KATHOLIEKE UNIVERSITEIT LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

DEPARTEMENT ELEKTROTECHNIEK

Computer-Aided IC Design
Design of an evolutionary algorithm

Arnout Devos
Joost Verplancke

December 3, 2015

Prof. G.Gielen
Ir. A.Coyette

Contents

1 Algorithms

1.1 Initialize population

The initialization of the population is done in a pseudo-random way. More specifically a uniform hypercube distribution is used through *Latin hypercube sampling*. This ensures a random uniform distribution in the population space.

1.2 Sort population

In the single objective case, a regular sort based on this objective gives a clear ranking of the solutions.

For the multi-objective case it is not as straightforward and always clear to tell which individual is better than another. Therefore pareto-optimality is introduced. The pareto-optimal state is a state in which no single objective (or figure of merit) can be improved, without worsening another objective. To determine which individuals have a better figure of merit and to attain a good spread, rank and crowding distance need to be computed respectively.

1.2.1 Ranking

Ranking is done based on non-dominated sorting. A solution i_1 is said to be dominate another solution i_2 if:

$$f_k(i_1) \leq f_k(i_2) \forall k \text{ and } \exists j : f_j(i_1) < f_j(i_2) \tag{1}$$

Based on this, every individual gets assigned a rank. The individuals who are not dominated by any other individual are assigned a rank of 1. The individuals who are not dominated by any other points but those of rank 1 are assigned a rank of 2. This process is repeated until every individual has a rank. This implementation can be found in Algorithm 1.

1.2.2 Crowding distance

At the point of convergence a good spread of solutions is preferable to give a large space of choice for further optimization. Points which are positioned further from other points in the same rank are favorable. Therefore the crowding distance from [2] is used as a measure. It is implemented in Algorithm 2.

1.2.3 Sorting for objectives

After the rank and crowding distance of each point have been determined, an ordered result can be generated by first sorting on rank (ascending) and then on crowding distance (descending).

Algorithm 1 Ranking algorithm

```
procedure RANKING(population)                                   ▷ Compute Ranking
    rankCurrent ← 1
    ok ← TRUE
    workingSet ← population
    while workingSet has individuals do
        for each individual i do
            for each individual j where j ≠ i do
                ok ← TRUE
                if m_k(i) ≥ m_k(j) ∀k then          ▷ Check for pareto breaking condition
                    if m_k(i) ≠ m_k(j) ∀k then          ▷ Trow out those with all equal m
                        individual i is dominated by j  ▷ Individual i isn't in current rank
                        ok ← FALSE
                        break
                    end if
                end if
            end for
            if ok = TRUE then
                rank(i) ← rankCurrent
                remove from workingSet
            end if
        end for
        rankCurrent ← rankCurrent + 1
    end while
end procedure
```

Algorithm 2 Crowding distance algorithm

```
procedure CROWDINGDISTANCE(population)                  ▷ Compute Crowding Distance
    for each rank F_i do
        Crowding distance (CD) of each individual in F_i ← 0
        for each objective m do
            sort individuals in F_i by ascending m
            CD of min and max m in F_i ← ∞
            for each individual k in F_i where m ∈ ]min(m), max(m)[ do
                CD(k) = CD(k) + (m(k+1) − m(k−1)) / (max(m) − min(m))
            end for
        end for
    end for
end procedure
```

1.3 Selection Tournament

The parents that will be used in the mating pool are selected in a pseudo-random way. More specifically, 2 parents are randomly (uniformly) selected from the sorted population and compared in terms of index. Lower index means lower rank and/or smaller crowding distance and is therefore chosen. Note that in order to retain higher ranked individuals, the population is virtually split in two and that the first part (with higher ranks) is favored. Special care has been taken to make sure a parent cannot be chosen twice in order to keep a diverse gene mating pool. This implementation can be found in Algorithm 3.

Algorithm 3 Selection of parents for mating pool

```
procedure SELECTIONTOURNAMENT(population, NP)
    selected ← {}
    while size(selected) < NP do
        workingSet ← 50% chance whole population & 50% chance best half population
        a ← random individual of workingSet
        b ← random individual of workingSet and b ≠ a
        Add individual (a or b) with lowest index to selected
    end while
    return selected
end procedure
```

1.4 Recombination and mutation mechanisms

If through recombination (with small mutation) or pure mutation, genes of a generated child would exceed the upper or lower bounds of the normalized space, it is bounced back to stay within the bounded interval.

An improvement of the current implemented recombination and mutation mechanisms would be Simulated Binary Crossover from [2].

