Available online at www.sciencedirect.com

ScienceDirect

Mathematics and Computers in Simulation 79 (2008) 1137–1152

MATHEMATICS AND COMPUTERS IN SIMULATION

www.elsevier.com/locate/matcom

Electronic design automation using a unified optimization framework

Yiming Li[a,*], Shao-Ming Yu[b], Yih-Lang Li[b]

[a] *Department of Communication Engineering, National Chiao Tung University, Hsinchu, Taiwan*
[b] *Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan*

Received 7 September 2007; received in revised form 2 October 2007; accepted 2 November 2007
Available online 17 November 2007

Abstract

This work proposes an object-oriented unified optimization framework (UOF) for general problem optimization. Based on biological inspired techniques, numerical deterministic methods, and C++ objective design, the UOF itself has significant potential to perform optimization operations on various problems. The UOF provides basic interfaces to define a general problem and generic solver, enabling these two different research fields to be bridged. The components of the UOF can be separated into problem and solver components. These two parts work independently allowing high-level code to be reused, and rapidly adapted to new problems and solvers. The UOF is customized to deal with several optimization problems. The first experiment involves a well-known discrete combinational problem, wihle the second one studies the robustness for the reverse modeling problem, which is in high demanded by device manufacturing companies. Additionally, experiments are undertaken to determine the capability of the proposed methods in both analog and digital circuit design automation. The final experiment designs antenna for rapidly growing wireless communication. Most experiments are categorized as simulation-based optimization tasks in the microelectronics industry. The results confirm that UOF has excellent flexibility and extensibility to solve these problems successfully. The developed open-source project is publicly available.

Keywords: Biological inspired techniques; Deterministic method; Design automation; Simulation-based optimization

1. Introduction

A major obstacle in designing a general-purpose optimizer in any one programming language is the difficulty in defining a comprehensive interface for problems and solvers. Appropriate interfaces for problems and solvers produce objects that are heuristically easy to reuse again in a specific framework. Several optimization frameworks, such as GALIB [31], DESMO [5], and NP-Opt [25] have been proposed in the public domain. These packages have certain advantages, but still have room for improvement in terms of the connections between problem and solver for real-world applications. This work implements a C++ unified optimization framework (UOF) for general problems and solvers. Benefitting from C++ design pattern techniques, UOF allows users to define the problems and solvers from the fundamental level. The developed UOF consists of two parts, the problem part and the solver part. The basic interface of problem-related classes contains initialization, evaluation, and constraint procedures. The solver-related

* Corresponding author at: P.O. Box 25-178, Hsinchu 300, Taiwan.
E-mail address: ymli@faculty.nctu.edu.tw (Y. Li).

classes include the solver, solution, termination, and information procedures. Separating each part into several classes increases the efficiency of reutilization.

This work address several problems to demonstrate the flexibility and extensibility of UOF, particularly in the electronic design automation field. The first experiment considers the famous traveling salesman problem (TSP), which is a NP problem, and the other experiments which are quite difficult problems in the fabrication of semiconductor devices and electromagnetic fields. The modern microelectronics industry involves many optimization jobs that are subject to simulated results from certain simulators, such as device model parameter extraction [18,19,22], automatic design of integrated circuit (IC) [12], process simulation [7], reverse modeling [1], and antenna optimization [14,24]. These terms need to feed specific parameters into some commercial simulator to obtain the corresponding simulated results. The engineers adjust the parameters based on the results, and again feed the adjusted parameters to retrieve the improved results. This circular procedure continues until the results match the requirements of the customers or supervisors. Currently, this mechanical procedure is typically performed by engineers with expertise. Therefore, a well-defined framework can assist engineers to complete their work easily with various optimization techniques. In particular, the proposed UOF contains biological inspired techniques which can been applied to solve optimization problems without requiring much empirical knowledge.

The rest of this paper is organized as follows. Section 2 introduces the architecture of the UOF. Section 3 then explains the detailed coding methodologies for problem and solver, and provides pseudocode for building problems and solvers. Section 4 discusses the achieved results. Conclusions are finally drawn in Section 5, along with recommendations for future research.

2. The unified optimization framework

Fig. 1 illustrates the class diagram of UOF. The class UOFId is the base-class of every class here for run-time type information (RTTI) purposes. All members in UOF can be easily categorized into problem-related and solver-related classes. The UOFProblem class is the main class to define the problem; the UOFInitializer class is responsible for the initialization of the solution for each problem; the UOFEvaluator class provides the method for evaluating the result obtained by the UOFProblem derived classes; the UOFConstraint class defines the constraint of each parameter; the UOFSolver class takes the major character in solver relative category; the UOFSolution class stores possible solutions and several operators of the specified solver; the UOFTerminator class judges when and how to stop the optimization process of the solver, and the UOFInfo class logs the behavior of the solver during the solving process. All classes in both categories are abstract classes, and need to be implemented in derived classes.

Fig. 2 shows the built-in class specializations of the most important two classes in current UOF. As shown in Fig. 2 a, NPProblem, FunctionProblem, and ExtSimProblem are directly inherited from UOFProblem. The UOFProblem class has general interfaces for the user to define any problems. NPProblem provides convenient interfaces for defining NP-hard problems such as TSP and the single/parallel machine scheduling problem. Moreover, optimization problems of linear and non-linear continuous functions, such as DeJong functions [11], can be defined by FunctionProblem class. ExtSimProblem class has basic linkages to external software to perform simulations, such as commercial TCADs and ECADs. The solver-related classes illustrated in Fig. 2 b can be separated into two categories, namely gradient-based and the population-based solvers. The gradient-based solver inherits directly from the UOFSolver class, and includes the Broyden–Fletcher–Goldfarb–Shanno (BFGS) [2] and Levenberg–Marquardt (LM) methods [29]. Both of these are quasi-Newton methods and are popularly used in deterministic solving packages [26]. The population-based solver

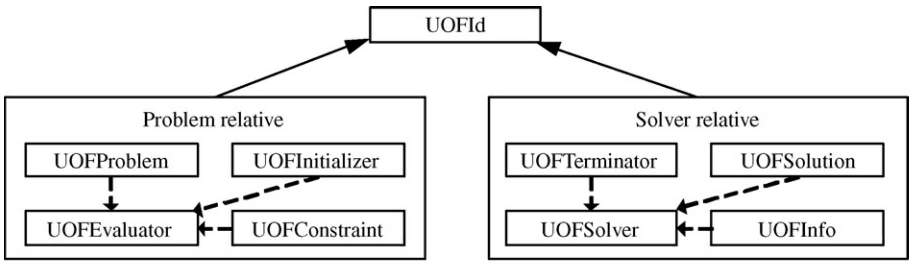

Fig. 1. An illustration of the architecture of UOF. Solid lines indicate inheritance and dashed lines indicate the ownership.

