Federal University of Amazonas (UFAM)
Postgraduate Program in Electrical Engineering (PPGEE)

OptCE: A Counterexample-Guided Inductive Optimization Solver

Higo Albuquerque, Rodrigo Araújo, Iury Bessa,
Lucas Cordeiro, and Eddie Lima

Mikhail Ramalho
mikhail.ramalho-gadelha@soton.ac.uk

Agenda

- Inductive Optimization Based on Counterexamples
- Illustrative Example
- CEGIO Algorithms
- OptCE Tool
- Experimental Evaluaion

General Steps from Inductive Optimization Based on Counterexamples

- **Modeling** – In the modeling step, the optimization problem is defined for a cost function and then its constraints are introduced
- **Specification** – This step consists of describing the behavior of the system and properties to be verified. A C code is generated with ESBMC functions to restrict the state space
- **Verification** - This step performs the verification of the C code, and informs if it has found a global optimization

Inductive Optimization Based on Counterexamples

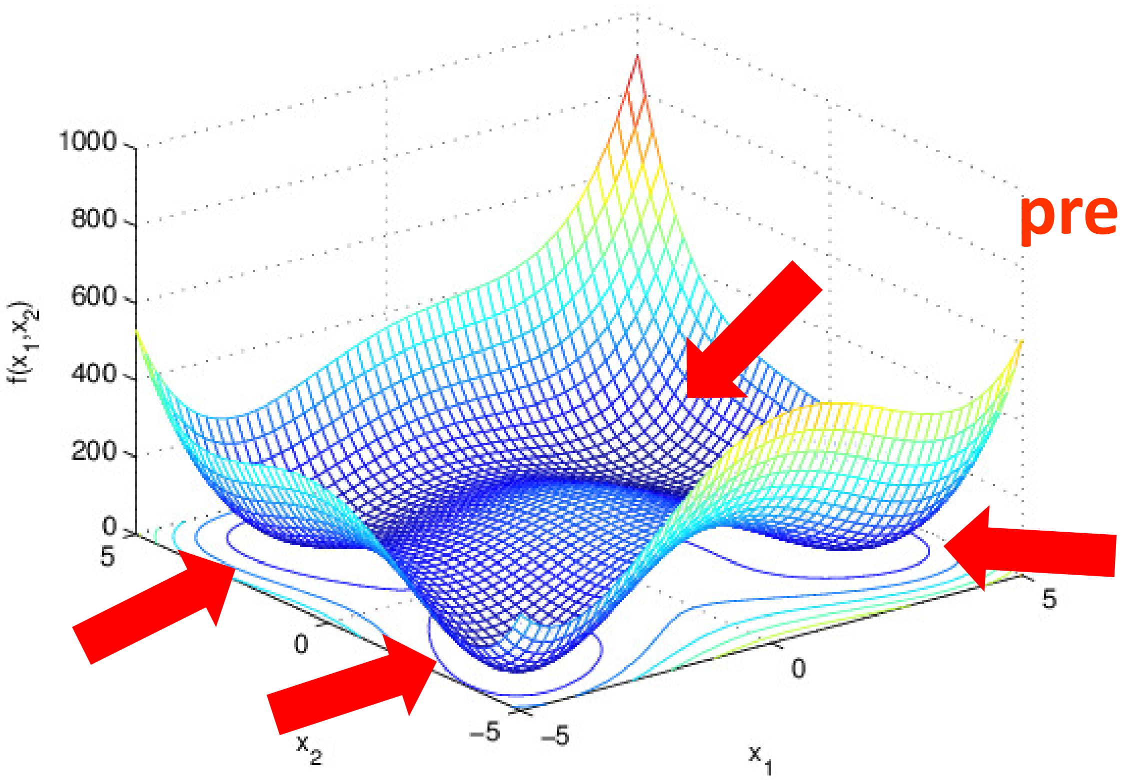
$X \subset \mathbb{R}^n$

- Given a cost function $f: X \to \mathbb{R}$, such that is the space of decision variables and $\Omega \subset X$, where Ω is the set of constraints
- A multivariate optimization problem consists of finding the vector of optimal decision variables X^*, which minimizes *f* considering Ω

$$\min f(x) \quad s.t. \ \Omega$$

- The problem will be non-convex, *if and only if f(x)* is a non-convex function

Inductive Optimization Based on Counterexamples



Function of Himmelblau presents four global minima

$$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

Inductive Optimization Based on Counterexamples

- To extend the verifier to solve an optimization problem, two code directives are used: **ASSUME** and **ASSERT**

- ***ASSUME*** is responsible for defining the constraints over the non-deterministic variables, from which the verifier restricts the state space

- ***ASSERT*** is used to define the property to be verified and return "True" or "False" for the optimization check

Inductive Optimization Based on Counterexamples

- The decision variables of the problem are defined as non-deterministic integers

- An integer variable controls the accuracy and discretization of the state space

$$p = 10^n$$

- where *n* is the number of decimal places of the decision variables

Inductive Optimization Based on Counterexamples

- Successive verification are executed iteratively increasing the precision, to converge to the optimal solution

- Each verification run checks the following property:

$$l_{\text{optimum}} \Leftrightarrow f(x) > f_c$$

Illustrative Example

- Given the following optimization problem:

$$\min_{x_1, x_2} f(x_1, x_2) = (x_1{}^2 + x_2 - 11)^2 + (x_1 + x_2{}^2 - 7)^2$$

$$s.t. \qquad \begin{array}{r} -5 \leq x_1 \leq 0 \\ 0 \leq x_2 \leq 5 \end{array}$$

- Minimize the Himmelblau function

Illustrative Example

```
int nondet_int();
int main(){
    int p = 1;
    float fc = 100;
    int X1 = nondet_int();
    int X2 = nondet_int();
    float x1, x2, fobj;
    __ESBMC_assume((X1>=-5*p) && (X1<=0*p));
    __ESBMC_assume((X2>=0*p) && (X2<=5*p));
    x1 = (float) X1/p;
    x2 = (float) X2/p;
    fobj = (x1*x1+x2-11)*(x1*x1+x2-11)+(x1+x2*x2-7)*(x1+x2*x2-7);
    __ESBMC_assume( fobj < fc );
    __ESBMC_assert( fobj > fc, “” );
    return 0;
}
```

Illustrative Example

```
int nondet_int();
int main(){
    int p = 1;
    float fc = 100;
    int X1 = nondet_int();
    int X2 = nondet_int();
    float x1, x2, fobj;
   __ESBMC_assume((X1>=-5*p) && (X1<=0*p));
   __ESBMC_assume((X2>=0*p) && (X2<=5*p));
    x1 = (float) X1/p;
    x2 = (float) X2/p;
    fobj = (x1*x1+x2-11)*(x1*x1+x2-11)+(x1+x2*x2-7)*(x1+x2*x2-7);
    __ESBMC_assume( fobj < fc );
   __ESBMC_assert( fobj > fc, “” );
    return 0;
}
```

The precision variable starts as 10^0

Illustrative Example

```
int nondet_int();
int main(){
    int p = 1;
    float fc = 100;
    int X1 = nondet_int();
    int X2 = nondet_int();
    float x1, x2, fobj;
   __ESBMC_assume((X1>=-5*p) && (X1<=0*p));
   __ESBMC_assume((X2>=0*p) && (X2<=5*p));
    x1 = (float) X1/p;
    x2 = (float) X2/p;
    fobj = (x1*x1+x2-11)*(x1*x1+x2-11)+(x1+x2*x2-7)*(x1+x2*x2-7);
    __ESBMC_assume( fobj < fc );
   __ESBMC_assert( fobj > fc, “” );
    return 0;
}
```

