Finding the optimal sorting network by harvesting the power of coevolution

Ghiga, Claudiu-Alexandru
claudiu.ghiga@info.uaic.ro

Nistor, Serban
serban.nistor@info.uaic.ro

November 16, 2020

Abstract

Finding the optimal sorting network is still a challenging problem and many different methods exist in order to try and solve it. One particular strategy consists in using genetic algorithms in a coevolution manner. We will see in this paper that this method can be really interesting and that it can be extend in many ways with minimal effort.

Keywords— Optimal Sorting Network, Genetic Algorithm, Coevolution

1 Introduction

Sorting networks are data-oblivious algorithms that are used to sort numbers. In other words, regardless of the form that the numerical input data takes, a sorting network will produce as output a monotonically increasing sequence of the provided data. Sorting networks are composed of wires and comparators.

The problem that we have tried to solve consists in finding an optimal size sorting network for a given number n of wires. Since the problem of finding an optimal sorting network of size k necessitates checking for the existence of all sorting networks with size smaller than k and the number of possible configurations grows by $\frac{n}{2}$ for each new comparator added, the problem is in co-NP. Because of this, much of the research done in this domain focuses on finding lower and upper bounds for the size of sorting networks with n wires and in using heuristic algorithms in order to reduce the search space sorting networks.

2 Proposed Method

Our proposed method focuses on using a nature inspired method, namely coevolution to solve this problem. We designed our coevolution procedure by thinking at the prey-predator model, and in this case the prey is the input that the sorting networks are receiving, the networks being in this case the predator.

This method has an important advantage in our case, because our input data can be hard to compute if we aim for big sized networks, lets say with a size of 16 for example, so by using the proposed method, we can begin by generating only a part of the input, which will then evolve alongside our sorting networks. As you probably noted, we have two species that need to evolve in order to survive: the input population and the sorting networks population.

The input population is represented using binary encoding, each individual being a binary array with the same size. As for the sorting networks population, an individual is represented as a list of pairs, each pair representing a comparator. The generation of both the populations is random, and as for the actual sizes used, the input population has 100 individuals and the sorting networks population has 300 individuals.

The genetic algorithm uses as the fitness function for the sorting networks population $\frac{\#no\_of\_sorted\_cases}{input\_population\_size}$ and the opposite form, $1 - \frac{\#no\_of\_sorted\_cases}{input\_population\_size}$ for the input population. In this case, the best sorting network would have a fitness of 1 if it can sort all of the inputs from the input population, and symmetrically an input would be evaluated by the number of sorting networks which could not sort it.

The selection is done by using the rank-based selection in order to try to avoid premature convergence. This method, in general leads to a slower convergence towards a global optima, but in this particular case it helps with not remaining stuck in a local optima, so the slower convergence is not really a problem. We also use elitism as an addition to this selection procedure because we empirically observed that the genetic algorithm needs less iterations to escape local optima in the late stages of the evolution compared to the case when no elitism was in place.

The number of individuals selected by using elitism is 10 from the total population. This procedure helps keeping the best sorting networks in the evolution loop, but also the best testing cases because this selection is used for both of the species. Another important thing to mention is the fact that this selection procedure keeps the number of the individuals in both populations unchanged.

In terms of genetic operators, we use mutation and crossover and we will discuss them separately for each population. For the input population, the mutation is something standard and it is done by flipping one randomly chosen bit from the individual. The mutation rate is at 1%, which was chosen empirically in order to keep the balance between exploration and exploitation. The crossover for the input population is also pretty standard when it comes to genetic algorithms and it is achieved by randomly selecting an even number of candidates that will go through crossover; the crossover itself it is done by splitting each candidate in half and then combining the pieces from two distinct candidates. The probability that a candidate will undergo crossover is set at 60%.

Now, when it comes to the operators used on the sorting networks population, things are a little bit different. We have implemented three different possible mutations. The first one consists in choosing two random comparators and swapping their contents for a network, the second one will be adding a new comparator to the current individual, at a randomly chosen index and the third one will remove one randomly chosen comparator from the current network structure.

The probability for an individual to undergo mutation is of 1%, and then the probability to apply the swapping mutation is of 80%, the adding mutation of 10% and the removing mutation of 10%.

3 Experiments

The experiments were conducted on a regular computer (Intel i7-4720HQ @ 2.60Ghz), but no multi-threading was used in order to speed-up the genetic algorithm. All the experiments were run 5 times and in each experiment the algorithm was ran for 100 iterations. The results can be observed below:

n	**t(s)**	σ **(s)**	**NetworkFitness**	$\sigma_{\textbf{NetworkFitness}}$	**InputFitness**	$\sigma_{\textbf{InputFitness}}$
2	767.51	128.49	1.0	0	0	$5.41 \cdot 10^{-8}$
3	773.60	130.25	1.0	0	0	$3.76 \cdot 10^{-8}$
4	745.53	160.38	1.0	0	0	$1.44 \cdot 10^{-6}$
5	798.29	142.44	0.995	$2.98 \cdot 10^{-3}$	$5 \cdot 10^{-3}$	$2.98 \cdot 10^{-3}$
6	822.28	151.25	0.993	$4.71 \cdot 10^{-3}$	$7 \cdot 10^{-3}$	$4.71 \cdot 10^{-3}$
7	776.49	153.37	0.978	$8.69 \cdot 10^{-3}$	$2.2 \cdot 10^{-2}$	$8.69 \cdot 10^{-3}$
8	762.30	176.84	0.949	$2.76 \cdot 10^{-2}$	$5.1 \cdot 10^{-2}$	$2.76 \cdot 10^{-2}$

Table 1: Comparisons of execution times for finding the minimum size sorting network for $n = 2 \dots 8$. The Fitness columns represent what fitness achieved the best individual from each population when the algorithm stopped.

4 Conclusion

As we've already seen, the described method can yield great results, depending on the size of the input. Although, the possibilities at which this method can be extended and also the number of different implementations that can be done for the core components: the fitness function, the selection procedure and the genetic operators are limitless, therefore, to conclude, by further experimenting with coevolution on this particular problem, even greater results may be achieved.

References

[1] Parberry, I. 1991 *On the computational complexity of optimal sorting network verification.*

[2] Booker, L. 1987 *Improving Search in Genetic Algorithms.*

[3] Baker, J.E. 1985 *Adaptive Selection Methods for Genetic Algorithms.*

[4] Hillis, W.D. 1990 *Co-evolving parasites improve simulated evolution as an optimization procedure.*

[5] How to perform Roulette wheel and Rank based selection in a genetic algorithm?
`https://medium.com/@setu677/howtoperformroulettewheelandrank-basedselectioninageneticalgorithmd0829a37a189`

[6] Cooperative coevolution
`https://en.wikipedia.org/wiki/Cooperative_coevolution`

[7] Coevolutionary Principles
`https://www.cs.tufts.edu/comp/150GA/handouts/nchbmain.pdf`