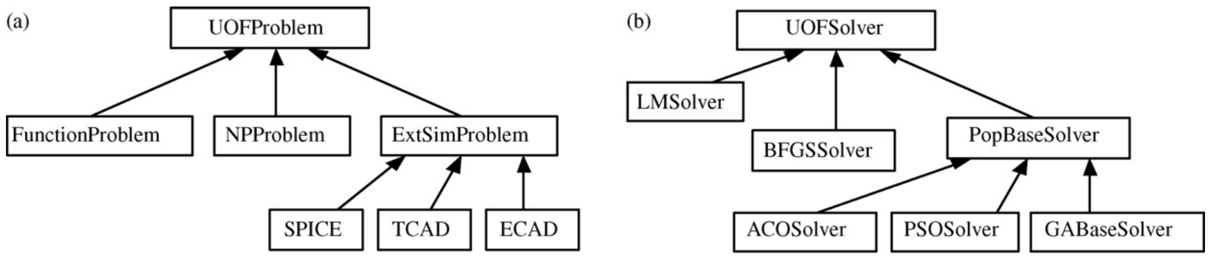

Fig. 2. The class hierarchy of the UOFProblem and UOFSolver. The solid lines indicate inheritance.

includes several heuristic methods such as GABaseSolver for genetic algorithm (GA) [8], PSOSolver for particle swarm optimization (PSO) [13], ACOSolver for ant colony optimization (ACO) [3] and SASolver for simulated annealing method [10,15]. The fundamental abstract classes are described in detail below.

2.1. *The UOFProblem class*

The interface of UOFProblem class is defined as follows:

```
class UOFProblem : public UOFId
{
public:
        virtual double GetResult(void *inp)=0;
        virtual void GetResult(void *inp,void *out)=0;
        virtual void GetJacobianResult(void *inp,void *out)=0;
        virtual void GetExternalData(void *data)=0;
        virtual void SetConstraint(void *cnt)=0;
}
```

The resulting function is the major function of UOFProblem class to calculate the results of given parameters. The function has two overloaded versions. One is a convenient version which returns a result in a ‘double’ format, and the other is the type-free version that returns data in any specified type. The GetJacobianResult function is used by the gradient-based solvers, which return the Jacobian matrix back to the solver.

2.2. *The UOFEvaluator class*

The UOFEvaluator interface is declared as follows:

```
class UOFEvaluator : public UOFId
{
public:
    UOFEvaluator(UOFProblem & src) : m_pProblem(& src){}
    virtual double operator()(void* src) = 0;
    UOFProblem *m_pProblem;
}
```

The UOFEvaluator itself is a function object [4] that acts like a function pointer in C, but takes advantages of C++ object-oriented programming. It stores the pointer of UOFProblem, and performs the communications with UOFProblem in the operator( ) function. UOFEvaluator is like an agent that connects the UOFProblem and UOFSolver to reduce the coupling effects between UOFProblem and UOFSolver.

2.3. *The UOFInitializer class*

The interface of UOFInitializer:

```
class UOFInitializer
{
public:
      virtual void operator()(void* src, UOFProblem* pT);
}
```

UOFInitializer performs solution initialization, which is called by UOFSolution object. The solution to be initialized should be transferred to void* type. Moreover, UOFInitializer also stores the UOFProblem pointer, and allows the user to retrieve extra information in UOFProblem object.

2.4. *The UOFSolver class*

The interface of UOFSolver is defined as follows:

```
class UOFSolver : public UOFId
{
public:
      virtual void Initialization()=0;
      virtual void Solve()=0;
      virtual bool Configuration(const char *filename);
}
```

In this class, the Initialization function performs the initialization steps before the solving procedure. Configuration function allows the user to configure the solver through a script file, and the Solve function controls the entire optimization process.

2.5. *The UOFSolution class*

UOFSolution is an abstract class to describe the solution format used in any UOF components. The class declaration is listed as below:

```
class UOFSolution : public UOFId
{
public:
      UOFSolution(UOFInitializer *init, UOFEvaluator *eval);
      UOFSolution(const UOFSolution&);
      UOFSolution& operator=(const UOFSolution&);
      virtual UOFSolution* clone() const;
      virtual void copy(const UOFSolution&);
      virtual double score();
}
```

The score function in UOFSolution invokes UOFEvaluator to obtain the simulation results of given parameters, and calculates the fitness score.

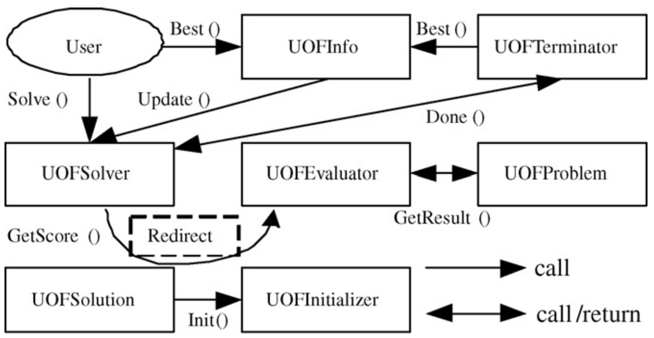

Fig. 3. The working flow and object inner connections in UOF from users' point of view.