Decision variables are declared as non-deterministic integers

Illustrative Example

```c
int nondet_int();
int main(){
    int p = 1;
    float fc = 100;
    int X1 = nondet_int();
    int X2 = nondet_int();
    float x1, x2, fobj;
    __ESBMC_assume((X1>=-5*p) && (X1<=0*p));
    __ESBMC_assume((X2>=0*p) && (X2<=5*p));
    x1 = (float) X1/p;
    x2 = (float) X2/p;
    fobj = (x1*x1+x2-11)*(x1*x1+x2-11)+(x1+x2*x2-7)*(x1+x2*x2-7);
    __ESBMC_assume( fobj < fc );
    __ESBMC_assert( fobj > fc, "" );
    return 0;
}
```

Statements of ASSUMEs are used to specify constraints and reduce state space

Illustrative Example

```
int nondet_int();
int main(){
    int p = 1;
    float fc = 100;
    int X1 = nondet_int();
    int X2 = nondet_int();
    float x1, x2, fobj;
    __ESBMC_assume((X1>=-5*p) && (X1<=0*p));
    __ESBMC_assume((X2>=0*p) && (X2<=5*p));
    x1 = (float) X1/p;
    x2 = (float) X2/p;
    fobj = (x1*x1+x2-11)*(x1*x1+x2-11)+(x1+x2*x2-7)*(x1+x2*x2-7);
    __ESBMC_assume( fobj < fc );
    __ESBMC_assert( fobj > fc, “” );
    return 0;
}
```

Property $l_{optimum}$ is verified

Illustrative Example

```c
int nondet_int();
int main(){
    int p = 1;
    float fc = 100;
    int X1 = nondet_int();
    int X2 = nondet_int();
    float x1, x2, fobj;
    __ESBMC_assume((X1>=-5*p) && (X1<=0*p));
    __ESBMC_assume((X2>=0*p) && (X2<=5*p));
    x1 = (float) X1/p;
    x2 = (float) X2/p;
    fobj = (x1*x1+x2-11)*(x1*x1+x2-11)+(x1+x2*x2-7)*(x1+x2*x2-7);
    __ESBMC_assume( fobj < fc );
    __ESBMC_assert( fobj > fc, "" );
    return 0;
}
```

When $l_{optimum}$ is false, $\exists f(x) < fc$, fc must be updated and verification repeated.

Illustrative Example

```c
int nondet_int();
int main(){
    int p = 1;
    float fc = 100;
    int X1 = nondet_int();
    int X2 = nondet_int();
    float x1, x2, fobj;
    __ESBMC_assume((X1>=-5*p) && (X1<=0*p));
    __ESBMC_assume((X2>=0*p) && (X2<=5*p));
    x1 = (float) X1/p;
    x2 = (float) X2/p;
    fobj = (x1*x1+x2-11)*(x1*x1+x2-11)+(x1+x2*x2-7)*(x1+x2*x2-7);
    __ESBMC_assume( fobj < fc );
    __ESBMC_assert( fobj > fc, "" );
    return 0;
}
```

If l_{optimum} is true, $\nexists\, f(x) < fc$, fc is the optimal value

CEGIO Algorithms

- **CEGIO-G** - Applies to general functions
- **CEGIO-F** - Applies to semi-definite and positive functions
- **CEGIO-S** - Applies to convex functions

CEGIO-G Algorithm

Data: A cost function $f(x)$, a set of constraints Ω, and a desired precision η.

Results: The optimal decision vector x^* and the optimal cost function $f(x^*)$.

Initialize $f(x^{(0)})$ randomly;
Initialize precision variables with $p = 1, i = 1$ e $k = \log p$;
Declare the decision variables $x^{(i)}$ as non-deterministic integer variables;
while $k \leq \eta$ **do**
  set the limits of x with the *ASSUME* directive, such that $x \in \Omega^{\mathbf{k}}$;
  describe the model for $f(x)$;
  do
    set constraint $f\left(x^{(i)}\right) < f(x^{(i-1)})$ as the ASSUM directive;
    check for satisfiability of l_{optimum} given by equation (slide 8) with the ASSERT directive;
    analysis $x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$ based on the counter-example;
    make $i = i + 1$;
  while l_{optimum} is satisfying;
  update the precision variable p, and consequently k;
end
$x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$;
return $x^* = x^{(i)}$;

CEGIO-G Algorithm

Data: A cost function $f(x)$, a set of constraints Ω, and a desired precision η.
Results: The optimal decision vector x^* and the optimal cost function $f(x^*)$.

Initialize $f(x^{(0)})$ randomly;
Initialize precision variables with $p = 1, i = 1$ e $k = \log p$;
Declare the decision variables $x^{(i)}$ as non-deterministic integer variables;
while $k \leq \eta$ **do**
    set the limits of x with the *ASSUME* directive, such that $x \in \Omega^{k}$;
    describe the model for $f(x)$;
    do
        set constraint $f\left(x^{(i)}\right) < f(x^{(i-1)})$ as the ASSUM directive;
        check for satisfiability of l_{optimum} given by equation (slide 8) with the ASSERT directive;
        analysis $x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$ based on the counter-example;
        make $i = i + 1$;
    while l_{optimum} is satisfying;
    update the precision variable p, and consequently k;
end
$x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$;
return $x^* = x^{(i)}$;

If l_{optimum} is satisfactory

CEGIO-G Algorithm

Data: A cost function $f(x)$, a set of constraints Ω, and a desired precision η.

Results: The optimal decision vector x^* and the optimal cost function $f(x^*)$.

```
Initialize f(x^(0)) randomly;
Initialize precision variables with p = 1, i = 1 e k = log p;
Declare the decision variables x^(i) as non-deterministic integer variables;
while k ≤ η do
    set the limits of x with the ASSUME directive, such that x ∈ Ω^k;
    describe the model for f(x);
    do
        set constraint f(x^(i)) < f(x^(i−1)) as the ASSUM directive;
        check for satisfiability of l_optimum given by equation (slide 8) with the ASSERT directive;
        analysis x* = x^(i) and f(x*) = f(x^(i)) based on the counter-example;
        make i = i + 1;
    while l_optimum is satisfying;
    update the precision variable p, and consequently k;
end
x* = x^(i) and f(x*) = f(x^(i));
return x* = x^(i);
```

Updates the restrictions based on the counter example

CEGIO-G Algorithm

Data: A cost function $f(x)$, a set of constraints Ω, and a desired precision η.
Results: The optimal decision vector x^* and the optimal cost function $f(x^*)$.