1.4.1 Recombination

Two parents are randomly selected for recombination and each gene of every parent has a 50% chance of ending up as the resulting gene in the child. On top of this 50% chance selection a small mutation σ_{rec} is added to every gene (see Figure 1) in order to get slightly differing results which may be an improvement.

Instead of this recombination, linear interpolation (see Figure 2) between the genes of the parents is used in about 30% of the time. This is a good way to combine partial optimizations of linearly behaving variables (such as resistor values, transistor widths etc.) according to [3].

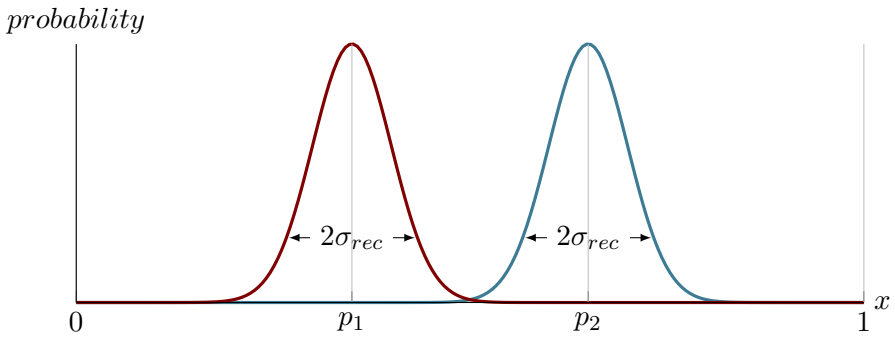


Figure 1: Recombination with small mutation on p_1 or p_2

Figure 2: Linear interpolation

1.4.2 Mutation

A Gaussian distribution is applied to a random amount of genes of one parent. The σ_{mut} of this distribution is slightly larger than the one used on top of recombination (see Figure 3).

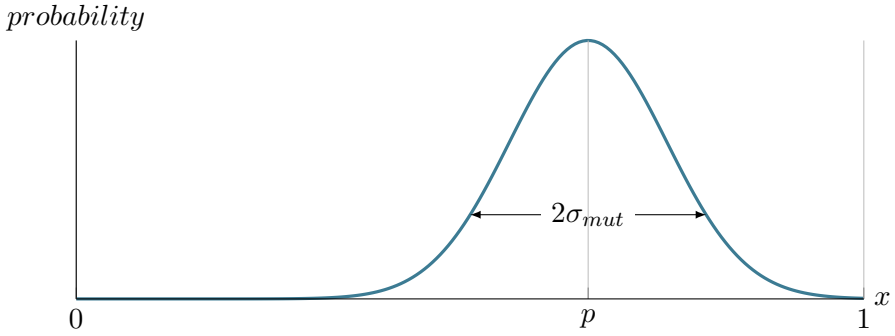


Figure 3: Pure mutation

1.5 Crop Population

To maintain the same number of individuals after every iteration, cropping the population is needed after sorting. A drawback of simply discarding the worst individuals (on the bottom of the sorted by rank and CD population), is that the crowding distance of a relatively bad individual might improve when discarding another closely positioned individual. If one discards this first individual solely on its crowding distance calculated on the full population, the spread might be deteriorated. A possible improvement could be that only one individual is discarded at a time and subsequently the crowding distance is recalculated and the population is resorted.

However, it takes a lot of time to recompute the crowding distance again each time. Therefore the (sorted) surplus of the population is split up into three parts where one part is deleted at a time and then the crowding distance is recomputed. This implies a total of two extra crowding distance calculations.

1.6 Stop Criterion

When convergence is reached, the algorithm should automatically end its operation. Therefore a stop criterion is needed. An ideal stop criterion would consist of all candidates having rank equal to one and a homogeneous spread. Due to the random nature of the algorithm chances are not high that this will happen in reasonable computation time while other solutions are still acceptable.

1.6.1 Rank

When the algorithm approaches convergence, one does not want pareto-suboptimal solutions. Therefore a first quick convergence check is done by stating that all individuals in the solution set should have a rank of 1. This also functions as a safety measure against premature stops if by bad luck only a small change in parameters is introduced early on in the run (see 1.6.2).

1.6.2 Area

Secondly, if the area under the solution set does not change considerably for a few iterations, it can be concluded that the algorithm has reached the optimal solutions border. Therefore a relative change less than 1μ in area is used as a measure for convergence.