2.6. *The working flow of the UOF*

Fig. 3 shows the working flow and object inner connections in the UOF from the user point of view. Users initially call the Solve function in the UOFSolver class to start the optimization operation. Meanwhile, the UOFInitializer initializes the UOFSolution objects. The Solve function invokes the GetScore function in UOFSolution class to calculate the results from the given parameters. The UOFSolution class redirects and passes the GetScore message to the UOFEvaluator class. The UOFEvaluator object passes the solution to UOFProblem, and calls GetResult to obtain the corresponding results. While the solver solves the problem, in each iteration the UOFInfo updates the current status of the solver, such as the best parameter set. Moreover, the UOFTeriminator also updates the current best solution from UOFInfo, and tells UOFSolver when to stop the optimization procedure. Finally, users can obtain the best solution from the UOFInfo object.

3. Implementation examples

This section constructs of simulation-based problems and two solvers step by step given to demonstrate the capability and extensibility of UOF.

3.1. *Simulation-based problem*

The major task in implementing a simulation-based problem in UOF is to encapsulate the I/O operations to external simulators within the subclass of the UOFProblem class. This section describes the specialization of UOFProblem class the specialization of UOFProblem class without loss of generality, taking the TCAD reverse modeling problem as an example. Three classes, TCADProblem, TCADEvaluator and TCADInitializer, are derived. TCADInitializer is responsible for generating a random initial parameter set. TCADEvaluator measures the error between the target data and the simulated result. TCADProblem provides the GetResult, GenerateIntermediateFiles, and ReadOutput-Files functions. Fig. 4 a shows the working flow of these methods. The Figure indicates that the GetResult function is rather simple. GetResult first calls the GenerateIntermediateFiles function to generate the input files required by the simulator, then runs the simulator, and finally calls the ReadOutputFiles to retrieve the simulated result. GenerateIntermediateFiles function generates the input files required by the simulator. These files are called m_InterMediateFiles. This procedure requires the m_InputFile data, and maps the value of each parameter into the m_InputFile to generate the m_InterMediateFiles. The ReadOutputFiles function parses the output file of the simulator to retrieve the necessary simulated result. Fig. 4 b shows the flowchart of TCADEvaluator. TCADEvaluator calls GetResult in TCADProblem to retrieve the simulated result, and returns the error between the simulated result and the target data to the caller.

3.2. *Genetic algorithm solver*

This section describes the construction of a genetic algorithm solver. The first step is to declare three classes named GABaseSolver, TerminateUponIteration and GA1DArraySolution, which are inherited from PopBaseSolver, UOFTerminator and PopSolution, respectively. In this experiment, the stopping criteria was set to reach the maximum

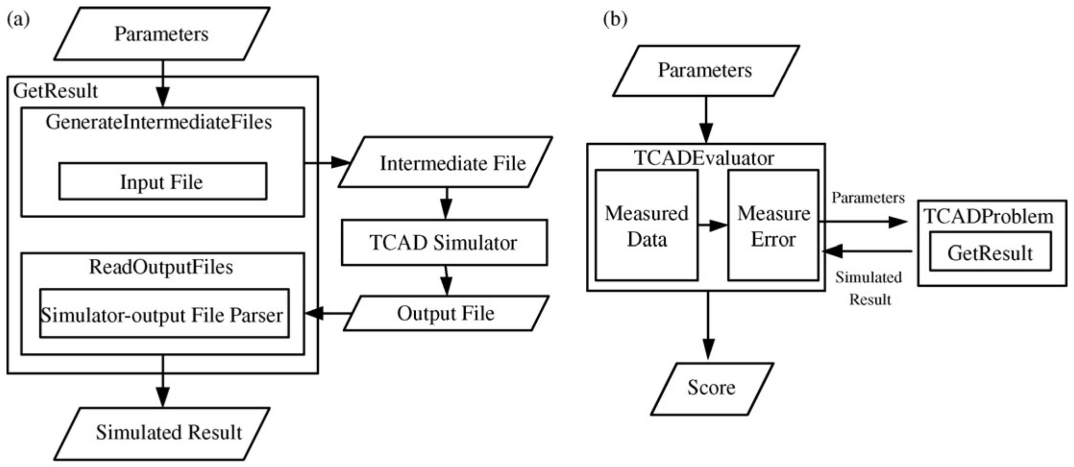

Fig. 4. (a) The working flow of GetResult in TCADProblem and (b) the flowchart of TCADEvaluator.

iteration, therefore, so the TerminateUponIteration function would return true if the maximum iteration was reached. The GA1DArraySolution class rewrites the GetScore function, and declares the GA crossover and mutation operator. The Solve function is the entry point of the solver called by the main function of the program. It first calls the Initialization, then performs Evolve to evaluate each individual and perform further genetic operators, namely selection, crossover, and mutation until TerminateUponIteration (in Done function) reports the "true" event. The pseudocode for Solve and Evolve functions are as follows:

```
void GABaseSolver::Solve()
{
    Initialization();
    do{
      Evolve();
    }while(!Done());
}
void GABaseSolver::Evolve()
{

1.Compute fitness score of each parameter set
  For each solution (in m_Pop array):
  Call m_Pop[a]→ score();
2.Sort the fitness score:
  sort(m_Pop.begin(),m_Pop.end(),MinScore);
3.Selection operation:
  (*m_pSlct)(m_Pop, selected);
4.Crossover operation:
  (*m_pCros)(selected[p1], selected[p2], m_Pop[a], m_Pop[b]);
5.Mutation operation:
  (*m_pMutr)(m_Pop[a]);
}
```