Initialize $f(x^{(0)})$ randomly;
Initialize precision variables with $p = 1, i = 1$ e $k = \log p$;
Declare the decision variables $x^{(i)}$ as non-deterministic integer variables;
while $k \leq \eta$ **do**
    set the limits of x with the *ASSUME* directive, such that $x \in \Omega^k$;
    describe the model for $f(x)$;
    do
        set constraint $f(x^{(i)}) < f(x^{(i-1)})$ as the ASSUM directive;
        check for satisfiability of $l_{optimum}$ given by equation (slide 8) with the ASSERT directive;
        analysis $x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$ based on the counter-example;
        make $i = i + 1$;
    while $l_{optimum}$ is satisfying;
    update the precision variable p, and consequently k;
end
$x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$;
return $x^* = x^{(i)}$;

If No, update the precision variable

CEGIO-G Algorithm

Data: A cost function $f(x)$, a set of constraints Ω, and a desired precision η.
Results: The optimal decision vector x^* and the optimal cost function $f(x^*)$.

```
Initialize f(x^(0)) randomly;
Initialize precision variables with p = 1, i = 1 e k = log p ;
Declare the decision variables x^(i) as non-deterministic integer variables;
while k ≤ η do
    set the limits of x with the ASSUME directive, such that x ∈ Ω^k;
    describe the model for f(x);
    do
        set constraint f(x^(i)) < f(x^(i−1)) as the ASSUM directive;
        check for satisfiability of l_optimum given by equation (slide 8) with the ASSERT directive;
        analysis x* = x^(i) and f(x*) = f(x^(i)) based on the counter-example;
        make i = i + 1;
    while l_optimum is satisfying;
    update the precision variable p, and consequently k;
end
x* = x^(i) and f(x*) = f(x^(i));
return x* = x^(i);
```

Repeat until desired accuracy is reached

Deviating from local minima

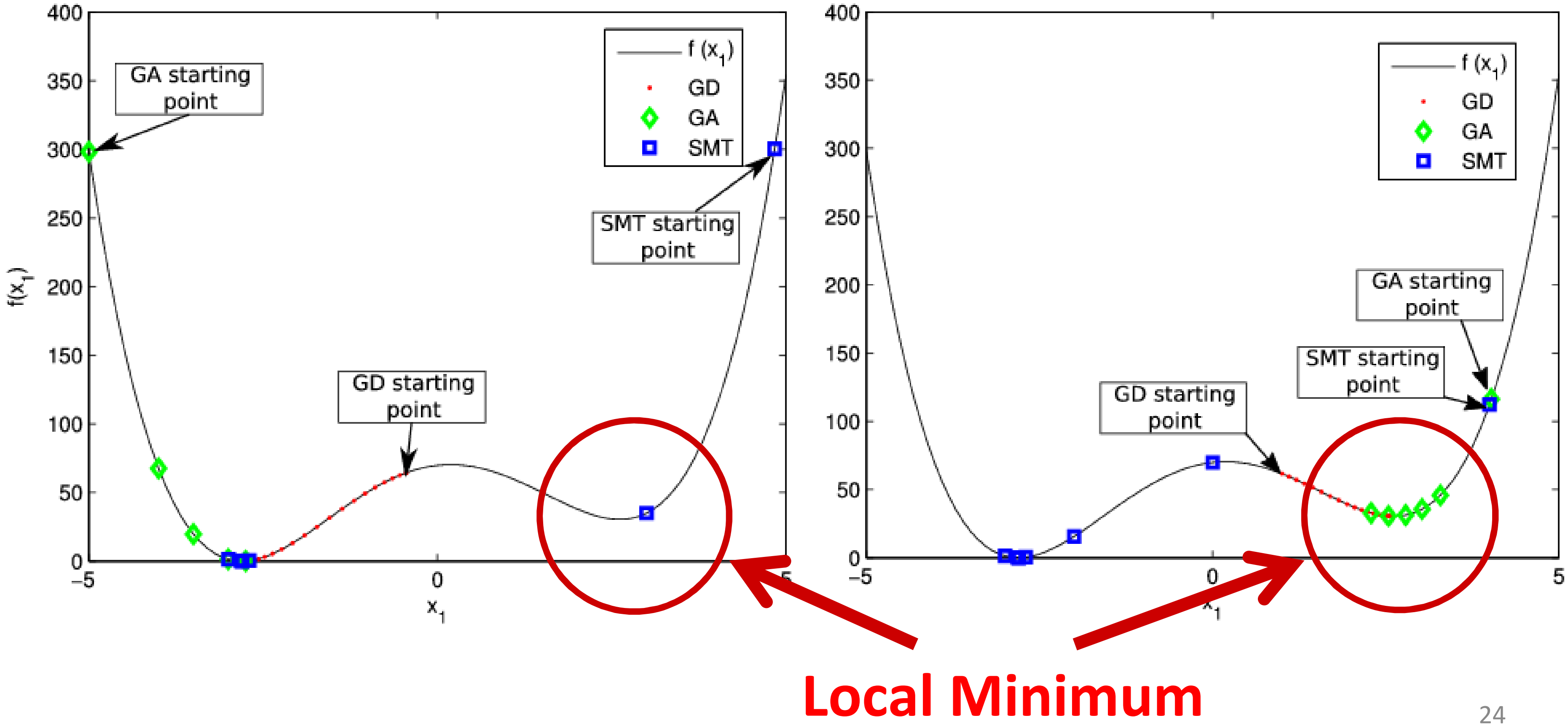
- We evaluated the performance of our methodology to minimize the Himmelblau function, and we compared with the downward gradient (GD) and genetic algorithm (GA) methods.

$$f(x_1, x_2) = (x_1{}^2 + x_2 - 11)^2 + (x_1 + x_2{}^2 - 7)^2$$

- The proposed methodology does not report minimum locations as in GD and GA, and it is able to find the global minimum