Right now only two iterations are considered, this can be optimized by taking into account the relative change of area underneath the objective curve over the n previous iterations

1.6.3 Spread

The third part of the stop criterion considers the spread of the individuals. The crowding distance (from section 1.2.2) is a measure of the distance between individuals and therefore

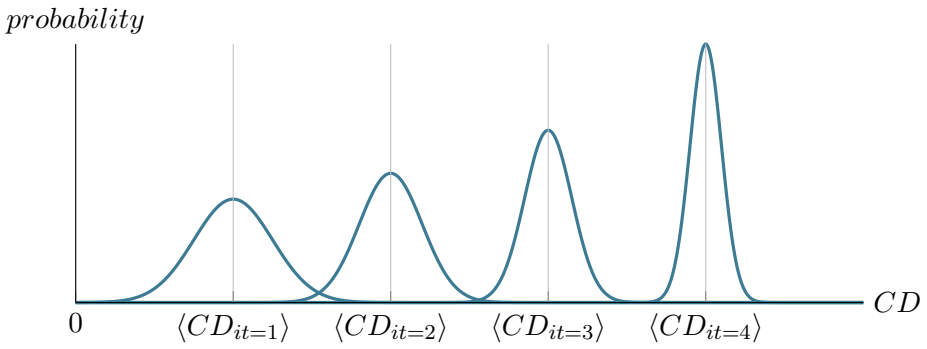


Figure 4: Evolution of mean and standard deviation of CD with iterations

a good measure for the overall spread. When the spread is becoming better and better this means that the mean of the spread is rising. At the same time if the solution points become about equidistant it means that the standard deviation falls as can be seen in Figure 4. Therefore a relative measure of the mean to the standard deviation is a good measure for convergence. It has been chosen to be 2.7.

2 Impact of parameters on algorithm

In our implementation, some standard parameters (P,NC,σ_{mut},σ_{rec}) change over the course of the iterations and depend on a new set of parameters. These will first be explained, then the impact of changing these parameters will be shown and some conclusions will be drawn. The choice of parameters is done for ZDT6.

2.1 New Parameters

A parameter Q has been introduced which falls off exponentially over the course of a run, depending on the number of the current iteration it. This influences the parameters such as the recombination chance P, the number of children NC, the standard deviation of the Gauss curves for mutation and recombination, σ_{mut} and σ_{rec} respectively. The amount of parents is linearly dependent on the population size with constant α. The following relations have been implemented:

$$Q = Q_{dropoff}^{it-1} \tag{2}$$

$$P = 1 - Q \tag{3}$$

$$\sigma_{mut} = Q \cdot \sigma_{main} \tag{4}$$

$$\sigma_{rec} = \min(\sigma_{main}, \frac{\sigma_{main}}{20Q}) \tag{5}$$

$$NC = \text{round}(\frac{N}{2} + (2 \cdot N - \frac{N}{2}) \cdot Q^2) \tag{6}$$

$$NP = \frac{N}{\alpha} \tag{7}$$

2.2 Parameter Sweeps

The dependencies give following parameters to sweep: $Q_{dropoff}$, σ_{main}, N, α. Changing α did not change anything and was not included. The results of the sweeps can be seen in figure 5. Note that ideally one would want to sweep the parameters together to get an optimum but, due to high computing time, only one parameter is swept at a time. The chosen parameters are summarized in table 1.

2.2.1 Population size N

As can be seen in figures 5a and 5b, changing the population size does not have a significant influence on the number of iterations needed for convergence. However, the time for each iteration goes up, which has a much bigger impact, increasing the total time needed for the run significantly. We conclude that the only reason to take a larger population size is if you want to have more solution points to choose from at the end of the run.

Therefore a low value is chosen for N, being 30.

2.2.2 Iteration size control $Q_{dropoff}$

Changing the $Q_{dropoff}$ does not have a large effect on the number of iterations or time, but does make the resuts more consistent, as can be seen from the standard deviations in figures 5c and 5d.

Because of the increased consistency of computing time a high value of $Q_{dropoff}$ is chosen, being 0.993.

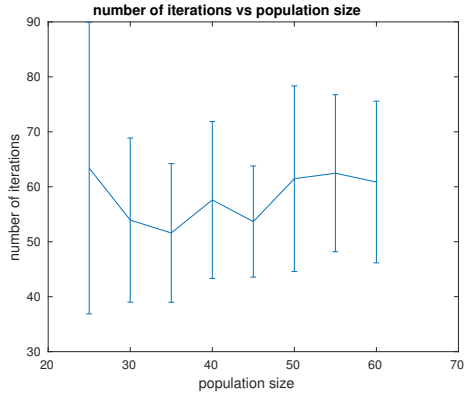
2.2.3 Randomness control σ_{main}

From figures 5e and 5f the only conclusion that can be drawn is that σ_{main} should not be chosen too low. This makes our mutations too local, resulting in too little exploration.