3.3. Particle swarm optimization solver

The PSO, like the GA, is a population-based solver, meaning that the content are alike. The major difference between these two solvers occurs in the Evolve function. The PSOSolver performs a move operation defined in the PSOMove function of the PSO1DArraySolution to update the parameters after sorting the individuals. The pseudocode of the Evolve function is given below.

```
void GABaseSolver::Evolve()
{
    1.Compute fitness score of each parameter set
      For each solution (in m_Pop array):
      Call m_Pop[a]→ score();
    2.Sort the fitness score:
      sort(m_Pop.begin(),m_Pop.end(),MinScore);
    3. Move particles:
      (*m_pMove)(m_Pop[0], m_Pop[i], m_Vc, m_K1, m_K2);
}
void PSOMove::operator()
( PopSolution* bestp, PopSolution* currentp, double w, double k1, double k2)
{
    1.Convert bestp and currentp to PSO1DArraySolution type
    2.Compute the velocity of currentp;
    3.Update the position of currentp;
}
```

4. Results and discussion

This section presents the experimental results. The first experiment involved a traditional TSP; the second experiment involved a reverse modeling problem of semiconductor device, the third and fourth experiment involved circuit specification problems of low noise amplifier (LNA) and static random access memory (SRAM). The final experiment involved an antenna design problem for wireless communication.

4.1. The traveling salesman problem

The TSP is a well-known discrete combinational problem. Its objective is to discover a minimized route for a salesman to visit each city only once and finally return to the starting point. Three classes, the TSPProblem, TSPEvaluator,

Table 1
The optimized result of the TSP

Method	Geometry	Number of cities	Average number of iterations	Percent of success run
GA	Circle	30	2231	80
ACO	Circle	30	4	80
GA	Circle	50	6562	30
ACO	Circle	50	18	60
GA	Matrix	9	24	100
ACO	Matrix	9	4	100
GA	Matrix	25	3600	30
ACO	Matrix	25	26	100

TSPInitializer, which are inherited from UOFProblem, UOFEvaluator, UOFInitializer, respectively, are defined to implement TSP in the UOF standard. The TSPInitializer is utilized to create a random string as a route to travel to each city. The TSPEvaluator simply returns the total length of the given route, which is derived from TSPProblem. Pseudocode for TSPInitializer, TSPEvaluator, and GetResult function in TSPProblem is given below.

```
class TSPInitializer : public UOFInitializer
{
    public:
      virtual void operator()(void* src, UOFProblem* pProblem)
      {
        1.Random generates the initial solution S
        2.Set S to src
      }
}
class TSPEval : public UOFEvaluator
{
    public:
      virtual double operator()(void* src)
      {
        1.Get total length of the route from TSPProblem:
        TotalLength = m_pProblem → GetResult(src);
        2.Return TotalLength
      }
}
double TSPProblem::GetResult(void* param)
{
    1.Convert param into proper data type
    2.Compute total_distance from parm
    3.Return total_distance
}
```

Four geometric distributions of cities were explored. The first and second distributions formed a circle, and contained 30 and 50 cities, respectively. The third and fourth distributions were 3 × 3 and 5 × 5 matrices, respectively. The GA and ACO were performed to compare the efficiency of each problem, as shown in Table 1. The success run means that the run can find optimal solution, and the low percent of success run for some cases with GA methods is due to the limitation of iteration number. In our examinations, the number of iteration is limited to 10,000. Table 1 summarizes the result, revealing that the ACO method is better than GA method for solving this problem. Fig. 5(a) shows the layout of the 5 × 5 matrix, and Fig. 5(b) shows the optimal solution of this layout. The result also confirms that the UOF can solve the discrete combinational optimization problem.

4.2. *The device reverse modeling problem*

TCAD simulation is widely used for the analysis of semiconductor devices [9,16,17,20,23,28]. Discovering the associated optimal configurations for a set of *I–V* curves of a given device forms an inverse problem [1]. This problem involves process and device simulations, and has sensitive parameters. Due to the above features, the problem currently plays an important role in technology development, as well as the performance diagnosis of nanoscale CMOS devices. As discussed in Section 3.1, the GetResult function needs to be defined in the TCADProblem class in order to generate input file for the TCAD simulator, and receive results after the simulation. The following shows the pseudocode to

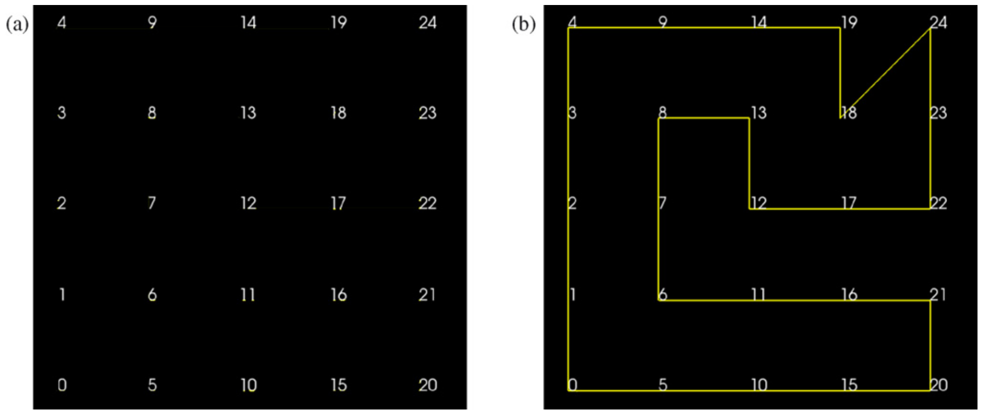

Fig. 5. (a) A layout of the examination case of the TSP and (b) the optimal solutions for this layout.

invoke the external simulators, which can be used to generate a TCAD problem in the UOF.

```
void TCADProblem::GetResult(void* in, void* out)
{
    1.GenerateIntermediateFiles(in);
    2.Execute the command to run the simulator;
    3.ReadOutputFiles(out);
}
void TCADProblem::GenerateIntermediateFiles(void *parameter)
  {
      1.Convert parameter into proper data type
      2.Generate the execution command from parameter
  }
  void TCADProblem::ReadOutputFiles(void* result)
  {
      For each output file
        Read simulated result from the file and store into result
  }
```

This TCAD problem encodes the parameters as a real number string for evolution, and adopts the GA method for optimization. The simulator outputs a set of drain current for each candidate, and obtains the score of each candidate by measuring the error between the measured drain current and the simulated ones. The reverse modeling problem of sub-65 nm N- and P-MOSFET [9,16,17,20,23] is described here. The inset of Fig. 6 shows a 2D cross-section view of the simulated 65 nm MOSFET with an LDD doping profile. Fig. 6 shows the target *I–V* curves to be optimized, and several most concerned physical quantities empirically. Fig. 7 displays the extracted curves for both N- and P-MOSFETs under various device and process configurations. The result confirms the robustness of UOF for this reverse modeling problem.

