KU Leuven
ESAT

Computer Aided IC Design
Genetic Optimization

Matthijs Van keirsbilck
Onur

KU Leuven , November 29, 2016

Contents

Motivation and Objectives

The goal of this project is to design a general-purpose multidimensional optimization algorithm based on genetic evolution. The algorithm is then applied to circuit design to obtain optimal parameters for the circuit to achieve certain specifications. This document describes the successive improvements we made to our algorithm, and lists the results of each improvement.

The Algorithm

Genetic algorithms can optimize multidimensional objective functions, resulting in a series of Pareto-optimal 'individuals' from which the designer can choose the one best suited for some application. This is a major advantage of these algorithms. The algorithm consists of several parts:

1. initPopulation: initializes the population to random values. See subsection 2.1.
2. evaluatePopulation: calculates the scores of each individual for each objective function. See subsection 2.2.
3. sortPopulation: sorts the population based on the objective function scores. Divides the population in ranks, where individuals in the same rank are pareto-equal to each other. A lower rank means Pareto-superior objective function scores. See subsection 2.3.
4. selectionTournament: selects parents out of the population. See subsection 2.4.
5. geneticOperators: generates children from the selected parents, by recombination or mutation. See subsection 2.5
6. cropPopulation: prevents the population size from growing. See subsection 2.6.
7. stopCriterion: determines when the algorithm converged. See subsection 2.7

InitializePopulation

This sets the population to a random matrix, where matrix elements are chosen randomly from a uniform distribution between 0 and 1. In order to make the population properties more general, more manageable, they are always scaled to the $[0, 1]$ interval, except for the calculation of the objective scores (see 2.2). Ranks are set to 1, and crowding distance to 0 at initialization (see 2.3).

evaluatePopulation

We save the rank and crowdingDistance columns, then unnormalize the population parameters in order to map from parameter space to objective space. The score for each objective function is calculated for each individual, and stored in columns V+1:V+M. The renormalized population parameters are stored in columns 1:V.

sortPopulation

This function finds individuals that are on the same Pareto-optimal curve, and puts those in the same rank. The best rank is two, the worst one is rank three (it doesn't make sense to have more ranks as

Ranking

Crowding Distance

The algorithm thus far doesn't give us good coverage of the whole Pareto-optimal solution space. It generates lots of individuals close together, which contain almost the same information.
In order to work more efficiently and achieve better coverage of the Pareto-optimal space (by preventing individuals to exist very close together in parameter space), the concept of crowding distance can be used. It is basically a measure of how close an individual is to other individuals in the parameter space. See Reference [1] for the algorithm of the crowding distance calculation.
An extra column is added to the population matrix (which has index V+M+2). While sorting the population, crowding distance is taken into account, so that individuals with higher crowding distance are moved to the top within their rank.
A problem arises however: when two individuals are close together both of their crowding distances will be low. If we then sort on crowding distance and remove the lowest scoring individuals, both might be removed. This is not the intention, as we want to keep one individual out of the group in our population. To prevent this crowding distance is recalculated every time after an individual is removed from the population. This slows down the algorithm, so we only do this after all individuals in the population have reached rank 1 (which means that we've most likely reached the Pareto-optimal curve). Before that, we only calculate crowding distance once.

selectionTournament

geneticOperators

cropPopulation

stopCriterion

Future Improvements

Summary of Achievements

With our implementation, we managed to obtain pretty good results.

References

[1] A. Seshadri, "A fast elitist multiobjective genetic algorithm: Nsga-ii," *MATLAB Central*, vol. 182, 2006.