Project Report

Paper: Glow-worm swarm optimization algorithm for solving multi-objective optimization problem

Team : Shooting Stars

Members: Shreya Gupta (201531123), Naila Fatima (201530154), Naga Sunethra Vysyaraju (201530072)

Introduction:

A multi-objective optimization problem (MOP) aims at simultaneously optimizing a number of objective functions under certain constraints. MOPs are used in several applications in engineering, economics and other areas which require a decision to be taken between conflicting objectives. MOPs aim to to attain an optimal trade-off between all the objectives involved in order to maximize the performance. An MOP can be mathematically represented as

$$min\ (f_1(x), f_2(x), .. f_k(x))$$

such that **x** belongs to the set of decision vectors

MOPs are used in order to optimize the power consumed by electric systems, to choose monetary policies and to choose a portfolio between expected returns and risk.

In an MOP problem, there are n decision variables, m objective functions and k constraints.

Definition 1:

If a multi-objective optimization problem(MOP) consists of n decision variables, m objective functions and k constraints.

Then the optimal objective is

$$max\ y = f(x) = [f_1(x), f_2(x), f_3(x) ...., f_M(x)]$$

$$s.t.$$

$$g_i(x) <= 0, \quad i = 1,2,...,p$$

$$h_i(x) = 0, \quad i = 1,2,...,q$$

Where $x = (x_1, x_2, ..., x_n) \in D$ is a decision variable.

It should be observed that the **f** terms are objective functions which are to be optimized and the conditions involving **g** and **h** are the constraints.

Compared to single objective optimization problems,there are improvements of some objectives. The solution of multi-objective optimization problem is the set of many **Pareto optimal** solutions.

Definition 2:

1. Pareto dominate: The solution x^0 dominates x^1 ($x^0 \succ x^1$) if and only if

$$f_i(x^0) >= f_i(x^1),\ i = 1,2,3,...M$$
$$f_i(x^0) > f_i(x^1),\ \exists\ i \in \{1,2,...,M\}$$

2. Pareto optimal: The solution x^0 is Pareto optimal, if and only if

$$\neg \exists\ x^1 \succ x^0$$

Glow-worm swarm algorithm is good at solving continuous optimization problem.
Initially, each glowworm is randomly assigned to the feasible domain of the objective function, which has luciferin and the dynamic vision range called local-decision range $r^i_d(t)$, which is bounded above by a circular sensor range r_s ($0 < r^i_d(t) <= r_s$).
The position of the glowworm decides their luciferin value and the value of the objective function is higher if the position is better.
Every glowworm finds a neighbor set in which the brighter neighbor has more attraction and can attract the glowworm to move towards it. These neighbor sets are separately found in their own local-decision range.So, by choosing different neighbor everytime, we can change the direction of the flight of the glowworm (direction in which the glowworm moves).
When the density of neighbors is lower, the radius of the local-decision range will be increased in order to find more neighbors and when the density is higher, the radius will be reduced. We vary the radius according to the density as we require a certain amount of glowworms. So, glowworms bound to gather in the position of the glowworms having high luciferin value.

The four phases of GSO (Glowworm Swarm Optimization):
-The deployment of glowworms/initialization.
-Updating luciferin
-Movement
-Updating the local-decision range.

Pseudo Code of the algorithm is:

1. Begin
2. Randomly select values for n glow worms with same initial luciferin value l_0 and the same initial dynamic decision domain value r_0 in the feasible domain of the objective function.
3. while(t<=iter_max1 | ε_1<the error given)
   {
   for each glowworm i do:
   $l_i(t) = (1-\varrho)*l_i(t-1)+\gamma*J(x_i(t))$

   for each glowworm i do: $\Sigma_{i=1}^{n} X_i * Y_i$
   {
   $N_i(t) = \{ j:||x_j(t) - x_i(t)|| < r^i_d(t); l_i(t) < l_j(t) \}$ % find the set of neighborhood
   for each neighbourhood $j \epsilon N_i(t)$
   {

$$p_{ij}(t)=\frac{l_j(t)-l_i(t)}{\sum_{k\in N_i(t)}(l_k(t)-l_i(t))}$$

}

j = max(p_i') % select the direction of movement

if $N_i(t)$ is empty and t<D_num

update the location

else

$X_i(t+1)=X_i(t) + s*(\frac{Xj(t)-Xi(t)}{||Xj(t)-Xi(t)||})$ % update the position

end

$r^i_d(t+1) = \min\{ r_s, \max\{0, r^i_d(t), \beta(n_t - |N_i(t)|)\}\}$ %update the dynamic decision

}

t = t+1

Compute the coordinates of the extreme points and save them;

}

END

As we can observe, we are taking a random initialization of the decision variables. In our problem, the decision variables are the width and length (W,L) of the transistors. If we have n transistors, each glowworm vector will have 2n values corresponding to the W and L values. We then compute the luciferin value for each glowworm. The luciferin value has been initialized by counting the number of decision variables which are less than their initial values.