4.3. The low-noise amplifier optimization

In this experiment, UOF was adopted the hybrid optimization method for the LNA IC design [6,30]. First, the GA searches the entire problem space. The candidates that GA searched are passed to the circuit simulator to retrieve the results of circuit simulation during this period. According to the specified desired targets of S_{11}, S_{12}, S_{21}, S_{22},

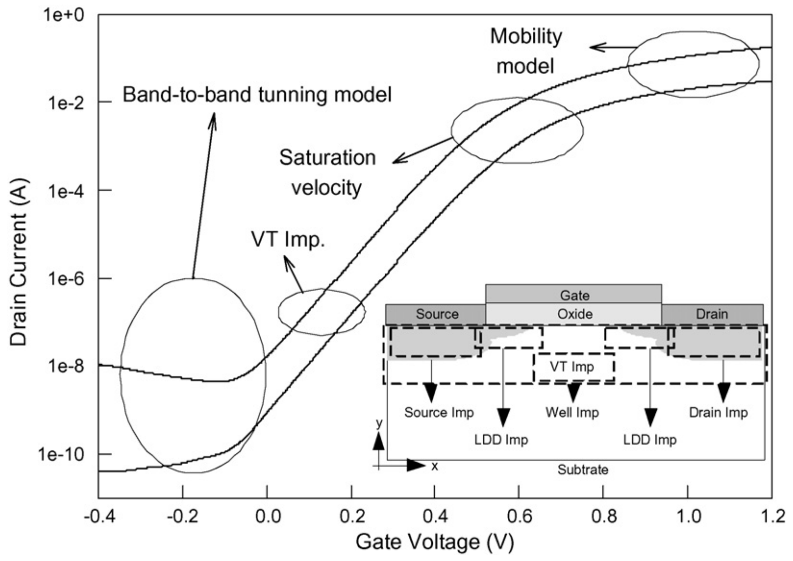

Fig. 6. An illustration of the target *I–V* curves to be extracted and empirical knowledge. The important sections are pointed out in circles. The inset plot is a cross-section view of the simulated MOSFET with LDD doping profile.

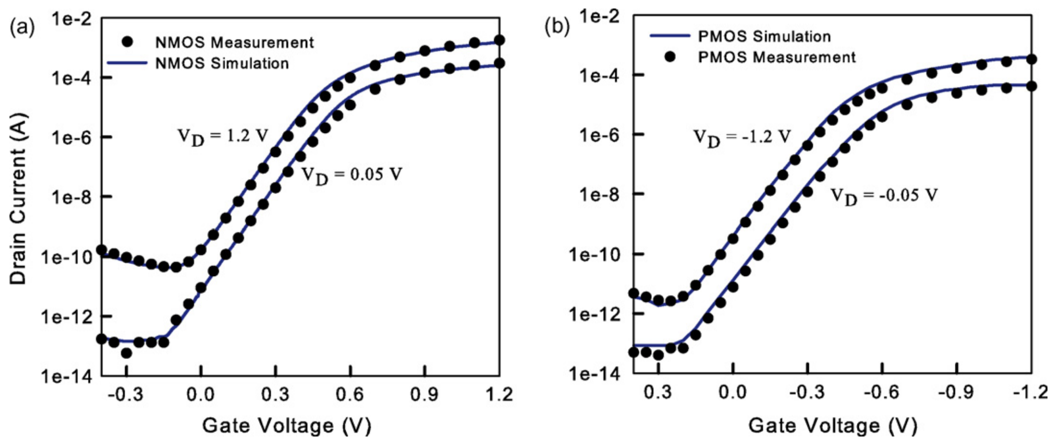

Fig. 7. The achieved accuracy of the extracted *I–V* curves for N- and P-MOSFETs, where gate length $L = 65$ nm and device width $W = 1$ μm.

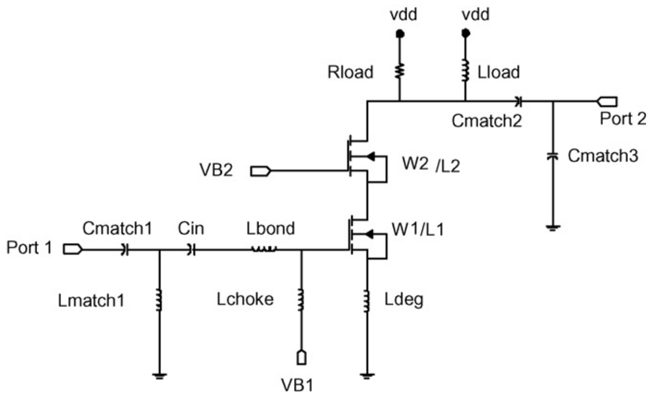

Fig. 8. The explored LNA circuit in our experiment.

