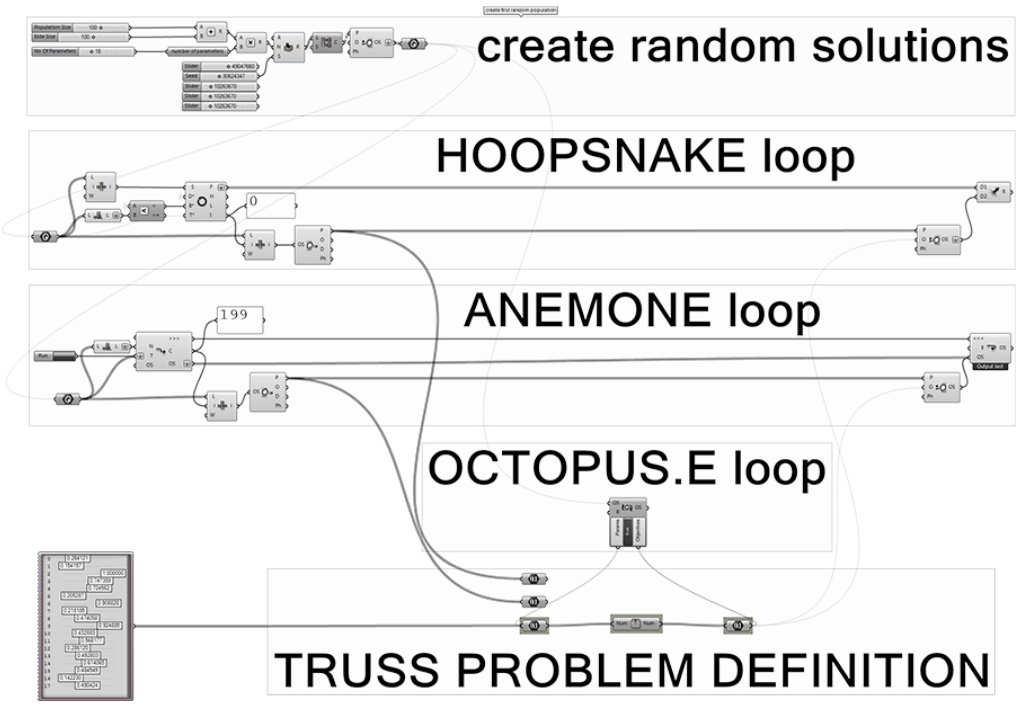
9 Finite Element Analysis of Truss Solutions



10 Solutions to Truss-Optimization in Objective Space

Number of Solutions	octopus.E Multi Distr. 4 Machines	octopus.E Multi Local	octopus.E Single Local	Anemone	Hoopsnake Hidden Preview	Hoopsnake Visible Preview
200	3.6 s	12.7 s	9.7 s	25.9 s	18.6 s	22.3 s
400	6.1 s	26.1 s	20.1 s	50.4 s	37.4 s	45.2 s

Table 1
Benchmark Results for the evaluation of the Truss Examples' Solutions



11 Benchmark Setup with Truss Example

leads to the performance loss. The experimental setup of distributed computing on four machines in a local network shows by far the best results. Still it is to be noted that the setup is experimental and under well-tuned conditions. Large geometric dependencies which have to be transferred over the network in each generation, or differently powerful slave-computers can degrade the performance of this approach quickly.

CONCLUSION

The experiments explained in this work try to exemplify original ways of thinking about how we use contemporary design tools. Simple principles are used along with the idea of modularity and flexibility. Paired with improved performance through parallelization, the setup and application of developmental

biology's principles is tried to be somewhat defanged. However, there are many chances regarding interface, workflow, and immediacy of parametric modeling environments with respect to augmented performance intelligence. The question of the role of the model in an architectural project is tightly related to the task of maintaining flexibility throughout the digital chain to not only improve a design's ultimate quality, but also the workflow in practice. With new paradigms in modern parametric modeling on the other hand, there is great potential in the research of representations, and the theoretically unlimited pool of formal languages waiting to be described.

REFERENCES

Bader J., Deb K., Zitzler E. 2010. 'Faster Hypervolume-based Search using Monte Carlo Sampling', M. Ehrgott et al., editors, Conference on Multiple Criteria Decision Making (MCDM 2008), volume 634 of LNEMS, pages 313–326, Heidelberg, Germany.