We then have to iterate the outlined procedure until either the maximum number of iterations is reached or until the error becomes less than a threshold. For each glowworm, we have to update the luciferin value associated with it by using the equation:

$l_i(t) = (1-\varrho)*l_i(t-1)+\gamma*J(x_i(t))$

$l_i(t)$ is the updated luciferin value and $l_i(t-1)$ is the luciferin value of the glowworm in the previous iteration. ϱ and γ are constants which have been set to 0.4 and 0.6 respectively. $J(x_i(t))$ is the cost function for the particular set of decision variables. We have used the sum of leakages to be our cost function.

For each glowworm, we then compute a neighbour set by finding neighbouring glowworms which are similar to them (the difference in decision variables is less than a threshold). The luciferin value of the neighbouring glowworm should also be less than that of its own. Since our decision variables are W,L , we find glowworms which have W,L values such that the difference

in these values is less than a threshold. Within the neighbour set, we find the probability p_{ij} for each pair. The probability formula has been shown above and it utilized the luciferin values of a glowworm and its neighbour. The maximum probability will give us the direction of movement and we update our location according to the update rule.
We then have to update our decision range value (r_d) by using the equation above. It should be observed that this value determines how much of the neighbourhood we are searching.
In this algorithm, all the non-inferior individuals of populations are given higher value of luciferin, and the dominated individuals have lower value of luciferin. The lower one will move toward the higher one.

The fitness function of the MOP-GSO:

Incorporating a fitness function while solving the MOP enhances the performance. A fitness function is used in order to decide if an individual is good or bad. For our fitness values, we use the rank of the individuals computed from the non-inferior sorting of all individuals.
It should be observed that a fitness function is used to modify the position updating formula. Using this step increases the convergence speed as well as the accuracy of the solution. The modified position updating formula is given below.

The movement of the MOP-GSO algorithm:

In order to increase population diversity and the convergence speed, a disturbance term is added to the location updating formula.

$$X_i(t+1) = X_i(t) + s((x_j(t)-x_i(t))/|x_j(t)-x_i(t)|)+u$$

$u = \alpha * randn * (iter\_max-t) / iter\_max$

s is step

randn is the random number that meets normal distribution.

α is the parameter relevant to the problem and set in advance and it ranges from 0 to 1.

α is generally set to 0.001

The algorithm is able to explore new areas in early stages which causes it to have a greater ability to search the local optimum in later stage improving the convergence speed and accuracy of the solution.

In this paper, to verify the performance of MOP-GSO algorithm,it is compared with NSGA2 and also proving that MOP-GSO is effective.

PVT Variations

We performed PVT variations in order to find the values of W and L which give optimal leakage values and which satisfy the delay constraints. The process variations were performed by varying the value of Vdd by +/- 10% (0.8V +/- 10%), varying the temperature between -55 degC to 125 degC, and +/- 10% variations in certain the process parameters. The process parameters which were varied include toxe, toxp, toxref, ndep, xj, lint and wint. These

parameters are present in the .pm files and are changed for both the nmos and pmos transistors. For PVT variations we choose random values for Vdd, temperature and the process parameters whi++ch were within the range specified. This technique may however not give W and L values which are invariant in the worst possible case.

In order to compute the W and L values for the worst case scenario, we found the PVT parameters for this case and computed the W and L values by using our GSO algorithm.

Glowworm swarm optimisation algorithm is implemented on below circuits:

Values used in the implementation:
ρ(luciferin decay constant)=0.4 $n_t = 4$
γ(luciferin enhancement constant)=0.6 s(step size)=0.0003 β=0.08

1) Inverter

An inverter consists of 2 Mosfets(1 nMOS and 1 pMOS). Our problem is to minimize the leakage(objective function) of the circuit under delay constraints with length and width as the decision variables.
We find the optimal values of W_n,L_n,W_p,L_p for which the objective function(summation of all leakages) is minimized under delay constraints.
Each glow worm is a vector of decision variables (W_n,L_n,W_p,L_p) of length 4 since we have 2 Mosfets.
There are 2 leakages(objective function) associated with input (0 and 1) for inverter and 2 delays(constraints) corresponding to t_{LH} and t_{HL} for inverter.
So in MOP-GSO algorithm, summation of leakages is decreased under the delay constraints, by iteratively changing the values of glowworm vector.

