Open-Ended Evolution With or Without A Priori Knowledge Included?

Jim Torresen

Department of Informatics
University of Oslo
P.O. Box 1080 Blindern
N-0316 Oslo, Norway
jimtoer@ifi.uio.no
http://www.ifi.uio.no/∼jimtoer

Abstract. Several approaches exist to improve the quality of evolution. In this paper, a priori knowledge applied to open-ended evolution will be discussed. It will be focused at assessing the positives and negatives related to *extending* the complexity of evolved systems and *limiting* the evolutionary exploration, respectively. Relevant knowledge relates to functional modularity, structural regularity as well as hierarchy.

1 Introduction

Open-ended evolution represents systems where the evolution is continuous and not stopped by a fitness criteria. There is an expectation that evolvable systems would become crucial in the future development of computer systems as traditional design schemes are reaching their limits. Increasing size and complexity of electronic devices and systems have during the recent years led to a demand for new design schemes and tools. The new technology could be appropriate for systems implementing complex real-world applications within image and signal processing. Much research is now conducted on such applications to improve the performance as well as the speed of processing using traditional algorithms.

Evolving a system is promising but there is still a long way to go before it is a real alternative to traditional design schemes [6]. By searching a larger design search space, evolution may find solutions for a task, unsolvable, or more optimal than those found using traditional design methods. However, this is also a problems since the search space easily becomes too large [2, 12]. Starting from scratch when evolving is not very biological motivated. Human beings try to apply all their earlier knowledge and skills when trying to solve a problem. If we are not able to solve it by ourselves, we search knowledge either in books/web etc. or by getting help from computer tools or other people, who have more knowledge or skills, to solve the problem. In evolution, there has been a tendency that introducing human expert knowledge would limit the exploration of the large search space. Thus, risking to loose interesting solutions that humans would not in their wildest dreams have thought of. This is true, but so far few such revolutionary systems have occurred.

Therefore, we find it appropriate to think it the other way around. If we tell the system what we know already, the evolution can go on from there to explore new and revolutionary systems. This is motivated by "inventing the wheel": If you know about the wheel, there is a much higher chance that you could invent a vehicle of some sort than if you did not.

Incremental evolution for evolvable hardware (EHW) was first introduced by Torresen in [4]. This is an approach that uses divide-and-conquer of the application. It has been shown that dividing the application is a very promising approach. It was proposed for EHW as a way to incrementally evolve a hardware system. The scheme is called *increased complexity evolution* since the system is evolved by evolving smaller subsystems. Increased building block complexity is also a part of this approach, where the building blocks are becoming more complex as the system complexity increases.

Experiments show that the number of generations required for evolution by the new method can be substantially reduced compared to evolving a system directly in one operation. This has been shown both for a character recognition problem [5], a road image recognition problem [7] and prosthetic hand control [8]. In some experiments, better classification results than for artificial neural network were obtained. In addition, the hardware circuit was much smaller than what would have been required for running a neural network. Further, circuits for larger problems – than those evolvable by other schemes, have been evolved [11].

Through the research, the architecture as well as the incremental evolutionary algorithm have been extended and improved. It has been shown that even though the total number of generations is less, the performance of the evolved circuit is substantially better with the proposed scheme [8]. Thus, one achieves to solve complex problems where no good solution can be found by a single run evolution. Moreover, different ways of conducting the evolution have been proposed and tested. This includes evolving the best combination of circuits among a set of alternative circuits [10] and dynamic fitness functions in evolvable hardware [9]. Thus, we believe this approach is a good foundation for introducing *more* human *knowledge* into the design. That is, more manual design knowledge would be applied in the evolution. This is the topic to be further presented in this paper. By including more *a priori* knowledge, we should be able to evolve more complex and thus, more useful systems than the systems that have been evolvable so far.

2 A priori Knowledge

This section gives an overview of different aspects of design knowledge [3]. That is, we list those methods which a designer would apply to design a system. A priori hardware design knowledge is applied in a set of different ways:

- **Design specification.** Most traditional design is according to a predefined detailed *design specification* of the system. In evolution of digital circuits, normally only input-output vectors are specified. Specification of e.g. timing is rare even though this is often important in traditional design.
- **Partitioning the design task.** This regards how to best partition a given problem to solve. This design scheme is also called a hierarchical design scheme. For normal

design, it will be a mixture of top-down and bottom-up design. A designer would normally start with preparing a top-level block description and continue by implementing one sub-circuit of the system at a time. He would usually know what is a reasonable partition of the design-task whereas this would normally not be available for a fully automatic design of a system. However, this would be quite similar to design based on increased complexity evolution presented in the introduction. We start with a complete data set that is partitioned (top-down). This is followed by a bottom-up evolution of the system. Thus, a priory knowledge is put into the top-down design only. Another aspect – at lower level, is to emphasize on the parts (or function) of the system not working. That is, when evolving a system, the fitness function should be adaptable to change its behavior according what is still left to be solved. This would have to be done in such a way that the parts working do not stop working.