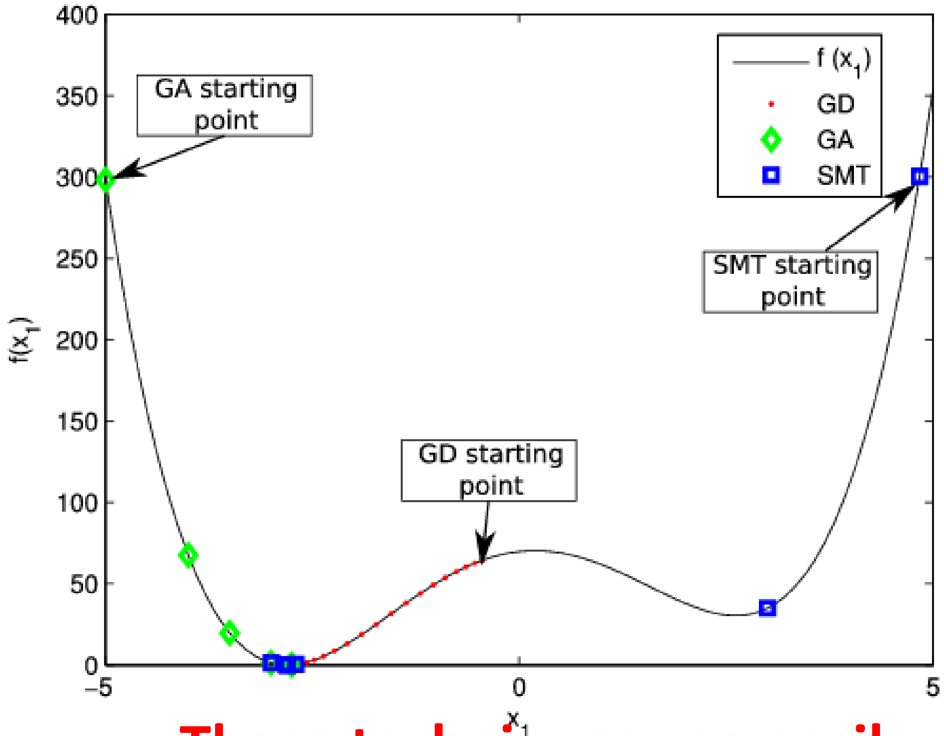
Deviating from local minima

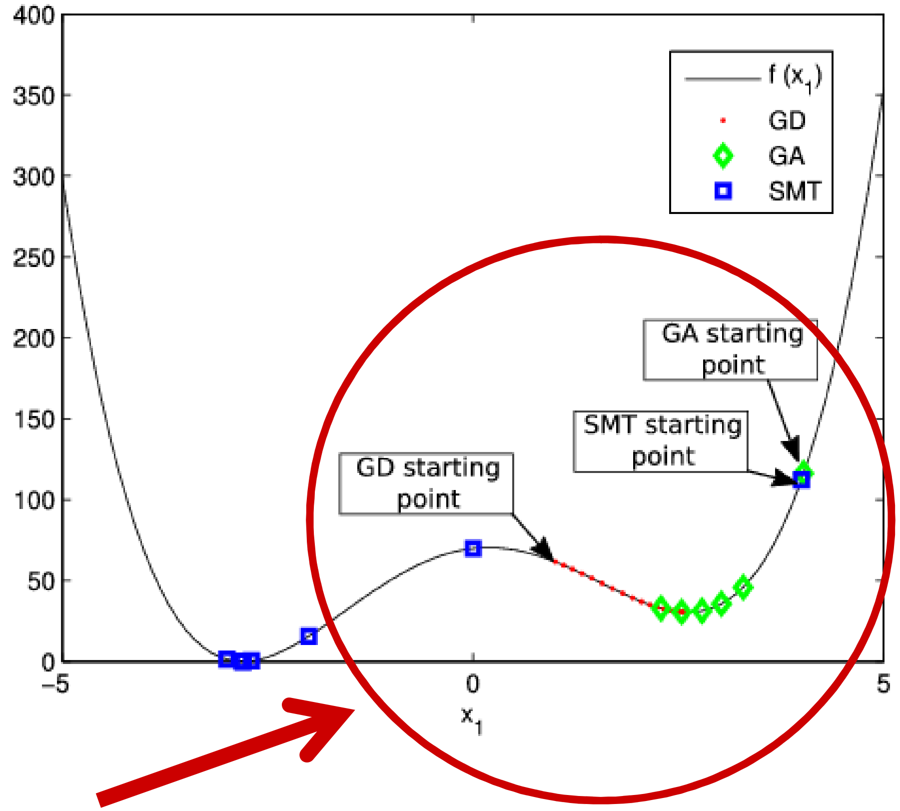
- Mathematical techniques and heuristics depend on initialization and can not ensure overall optimization.



Deviating from local minima

- Mathematical techniques and heuristics depend on initialization and can not ensure overall optimization.

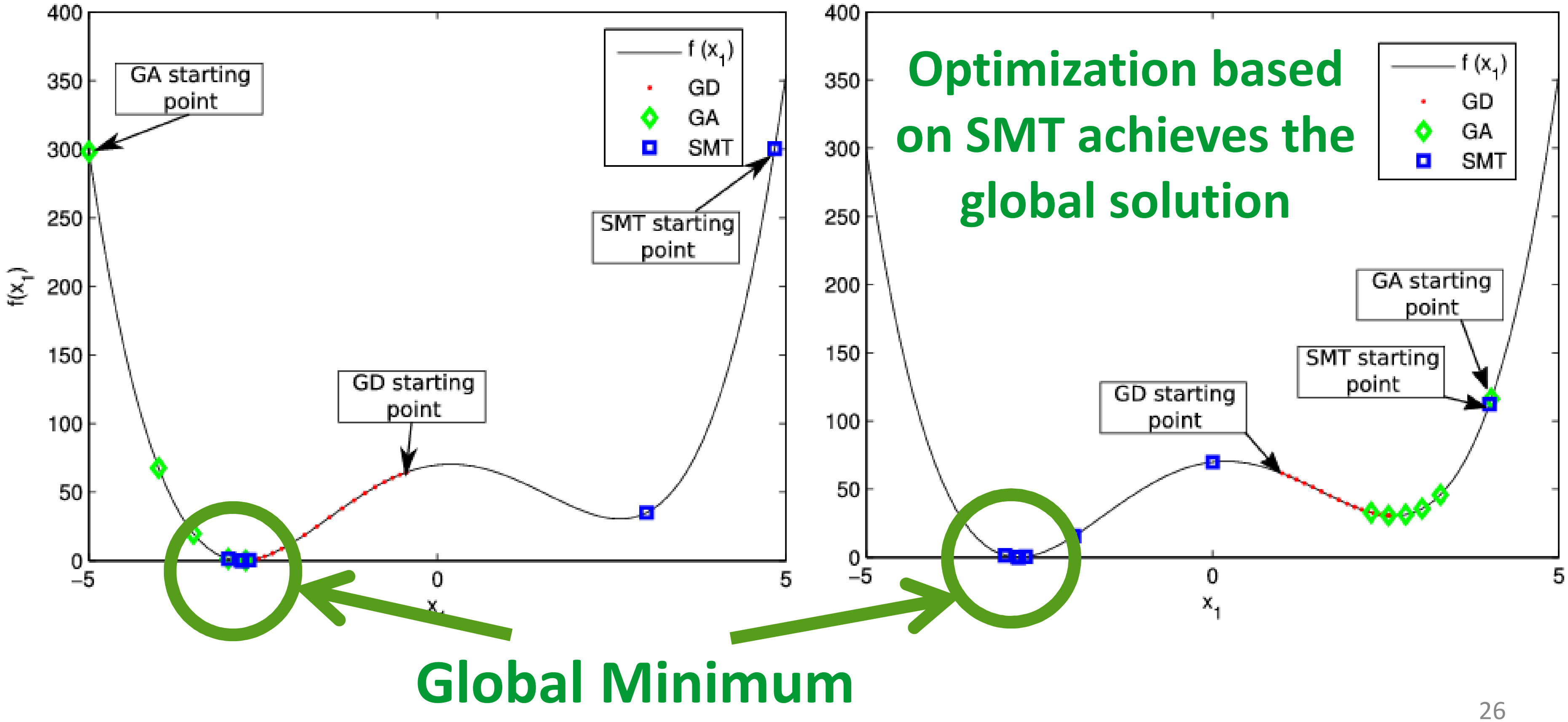




These techniques can easily stop in a local minimum

Deviating from local minima

- Mathematical techniques and heuristics dependent on initialization and can not assure global optimization



Functions with prior knowledge

- There are functions, in which we have some a priori knowledge, for example, semi-defined or positive definite functions.

$$f(x) \geq 0 \;\; e \;\; f(x) > 0$$

- Distance or energy functions belong to that class of functions

- From this, it is possible to propose modifications in the previous algorithm to improve the convergence time

CEGIO-F Algorithm

Data: A cost function $f(x)$, a set of constraints Ω, and a desired precision η.

Results: The optimal decision vector x^* and the optimal cost function $f(x^*)$.

Initialize $f_m = 0$;
Initialize $f(x^{(0)})$ randomly;
Initialize precision variables with $p = 1, i = 1$ e $k = \log p$;
Declare the decision variables $x^{(i)}$ as non-deterministic integer variables;
while $k \leq \eta$ **do**
    set the limits of x with the *ASSUME* directive, such that $x \in \Omega^{\mathrm{k}}$;
    describe the model for $f(x)$;
    describe $\delta = (f\left(x^{(i-1)}\right) - f_m)/\alpha$;
    if $(f\left(x^{(i-1)}\right) - fm > 0.00001)$ **then**
        do
            set constraint $f\left(x^{(i)}\right) < f(x^{(i-1)})$ as the ASSUM directive;
            while $(fm \leq f(x^{(i-1)}))$ **do**
                check for satisfiability of l_{optimum} given by equation (slide 8) with the ASSERT directive;
                make $fm = fm + \delta$;
            end
            update $x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$ based on the counter-example;
            make $i = i + 1$;
        while l_{optimum} is satisfying;
    end
    else
        break;
    end
    update the precision variable *p*, and consequently *k*;
end
$x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$;
return $x^* = x^{(i)}$;

CEGIO-F Algorithm

Data: A cost function $f(x)$, a set of constraints Ω, and a desired precision η.