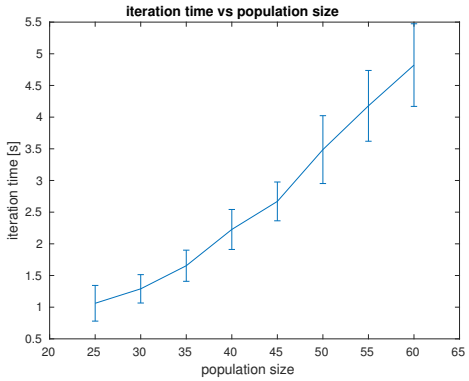
Therefore a rather high value is chosen for σ_{main} since it will drop off exponentially anyway with the factor Q.

parameter	ZDT6	Diff. pair
population size N	30	30
iteration control $Q_{dropoff}$	0.993	0.991
parent control α	2	2
randomness control σ_{main}	0.38	0.2

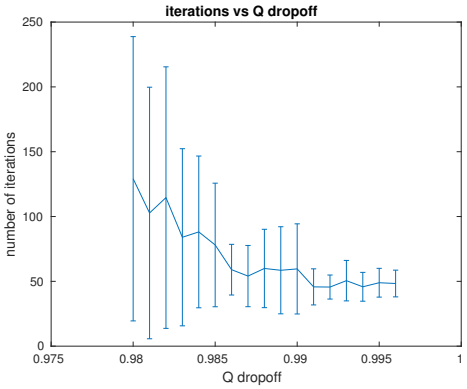
Table 1: Chosen parameters for solving ZDT6 and differential pair



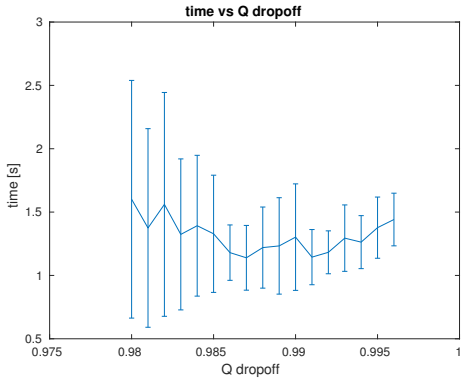
(a) Number of iterations vs population size



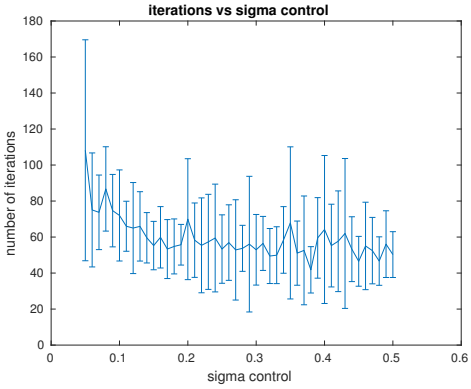
(b) Time to convergence vs population size



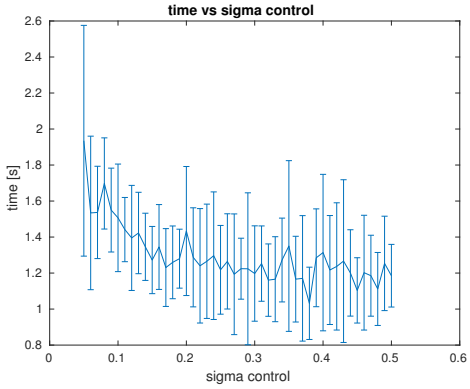
(c) Number of iterations vs $Q_{dropoff}$



(d) Time to convergence vs $Q_{dropoff}$



(e) Number of iterations vs σ_{main}



(f) Time to convergence vs σ_{main}

Figure 5: Results of parameter sweeping, each value for each parameter has been run 15 times, mean and one standard deviation shown

3 Final results for differential pair

The objectives in the optimization of the differential pair are minimal power usage (P) and maximal gain-bandwith product (GBW). In order to maximize GBW, we minimize $-GBW$. Final results are shown in Figure 6.