- **Reuse.** Designing a large circuit would be almost impossible if the designer had to design each sub-circuit from scratch every time it is used [1]. Predesigned units for evolution could be simple Intellectual Property (IPs) cores, custom functional blocks or design library objects. Further, an interesting aspect is to re-use *evolved* structures several times in a system. This could either be about extracting promising parts of a chromosome [1] or apply evolved units in later evolution [4].
- **Data buses.** Almost no digital design is conducted without the use of data buses, while many systems are evolved with single bit wires only. To increase the complexity of a design, data buses would probably have to be included. Still, single bit lines will be needed. Thus, an evolvable architecture would have to consist of both busses and bit lines. This issue is also related to reuse with applying functional blocks in the evolution. A scheme for evolution with reuse and data buses is presented in [3].
- **Design optimization knowledge.** There exists a set of computer based optimization tools that could be applied in hardware design. That is, the designer specify a more or less abstract description of the system which the tool synthesize an optimized design for. Not much work has been published on combining evolution and optimization tools.
- **Prototyping.** This is very closely related to evolution. It consists of building various designs – with alternative architectures, to compare what is best. This is inherently what evolution consists of as well.
- **Hardware/software co-design.** Normally, hardware is developed in close cooperation with software being developed. What to implement in hardware and what to code as software are often an open question in the initial design phase. In evolution, designs are mainly either for software *or* hardware.

Not all of these issues are straightforward to combine with evolution. The one most explored (that probably will be more explored in the future) seems to be partitioning of the design task. Another one that probably could be useful is reuse. This could be at several levels. One of the problems with fixed length genetic algorithms is that a building block found in one part of the chromosome could not be reused in another part [1]. Further, standard crossover (with random crossover point) is not structure preserving. There have been introduced algorithms trying to overcome these problems. However,

we feel that reuse should be further investigated when trying to build complex structures and systems. We have undertaken experiments involving both reuse as well as data buses. In the future, the goal is to *combine* as much a priori knowledge as possible together to make advanced open-ended evolution.

3 Conclusions

This paper has introduced various aspects of how a priori knowledge can be applied to open-ended evolution. It has been focused at assessing the positives and negatives related to *extending* the complexity of evolved systems and *limiting* the evolutionary exploration, respectively.

Acknowledgments

The work presented in this paper is a part of the project *Biological-Inspired Design of Systems for Complex Real-World Applications* (project number 160308/V30) funded by the The Research Council of Norway.

References

1. J.R. Koza et al. The importance of reuse and development in evolvable hardware. In J. Lohn et al., editor, *Proc. of the 2003 NASA/DoD Conference on Evolvable Hardware*, pages 33–42. IEEE, 2003.
2. W-P. Lee, J. Hallam, and H.H. Lund. Learning complex robot behaviours by evolutionary computing with task decomposition. In A. Birk and J. Demiris, editors, *Learning Robots: Proc. of 6th European Workshop, EWLR-6 Brighton*, volume 1545 of *Lecture Notes in Artificial Intelligence*, pages 155–172. Springer-Verlag, 1997.
3. J. Torresen. Exploring knowledge schemes for efficient evolution of hardware. In *Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware*.
4. J. Torresen. A divide-and-conquer approach to evolvable hardware. In M. Sipper et al., editors, *Evolvable Systems: From Biology to Hardware. Second International Conference, ICES 98*, volume 1478 of *Lecture Notes in Computer Science*, pages 57–65. Springer-Verlag, 1998.
5. J. Torresen. Increased complexity evolution applied to evolvable hardware. In Dagli et al., editors, *Smart Engineering System Design: Neural Networks, Fuzzy Logic , Evolutionary Programming, Data Mining, and Complex Systems, Proc. of ANNIE'99*, pages 429–436. ASME Press, November 1999.
6. J. Torresen. Possibilities and limitations of applying evolvable hardware to real-world application. In R.W. Hartenstein et al., editors, *Field-Programmable Logic and Applications: 10th International Conference on Field Programmable Logic and Applications (FPL-2000)*, volume 1896 of *Lecture Notes in Computer Science*, pages 230–239. Springer-Verlag, 2000.
7. J. Torresen. Scalable evolvable hardware applied to road image recognition. In J. Lohn et al., editor, *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 245–252. IEEE Computer Society, Silicon Valley, USA, July 2000.

8. J. Torresen. Two-step incremental evolution of a digital logic gate based prosthetic hand controller. In *Evolvable Systems: From Biology to Hardware. Fourth International Conference, (ICES'01)*, volume 2210 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 2001.
9. J. Torresen. A dynamic fitness function applied to improve the generalisation when evolving a signal processing hardware architecture. In *Applications of Evolutionary Computing: EvoWorkshops 2002*, volume 2279 of *Lecture Notes in Computer Science*, pages 267–279. Springer-Verlag, 2002.
10. J. Torresen. Evolving both hardware subsystems and the selection of variants of such into an assembled system. In *Proc. of the 16th European Simulation Multiconference (ESM2002)*, pages 451–457. SCS Europe, June 2002.
11. J. Torresen. Evolving multiplier circuits by training set and training vector partitioning. In P. Hadddow A. Tyrrel and J. Torresen, editors, *Evolvable Systems: From Biology to Hardware. Fifth International Conference, ICES'03*, volume 2606 of *Lecture Notes in Computer Science*, pages 228–237. Springer-Verlag, 2003.
12. X. Yao and T. Higuchi. Promises and challenges of evolvable hardware. In T. Higuchi et al., editors, *Evolvable Systems: From Biology to Hardware. First International Conference, ICES 96*, volume 1259 of *Lecture Notes in Computer Science*, pages 55–78. Springer-Verlag, 1997.