Results: The optimal decision vector x^* and the optimal cost function $f(x^*)$.

Initialize $f_m = 0$;
Initialize $f(x^{(0)})$ randomly;
Initialize precision variables with $p = 1, i = 1$ e $k = \log p$;
Declare the decision variables $x^{(i)}$ as non-deterministic integer variables;
while $k \leq \eta$ **do**
    set the limits of x with the *ASSUME* directive, such that $x \in \Omega^k$;
    describe the model for $f(x)$;
    describe $\delta = (f(x^{(i-1)}) - f_m)/\alpha$;
    if $(f(x^{(i-1)}) - f_m > 0.00001)$ **then**
        do
            set constraint $f(x^{(i)}) < f(x^{(i-1)})$ as the ASSUM directive;
            while $(fm \leq f(x^{(i-1)}))$ **do**
                check for satisfiability of l_{optimum} given by equation (slide 8) with the ASSERT directive;
                make $fm = fm + \delta$;
            end
            update $x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$ based on the counter-example;
            make $i = i + 1$;
        while l_{optimum} is satisfying;
    end
    else
        break;
    end
    update the precision variable *p*, and consequently *k*;
end
$x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$;
return $x^* = x^{(i)}$;

δ, sets the increment step for the candidate values to minimum

CEGIO-F Algorithm

Data: A cost function $f(x)$, a set of constraints Ω, and a desired precision η.
Results: The optimal decision vector x^* and the optimal cost function $f(x^*)$.

Initialize $f_m = 0$;
Initialize $f(x^{(0)})$ randomly;
Initialize precision variables with $p = 1, i = 1$ e $k = \log p$;
Declare the decision variables $x^{(i)}$ as non-deterministic integer variables;
while $k \leq \eta$ **do**
    set the limits of x with the *ASSUME* directive, such that $x \in \Omega^k$;
    describe the model for $f(x)$;
    describe $\delta = (f(x^{(i-1)}) - f_m)/\alpha$;
    if $(f(x^{(i-1)}) - fm > 0.00001)$ **then**
        do
            set constraint $f(x^{(i)}) < f(x^{(i-1)})$ as the ASSUM directive;
            while $(fm \leq f(x^{(i-1)}))$ **do**
                check for satisfiability of l_{optimum} given by equation (slide 8) with the ASSERT directive;
                make $fm = fm + \delta$;
            end
            update $x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$ based on the counter-example;
            make $i = i + 1$;
        while l_{optimum} is satisfying;
    end
    else
        break;
    end
    update the precision variable p, and consequently k;
end
$x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$;
return $x^* = x^{(i)}$;

No checking is required if $f(x^*) = 0$

CEGIO-F Algorithm

Data: A cost function $f(x)$, a set of constraints Ω, and a desired precision η.

Results: The optimal decision vector x^* and the optimal cost function $f(x^*)$.

Initialize $f_m = 0$;
Initialize $f(x^{(0)})$ randomly;
Initialize precision variables with $p = 1, i = 1$ e $k = \log p$;
Declare the decision variables $x^{(i)}$ as non-deterministic integer variables;
while $k \leq \eta$ **do**
    set the limits of x with the *ASSUME* directive, such that $x \in \Omega^k$;
    describe the model for $f(x)$;
    describe $\delta = (f(x^{(i-1)}) - f_m)/\alpha$;
    if $(f(x^{(i-1)}) - fm > 0.00001)$ **then**
        do
            set constraint $f(x^{(i)}) < f(x^{(i-1)})$ as the ASSUM directive;
            while $(fm \leq f(x^{(i-1)}))$ **do**
                check for satisfiability of l_{optimum} given by equation (slide 8) with the ASSERT directive;
                make $fm = fm + \delta$;
            end
            update $x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$ based on the counter-example;
            make $i = i + 1$;
        while l_{optimum} is satisfying;
    end
    else
        break;
    end
    update the precision variable p, and consequently k;
end
$x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$;
return $x^* = x^{(i)}$;

The WHILE creates
$\alpha + 1$ properties
to check

Convex functions

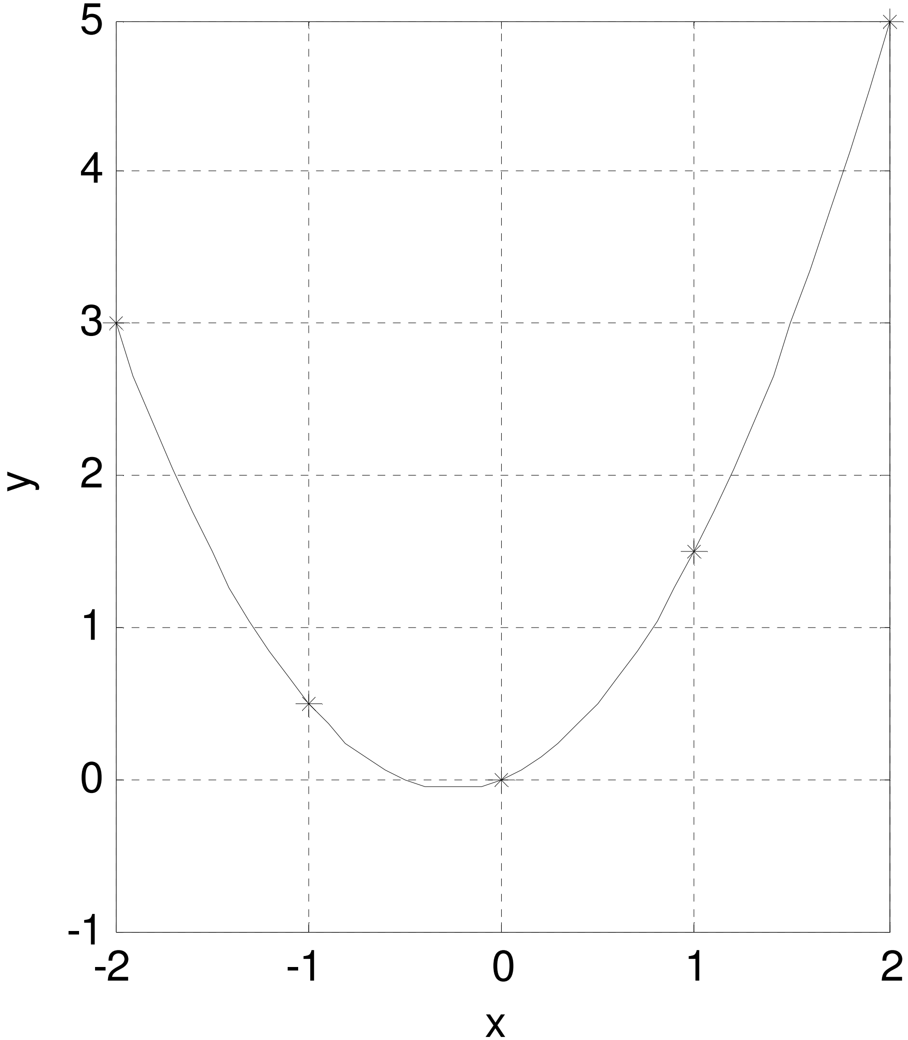
- Another type of special functions are convex functions. These are functions that satisfy triangular inequality.

$$f(\alpha x_1 + \beta x_2) \leq \alpha f(x_1) + \beta f(x_2)$$

$$x_i \in \mathbb{R}^n, with\ i = 1,2$$

$$\alpha, \beta \in \mathbb{R}, with\ \alpha + \beta = 1, \alpha \geq 0\ \ e\ \ \beta \geq 0$$

Convex functions



Convex functions

- A local minimum of a convex function *f*, in a convex set, is always a global minimum of *f*
- It is possible to update the set of constraints of the problem from the solution obtained before increasing the precision
  - The limits are defined by the predecessor and successor values of the solution found so far

CEGIO-S Algorithm

Data: A cost function $f(x)$, a set of constraints Ω, and a desired precision η.
Results: The optimal decision vector x^* and the optimal cost function $f(x^*)$.