Bittermann M.S. 2009. 'Intelligent Design Objects (IDO) - A cognitive approach for performance based design', Phd diss., TU Delft, Faculty of Architecture, http://repository.tudelft.nl, accessed Apr. 2014.

Bollinger K., Hofmann A., Preisinger C. 2011. 'Algorithmic Generation of Complex Spaceframes', Conference Proceedings IABSE-IASS, p.445, London.

Bhooshan S., Zaha Hadid Architects 'Computation and Design (co|de) group', http://www.zha-code-education.org/, accessed Apr. 2014.

Chatzikonstantinou, Y. 'Hoopsnake', software, http://yconst.com/software/hoopsnake/, accessed July 2014.

Davis, D. 2013. 'Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture.'' PhD dissertation, RMIT University.

Deb K., Goyal M. 1996. 'A combined genetic adaptive search (GeneAS) for engineering design', Computer Science and Informatics, 26(4), p. 30-45, 1996.

Deb K., Agrawal R. B. 1994. 'Simulated Binary Crossover for Continuous Search Space', IITK/ME/SMD-94027, Convenor, Technical Reports, Indian Institue of Technology, Kanpur, India.

De Landa M. 2001. 'Deleuze and the Use of the Genetic Algorithm in Architecture', Design for a digital world, p. 117-120, Wiley-Academy, Wiley, New York.

Kicinger R., Arciszewski T., De Jong K. 2005. 'Evolutionary Computation and Structural Design: a Survey of the State of the Art', Journal of Computers and Structures, Volume 83 Issue 23-24, p. 1943-1978.

Lehman J., Stanley K.O. 2011. 'Abandoning Objectives: Evolution through the Search for Novelty Alone', Evolutionary Computation journal, (19):2, p. 189-223, Cambridge, MA: MIT Press.

Neumann, John von 1951. 'The general and logical theory of automata', Pergamon Press.

Preisinger, C. 2013. 'Linking Structure and Parametric Geometry', AD Architectural Design Issue 'Computation Works', p. 110-113.

Rutten, D. 2013. 'Galapagos – On the logic and limitations of generic solvers', AD Architectural Design Issue 'Computation Works', p. 132-135.

Stanley K.O. 2007. 'Compositional Pattern Producing Networks: A Novel Abstraction of Development', Genetic Programming and Evolvable Machines, Special Issue on Developmental Systems, New York, NY: Springer.

Vierlinger, R. 2013. 'A Framework for Flexible Search and Optimization in Parametric Design', Rethinking Prototyping - Proceedings of the Design Modelling Symposium Berlin, Berlin.

Wolfram, S. 2002. 'A New Kind of Science', Wolfram Media, Champaign, IL.

Zitzler E., Laumanns M., Thiele L. 2001. 'SPEA2: Improving the Strength Pareto Evolutionary Algorithm', Swiss Federal Institute of Technology (ETH) Zürich, TIK-Report 103.

Zwierzycki, M. 'Anemone', software, http://www.grasshopper3d.com/group/anemone, accessed July 2014.

IMAGE CREDITS

All image credits to Robert Vierlinger (2014).

ROBERT VIERLINGER is a researching engineer and interdisciplinary consultant. Working on his PhD at University of Applied Arts Vienna, he investigates on digital representations and evolutionary design strategies. Robert develops the plug-ins octopus and octopus.e for Grasshopper. Further, he is involved in the development of karamba. Parametric engineering and optimizations for international competition- and construction-projects are the basis of his consultancy at Bollinger-Grohmann engineers. He studied structural design at TU Delft and TU Vienna, studied at Studio Hani Rashid Vienna, led workshops in Germany, England, Denmark, Hong Kong and Austria, and teaches at Studio Zaha Hadid Vienna.

KLAUS BOLLINGER has studied Civil Engineering at the Technical University Darmstadt and taught at Dortmund University. Since 1994 he has been Professor for Structural Engineering at the IoA/University of Applied Arts Vienna, since 2012 he is the Institute's dean. In 1983, Klaus Bollinger and Manfred Grohmann established the practice Bollinger + Grohmann, now located in Frankfurt am Main, Berlin, Vienna, Paris, Oslo, Melbourne, and Berlin with around 200 employees. The office'sscope of work includes building structures, facade design and building performance.