Implementation:
No. of glowworms = 20
No. of iterations = 200

Results obtained are:

1) Leakage Power obtained for W=132nm, L=22nm for inverter :

Leakage with input=0 and Leakage with input=1

2) Leakage power obtained after GSO :

Leakage with input=0 Leakage with input=1

```
3.3239900000000107e-07,2.5087399999999739e-07
```

3) W, L values for Nmos and Pmos:

W_p L_p W_n L_n

```
4.008323420524078529e+02,2.368653754668181932e+01,2.606058115890103863e+02,2.21659331718008244e+01
```

4) Delay for Not gate with W=132nm and L=22nm for inverter:

t_{LH} t_{HL}

```
1.9691267329999999919e-11,4.0931133440000000180e-12
```

5) Delay for not gate with W and L:

t_{LH} t_{HL}

```
5.5614220759999999899e-12,2.7132400379999999863e-12
```

2) Full Adder

A full adder consists of 28 Mosfets of which 14 are pMos and 14 are nMos. We aim to find the W,L values for these transistors which will give us the minimum leakage values satisfying the delay constraints. Since there are 28 transistors, there will be 56 values in each glowworm vector (as 2 values - W and L- for each transistors). We try to update the decision variables (W, L) by using the update rule and moving in the direction of the highest probability. We find the optimal values of W,L which enables us to minimize the objective function (summation of all leakages) which satisfies the delay constraints.

Implementation:
No. of glowworms = 20
No. of iterations = 200

Results obtained are:

Leakage Power for W = 132 nm and L = 22 nm for Full Adder:

```
1.95616e-06 2.01212e-06 1.65038e-06 1.64571e-06 1.62007e-06 1.45173e-06 1.60244e
-06 1.38308e-06
```

Leakage Power after GSO algorithm without PVT:

```
1.48312e-06 1.89614e-06 1.344789e-06 1.38042e-06 1.45383e-06 1.28261e-06 1.35172
e-06 1.21464e-06
```

Leakage Power after GSO algorithm with PVT:

```
1.53646e-06 1.93008e-06 1.45389e-06 1.44832e-06 1.49643e-06 1.31412e-06 1.51008e
-06 1.28539e-06
```

W and L of Pmos and Nmos:

```
8.833510644415796378e+02,2.420339204778296249e+01,8.789406012445797387e+02,2.299097665526128509e+01,4.410665786618338871e
+02,2.325165526350523848e+01,3.231789950728847316e+02,2.413895362001888856e+01,1.222337117134243840e+02,2.311402604042439890e
+01,7.422005863173574198e+02,2.250606219469495883e+01,2.268846936829447429e+02,2.323277598270782818e+01,1.790268744906422853e
+02,2.346022457537732819e+01,1.311572069743538691e+02,2.212805183405250986e+01,7.945094321026637090e+02,2.326336093250668924e
+01,3.411506726328486820e+02,2.299491641331088232e+01,5.516315244897162984e+02,2.359181730024552692e+01,4.097306634055514110e
+02,2.248563494414592157e+01,2.930379902305021460e+02,2.443988559953571382e+01,8.858394155493923563e+02,2.377770087819284228e
+01,8.842323199076622586e+02,2.410815625381655281e+01,1.589851175133522929e+02,2.288863963633572496e+01,6.662692822942763087e
+02,2.406304115935289190e+01,6.397819994370678387e+01,2.398011994507694311e+01,4.101457850295307708e+02,2.267893250281303708e
+01,2.045812062066795249e+02,2.410307878332676168e+01,8.200788674183500007e+02,2.467380804428015040e+01,2.689539294628973494e
+02,2.425705158246563187e+01,7.896884245391601098e+02,2.225428063833316727e+01,9.378169232005971026e+02,2.309424806495996663e
+01,1.557382093209184859e+02,2.300949833854479820e+01,8.919753599322697823e+01,2.284992320912850161e+01,2.363687915256722363e
+02,2.274085945201801451e+01
```

Worst Case Leakages before GSO algorithm:

```
1.21194e-04 2.08716e-04 2.28242e-04 2.76781e-04 2.38174e-04 2.75682e-04 2.74240e-04 2.83183e-04
```

Worst Case Leakages after GSO algorithm iterating until leakage is 40% less than the initial value:

```
3.45986e-05 1.17118e-04 1.25339e-04 1.83921e-04 1.48078e-04 1.62173e-04 1.80526e-04 1.91825e-04
```