Initialize $f(x^{(0)})$ randomly;
Initialize precision variables with $p = 1, i = 1$ e $k = \log p$;
Declare the decision variables $x^{(i)}$ as non-deterministic integer variables;
while $k \leq \eta$ **do**
    set the limits of x with the *ASSUME* directive, such that $x \in \Omega^{\mathbf{k}}$;
    describe the model for $f(x)$;
    do
        set constraint $f\left(x^{(i)}\right) < f(x^{(i-1)})$ as the ASSUM directive;
        check for satisfiability of $l_{\mathbf{optimum}}$ given by equation (slide 8) with the ASSERT directive;
        analysis $x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$ based on the counter-example;
        make $i = i + 1$;
    while $l_{\mathbf{optimum}}$ is satisfying;
    update the set $\Omega^{\mathbf{k}}$;
    update the precision variable *p*, and consequently *k*;
end
$x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$;
return $x^* = x^{(i)}$;

CEGIO-S Algorithm

Data: A cost function $f(x)$, a set of constraints Ω, and a desired precision η.

Results: The optimal decision vector x^* and the optimal cost function $f(x^*)$.

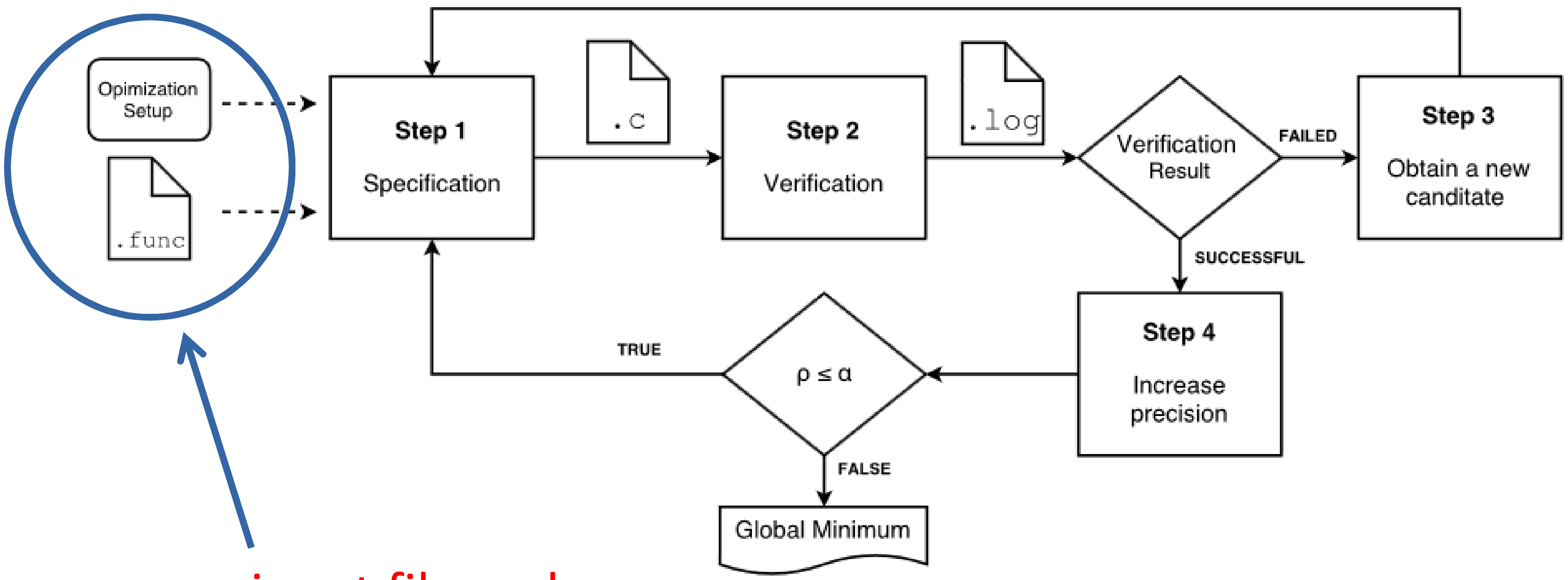
Initialize $f(x^{(0)})$ randomly;
Initialize precision variables with $p = 1, i = 1$ e $k = \log p$;
Declare the decision variables $x^{(i)}$ as non-deterministic integer variables;
while $k \leq \eta$ **do**
    set the limits of x with the *ASSUME* directive, such that $x \in \Omega^k$;
    describe the model for $f(x)$;
    do
        set constraint $f(x^{(i)}) < f(x^{(i-1)})$ as the ASSUM directive;
        check for satisfiability of l_{optimum} given by equation (slide 8) with the ASSERT directive;
        analysis $x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$ based on the counter-example;
        make $i = i + 1$;
    while l_{optimum} is satisfying;
    update the set Ω^{k};
    update the precision variable *p*, and consequently *k*;
end
$x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$;
return $x^* = x^{(i)}$;