3.1 Analytical approximation

The GBW can be calculated as follows:

$$BW = \frac{1}{2\pi R_{out} C_L} \tag{8}$$

$$Gain = 2gm_1 R_{out} \tag{9}$$

$$GBW = \frac{gm_1}{\pi C_L} \tag{10}$$

Since the transistors of the differential pair do not have a good gain in sub-threshold conduction, the bounds for the offset voltage v_{off} are set in such a way that the transistors will operate in strong inversion while ensuring saturation operation. A small-signal differential input ac voltage with an amplitude of $20mV$ is applied. Then the transconductance g_m is given by the following equation [1]:

$$gm = \frac{2(\frac{I_{bias}}{2})}{V_{GS} - V_T} \tag{11}$$

Power is calculated as the product of bias current and supply voltage:

$$P = I_{bias} V_{supply} \tag{12}$$

Combining previous equations, the following relation between power consumption and GBW can be derived:

$$P = \pi C_L (V_{GS} - V_T) V_{supply} GBW \tag{13}$$

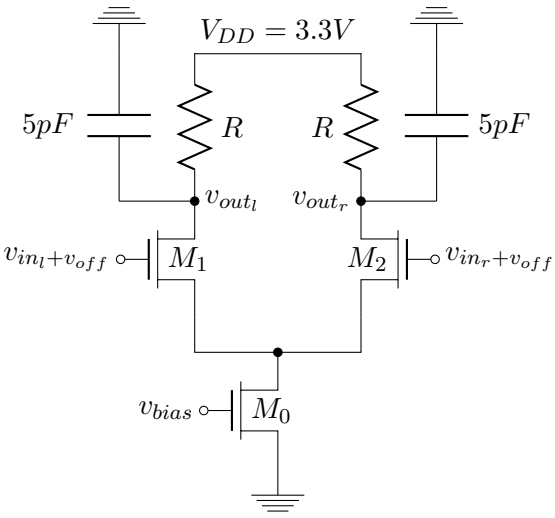
Note that this relation is only valid (and more or less linear) in small-signal operation, whereas the genetic algorithm will also encounter large-signal solutions. It can be seen in figures 6c and 6d that there is indeed a seemingly linear relation for low power and low GBW.

References

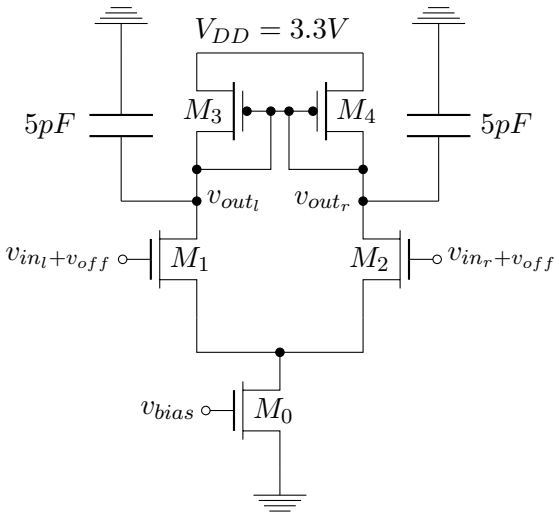
[1] P.Reynaert. *Design of Electronic Circuits*. Course at KU Leuven, 2014-2015.

[2] A.Seshadri *A fast elitist multiobjective genetic algorithm: NSGA-II*

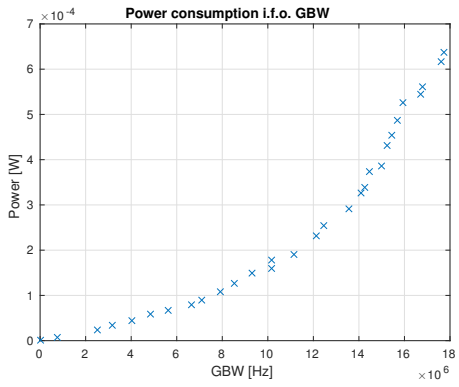
[3] J.Dessalles: Emergence in complex systems. Athens course at Telecom ParisTech, march 2015.
`http://icc.enst.fr/ECS/ECS.html`



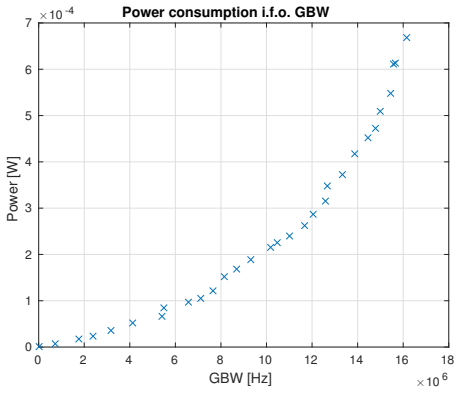
(a) Resistively loaded differential pair



(b) Diode loaded differential pair



(c) Resistive load, 69 iterations



(d) Diode load, 73 iterations

Figure 6: Circuits and results