K factor, the noise figure, and the input third-order intercept point, the simulated results are then passed to the evaluation to measure the fitness score. The score is evaluated for the global optimization of GA. The LM method then simultaneously makes local searches after obtaining an approximate solution, and sets the local optimum as the initial values for the GA to perform further optimization. The pseudocode for the LNA problem is given below.

```
class LNAEval : public UOFEvaluator
{
    public:
      virtual double operator()(void* src)
      {
        1. Get specified results from LNAProblem:
        Results = m_pProblem $\rightarrow$ GetResult(src);
        2. Calculate score according to the Results and Targets;
        3. Return score;
      }
}
double LNAProblem::GetResult(void* parameter)
{
    1.Generate netlist by the GenerateIntermediateFiles(parameter);
    2.Run the external circuit simulation;
    3.Extract the S parameter, K factor, the noise figure,
      and the input third-order intercept point;
    4.Return the extracted results;
}
```

The LNA IC studied herein, which is shown in Fig. 8, focuses on the working frequencies ranging from 2.11 to 2.17 GHz. The LNA circuit has two cascaded transistors. In this investigation, the compact model BSIM 3v3 was adopted for the 0.18 μm MOSFETs. More than 15 parameters need to be extracted in the designed 0.18 μm MOSFET LNA IC. The stopping criterion of the optimization is the minimization of errors between the extracted outputs and the specified target. Table 2 shows the optimized parameters of the investigated experiment. Table 3 shows the specified targets and extracted results are listed in the demonstrating that the extracted parameters can satisfy all the specified targets.

4.4. *The static random access memory optimization*

Considering the static noise margin (SNM) of six- and four-transistors (6T and 4T) SRAM cells with 65 nm CMOS devices [21,27], the device's channel length and supply voltage were also successfully tuned to meet the specified target of SNM. Fig. 9 shows the explored 4T and 6T SRAM cells. The circuit of 6T SRAM is a flip-flop comprising

Table 2
A list of the range of designing parameters and the optimized parameters for the LNA circuit

Element	Unit	Range	Result
Cmatch1	F	300–800	657.738f
Cmatch2	F	1–10	4.505p
Cmatch3	F	1–10	4.951p
Lbond	H	1–10	1.058n
Ldeg	H	0.1–5	1.155n
Lmatch1	H	1–10	5.257n
VB1	V	0.5–1.5	0.69V
VB2	V	0.5–5	1.96V
L	H	0.13–0.3	0.25u

Table 3
A list of the specified targets and the extracted results for the LNA circuit

Specification	Target	Result
S_{11} (dB)	< -10	-35.1
S_{22} (dB)	< -10	-19.1
S_{12} (dB)	< -25	-38.3
S_{21} (dB)	As large as possible	11.3
K	> 1	11.1
NF	< 2	1.17
IIP3	> -10	0.3

two cross-coupled inverters and two access transistors, M3 and M6. The flip-flop consists of two load elements (M5, M4), called pull-up (load) transistors and two storage elements (M2, M1), called pull-down (driver) transistors. Data are stored as voltage levels with the two sides of the flip-flop in opposite voltage configurations. Pseudocode for the SRAMEval and GetResult functions in SRAMProblem are given below.

```
class SRAMEval : public UOFEvaluator
{
    public:
      virtual double operator()(void* src)
      {
        1. Get static noise margin (SNM) from SRAMProblem:
        SNM = m_pProblem → GetResult(src);
        2. Return SNM;
      }
}
double SRAMProblem::GetResult(void* parameter)
{
    1.Generate netlist by the GenerateIntermediateFiles(parameter);
    2.Run the external simulator;
    3.Extract SNM from the simulation results;
    4.Return SNM;
}
```

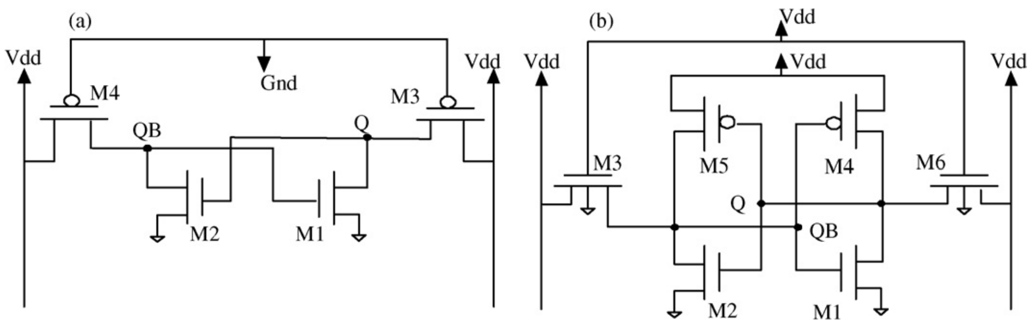

Fig. 9. The explored (a) 4T and (b) 6T SRAM cells.

In this experiment, the targets of SNM was set as 190 and 173 mV for the 4T and 6T SRAM cells, respectively. The SNM is the minimum DC-voltage disturbance necessary to upset the state of the SRAM cell. It is quantified by the length of the side of the largest square that can fit inside the butterfly curves formed by the cross-coupled inverters. The cell stability is based on the ability of the cell to resist accidental overwrites during different operating conditions in the presence of electrical noise and process variations. The factors that influence the cell stability include the device size, supply voltage and temperature. In this work, the device's channel length and supply voltage were adjusted to fit our targets. Table 4 summarizes the range of the circuit parameters and extracted results.

4.5. The antenna design problem

In this experiment, the antenna design problem was studied. The major drawback of this problem is that the geometry of the antenna affect the performance is well-known; however, the effect of the shape on performance is unclear. Fine-tuning the shape of an antenna designer involves much guesswork. Based on our experience and technique of optical proximity correction [32], this method appends or subtracts several small rectangles along the edges of the antenna to improve the return loss. The applied simulator solves the Maxwell equation, which is an important electromagnetic issue. The following is the pseudocode for the antenna problem.

```
class AntennaInitializer : public UOFInitializer
{
    public:

    virtual void operator()(void* src, UOFProblem* pProblem)
    {
      1. Partition the antenna into small segments;
      2. Encode the shift of all segments as parameter S and set S to src;
    }
}
class AntennaEval : public UOFEvaluator
{
    public:
      virtual double operator()(void* src)
      {
        1. Get the return loss curve (S11) from AntennaProblem:
        S11 = m_pProblem → GetResult(src);
        2. Calculate score according to the S11 of modified geometry;
        3. Return score;
      }
}
double AntennaProblem::GetResult(void* parameter)
{
    1.Generate new antenna geometry by the parameter;
    2.Run the external simulator;
    3.Extract return loss curve (S11) form the simulation;
    4.Return S11;
}
```