Restriction set update

OptCE: A Counterexample-Guided Inductive Optimization Solver

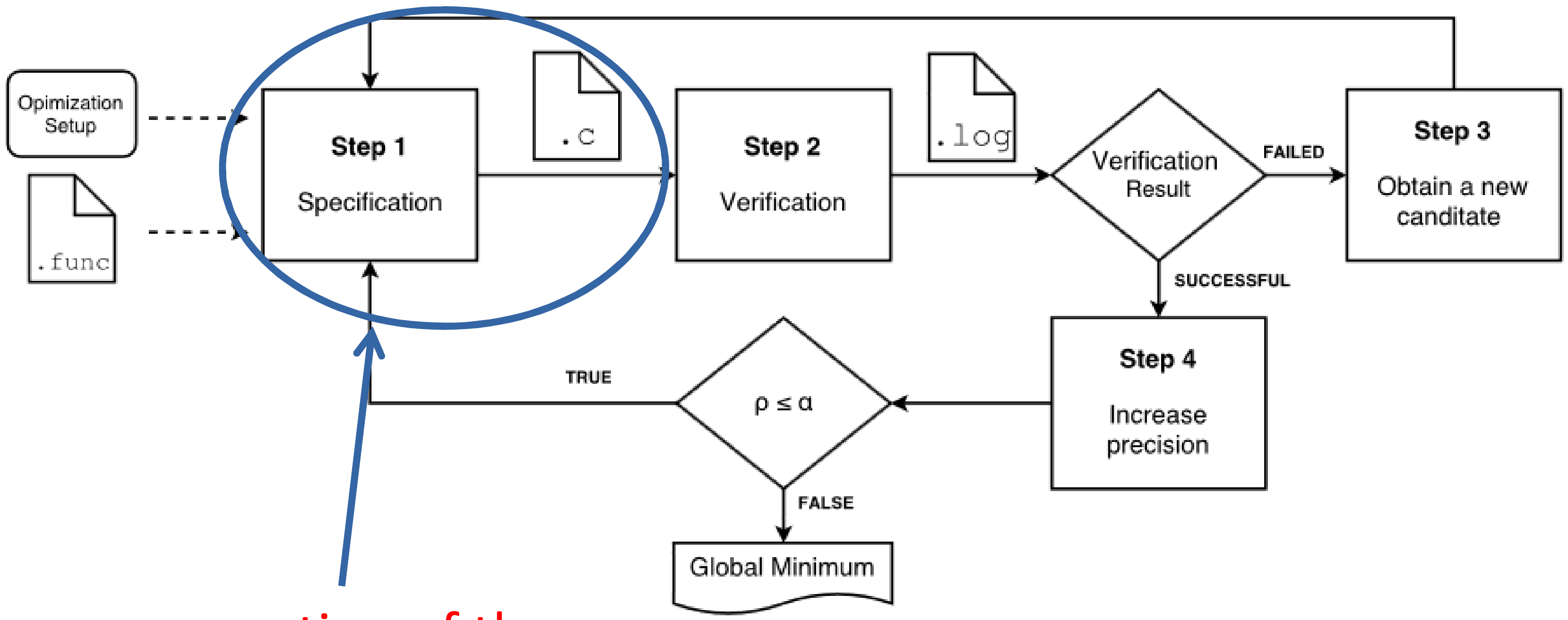
- The OptCE tool implements the CEGIOs algorithms
- Performs optimization based on counterexamples with various configurations of verifiers and solvers
- Establishes a new approach to optimize functions

OptCE: Architecture



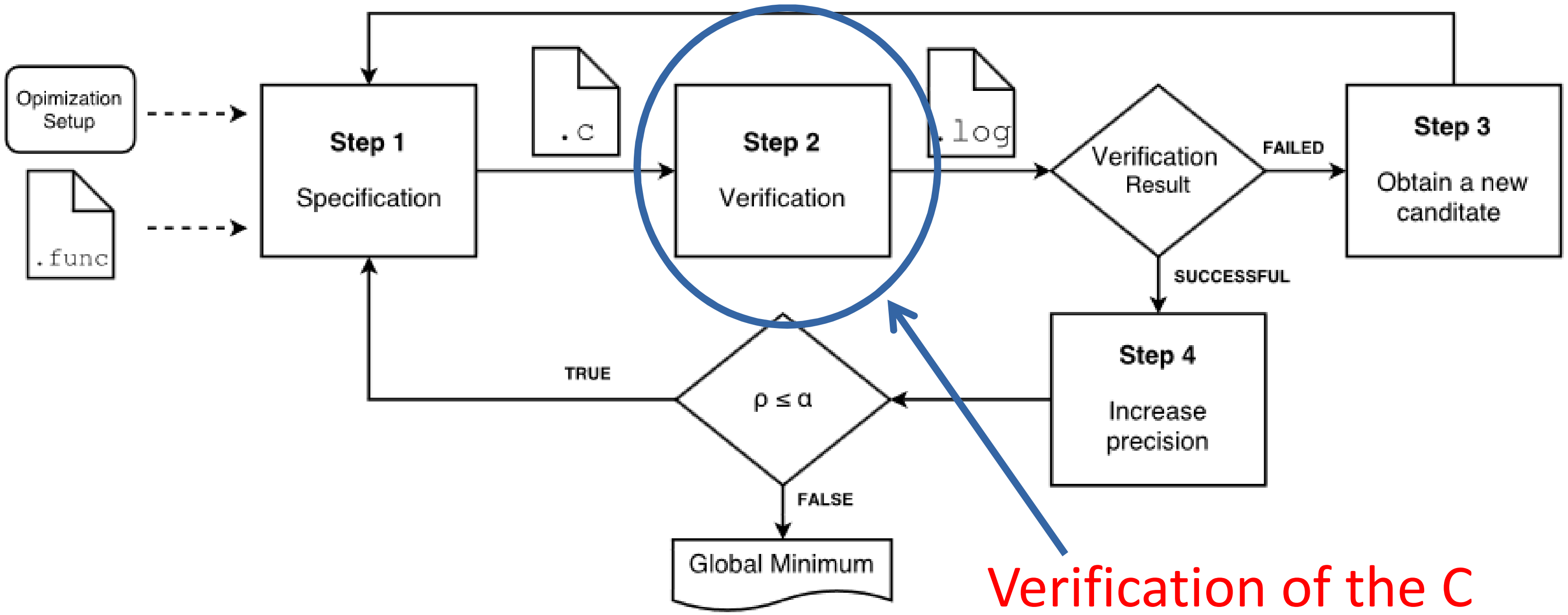
input file and parameters in terminal

OptCE: Architecture



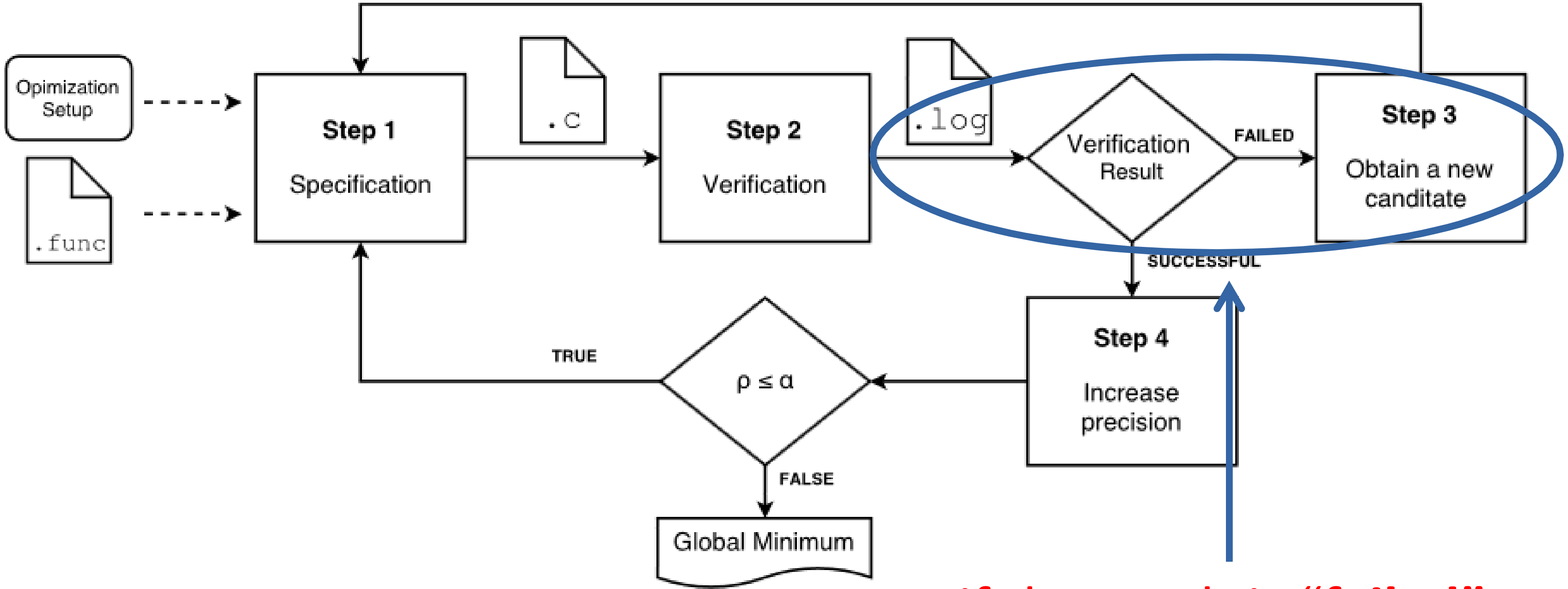
generation of the C file with specification