Table 4
A list of the range of designing parameters and the optimized parameters for the 4T and 6T SRAM cells

Factors (parameters)	Range	Result (6T)	Result (4T)
L1: channel length of transistor M1 (nm)	60–70	68.3	67.8
L2: channel length of transistor M2 (nm)	60–70	62.2	65.9
L3: channel length for other transistors (nm)	60–70	69.1	63.2
Vdd: supply voltage (V)	1.08–1.38	1.35	1.22

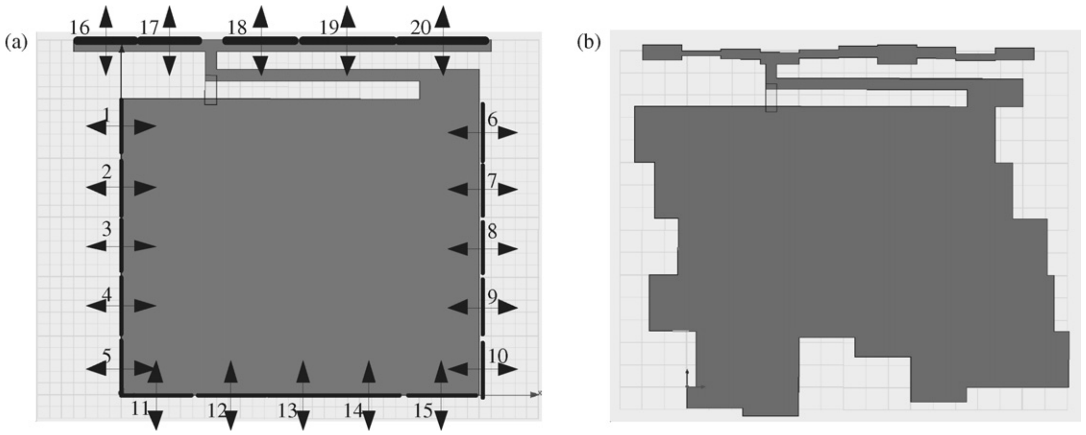

Fig. 10. (a) The original geometry and (b) the optimized geometry of examined antenna.

The parameters of this antenna design problem are the shift offsets of segments shown in Fig. 10 a, encoded as a real number string for evolution. Each segment from 1 to 15 can shift inward or outward within a specified range, while the value of parameter 16–20 indicates the thickness of the top line. The simulator outputs a return loss curve, and the score of each candidate is obtained by measuring the distance between the theoretical and the simulated curves within the working frequency. Figs. 10 a and 11 a presents the original shape of the examined antenna and the return loss. The working frequencies were set to 2.6 and 5 GHz. The frequency 2.6 GHz is the “S-band” used for mobile broadcasting, and the frequency 5 GHz is used by the 802.11a WLAN protocol. Fig. 10 b shows the optimized shape, and Fig. 11 b illustrates the corresponding result. The return losses at 2.6 and 5 GHz in Fig. 11 a are both lowered to 20dB. This finding indicates that this antenna easily fits the target.

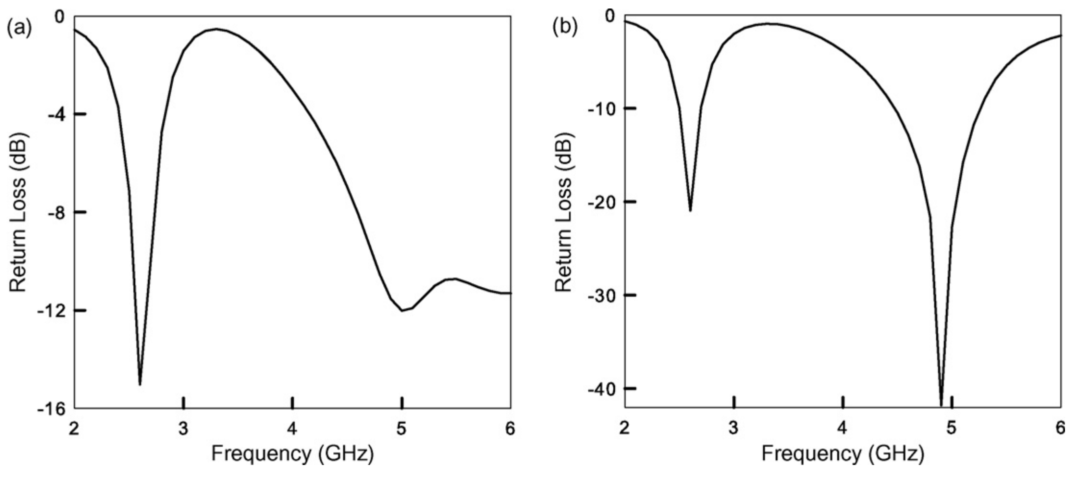

Fig. 11. The return loss (S11) of (a) the original geometry and (b) the optimized geometry.

5. Conclusions

This work presents a proposal of an object-oriented UOF for general problem optimization. The UOF is based on biological inspired techniques, numerical deterministic methods and C++ objective design, and has significant potential to perform optimization operations on various problems. The UOF bridges the gap between problems and solvers. This work successfully customizes UOF to deal with the following electronic applications: the TCAD reverse modeling problem; antenna shape optimization, and the specifications of LNA and SRAM circuits. Experimental results confirm capability of the proposed UOF. We believe that the proposed UOF benefit electronic design automation, and significantly help for engineers to complete their optimization jobs. The developed open-source project is available in the public domain (http://140.113.87.143/ymlab/uof/ and http://sourceforge.net/projects/uof/).

Acknowledgments

This work was supported in part by the National Science Council of TAIWAN under Contract NSC-95-2221-E-009-336, Contract NSC-95-2752-E-009-003-PAE, Contract NSC-96-2221-E-009-210, Contract NSC-96-2752-E-009-003-PAE, and by the MoE ATU Program, Taiwan, under a 2006-2007 grant.

References

[1] T. Binder, C. Heitzinger, S. Selberherr, A study on global and local optimization techniques for tcad analysis tasks, IEEE Trans. Comput. Aided Design 23 (2004) 814–822.
[2] R.H. Byrd, P. Lu, J. Nocedal, C. Zhu, A limited memory algorithm for bound constrained optimization, SIAM J. Sci. Comput. 16 (1995) 1190–1208.
[3] M. Dorigo, L.M. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem, IEEE Trans. Evol. Comput. 1 (1997) 53–66.
[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994.
[5] B. Gehlsen, B. Page, A framework for distributed simulation optimization, in: Proceedings of the 2001 Winter Sim. Conference, 2001, pp. 508–514.
[6] R. Gupta, D.J. Allstot, Parasitic-aware design and optimization of cmos rf integrated circuits, in: Proceedings of the IEEE International Microwave Symposium, 1998, pp. 1867–1870.
[7] J. Herrmann, B. Conaghan, L. Henn-Lecordier, P. Mellacheruvu, M.-Q. Nguyen, G. Rubloff, R. Shi, Understanding the impact of equipment and process changes with a heterogeneous semiconductor manufacturing simulation environment, in: Proceedings of the Winter Sim. Conference, 2000, pp. 1491–1498.
[8] J. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.
[9] K.-Y. Huang, Y. Li, C.-P. Lee, A time domain approach to simulation and characterization of rf hbt two-tone intermodulation distortion, IEEE Trans. Microwave Theor. Technol. 51 (2003) 2055–2062.
[10] M. Ishikawa, T. Toya, M. Hoshida, K. Nitta, A. Ogiwara, M. Kanehisa, Multiple sequence alignment by parallel simulated annealing, Comput. Appl. Biosci. 9 (1993) 267–273.
[11] K.A.D. Jong, An Analysis of Behavior of a Class of Genetic Adaptive Systems, Doctoral Dissertation, University of Michigan, Dissertation Abstracts International, 1975.
[12] S.-J. Jou, K.-F. Liu, C. Su, Circuits design optimization using symbolic approach, in: Proceedings of the IEEE International Symposium on Circuits System, 1995, pp. 1396–1399.
[13] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks, 1995, pp. 1942–1948.
[14] A.J. Kerkhoff, R.L. Rogers, H. Ling, Design and analysis of planar monopole antennas using a genetic algorithm approach, IEEE Trans. Antenn. Propag. 52 (2004) 2709–2718.
[15] J. Kim, S. Pramanik, M.J. Chung, Multiple sequence alignment using simulated annealing, Comput. Appl. Biosci. 10 (1994) 419–426.
[16] Y. Li, A parallel monotone iterative method for the numerical solution of multidimensional semiconductor Poisson equation, Comput. Phys. Commun. 153 (2003) 359–372.
[17] Y. Li, T.-S. Chao, S.M. Sze, A novel parallel approach for quantum effect simulation in semiconductor devices, Int. J. Model. Simul. 23 (2003) 94–102.
[18] Y. Li, Y.-Y. Cho, Intelligent bsim4 model parameter extraction for sub-100 nm mosfet era, Jpn. J. Appl. Phys. 43 (2004) 1717–1722.
[19] Y. Li, Y.-Y. Cho, C.-S. Wang, K.-Y. Huang, A genetic algorithm approach to ingap/gaas hbt parameters extraction and rf characterization, Jpn. J. Appl. Phys. 42 (2003) 2371–2374.
[20] Y. Li, H.-M. Chou, J.-W. Lee, Investigation of electrical characteristics on surrounding-gate and omega-shaped-gate nanowire finfets, IEEE Trans. Nanotechnol. 4 (2005) 510–516.

[21] Y. Li, C.-S. Lu, Characteristic comparison of sram cells with 20 nm planar mosfet, omega finfet and nanowire finfet, in: Proceedings of the IEEE Nanotechnical Conference, 2006, pp. 339–342.

[22] Y. Li, C.-T. Sun, C.-K. Chen, A floating-point based evolutionary algorithm for model parameters extraction and optimization in hbt device simulation, in: L. Rutkowski, J. Kacprzyk (Eds.), Advances in Soft Computing—Neural Networks and Soft Computing, Physica-Verlag, 2003, pp. 364–369.

[23] Y. Li, S.-M. Yu, A parallel adaptive finite volume method for nanoscale double-gate mosfets simulation, J. Comput. Appl. Math. 175 (2005) 87–99.

[24] W.-C. Liu, Design of a multiband cpw-fed monopole antenna using a particle swarm optimization approach, IEEE Trans. Antenn. Propag. 53 (2005) 3273–3279.

[25] A. Mendes, P. França, P. Moscato, Np-opt: an optimization framework for np problems, in: Proceedings of the IV SIMPOI/POMS, 2001, pp. 82–89.

[26] P. Mendes, D. Kell, Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation, Bioinformatics 14 (1998) 869–883.

[27] E. Seevinck, F. List, J. Lohstroh, Static-noise margin analysis of mos sram cells, IEEE J. Solid-State Circ. 22 (1987) 748–754.

[28] S. Selberherr, Analysis and Simulation of Semiconductor Devices, Springer-Verlag, New York, 1984.

[29] S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in C, Cambridge University Press, 1988, pp. 542–549.

[30] P. Vancorenland, G.V. der Plas, M. Steyaert, G. Gielen, W. Sansen, A layout-aware synthesis methodology for rf circuits, in: Proceedings of the IEEE/ACM International Conference on CAD, 2001, pp. 358–362.

[31] M. Wall, Galib a C++ Library of Genetic Algorithm Components, Mass. Inst. of Technol., Cambridge, 2000, pp. 339–342.

[32] S.-M. Yu, Y. Li, A pattern-based domain partition approach to parallel optical proximity correction in vlsi designs, in: Proceedings of the 19th IEEE International Parall. and Dist. Proce. Symposium.