OptCE: Architecture



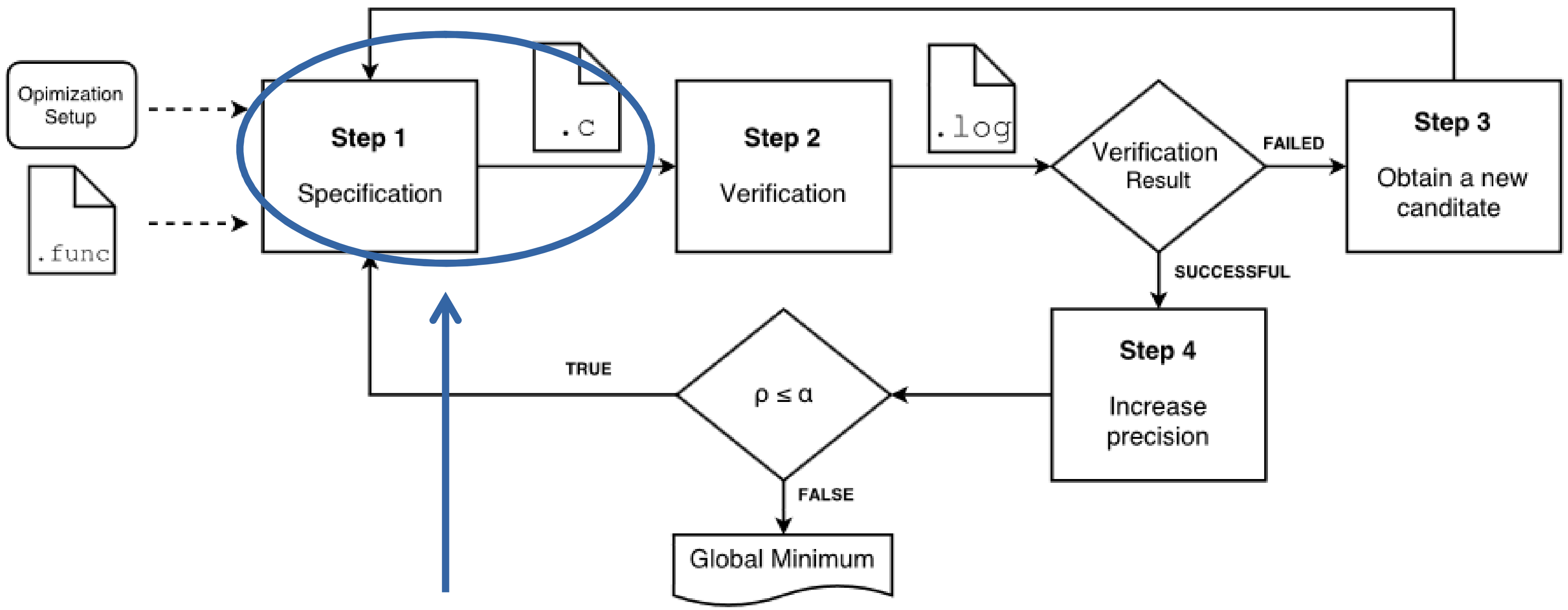
Verification of the C code with the verifier and solver established

OptCE: Architecture



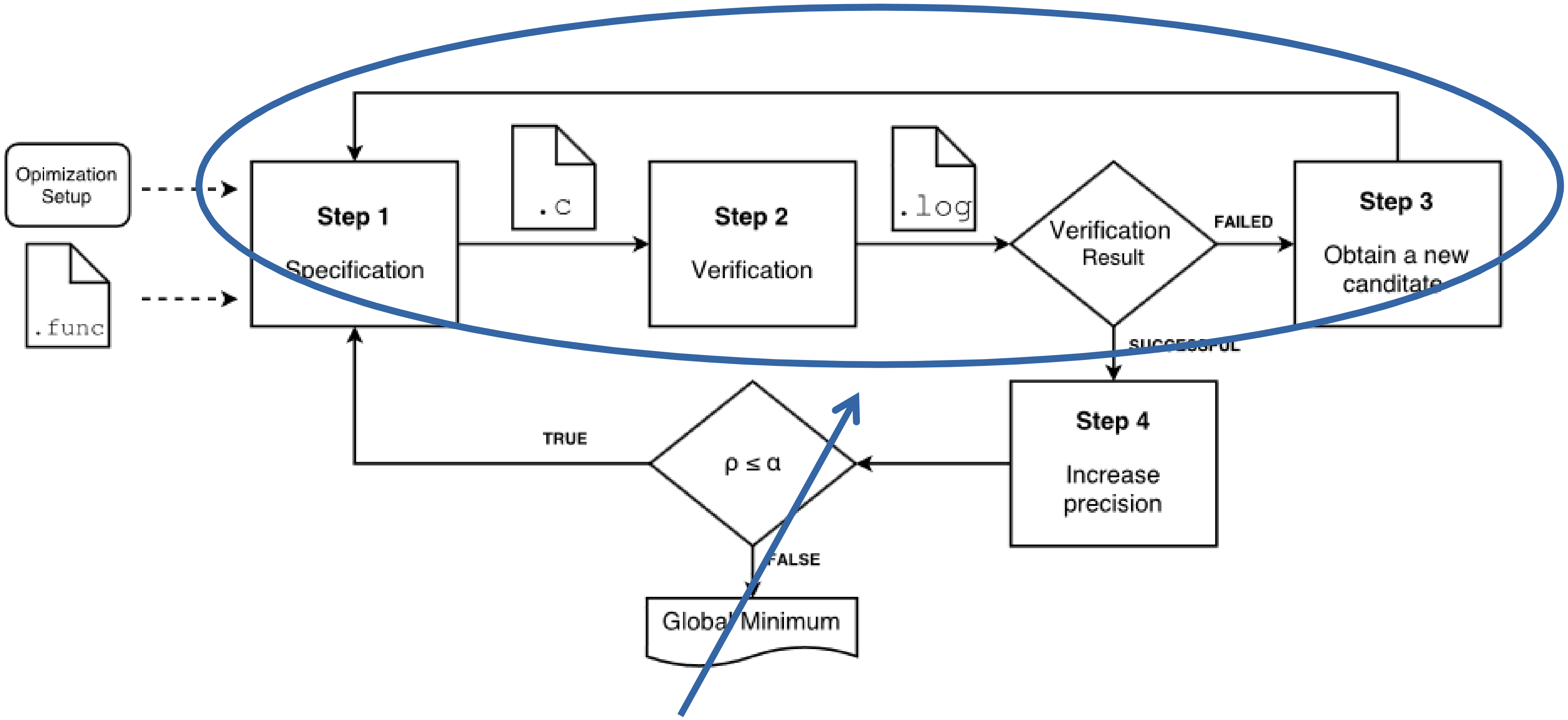
if the result is "**failed**", we obtain a new minimum candidate function

OptCE: Architecture



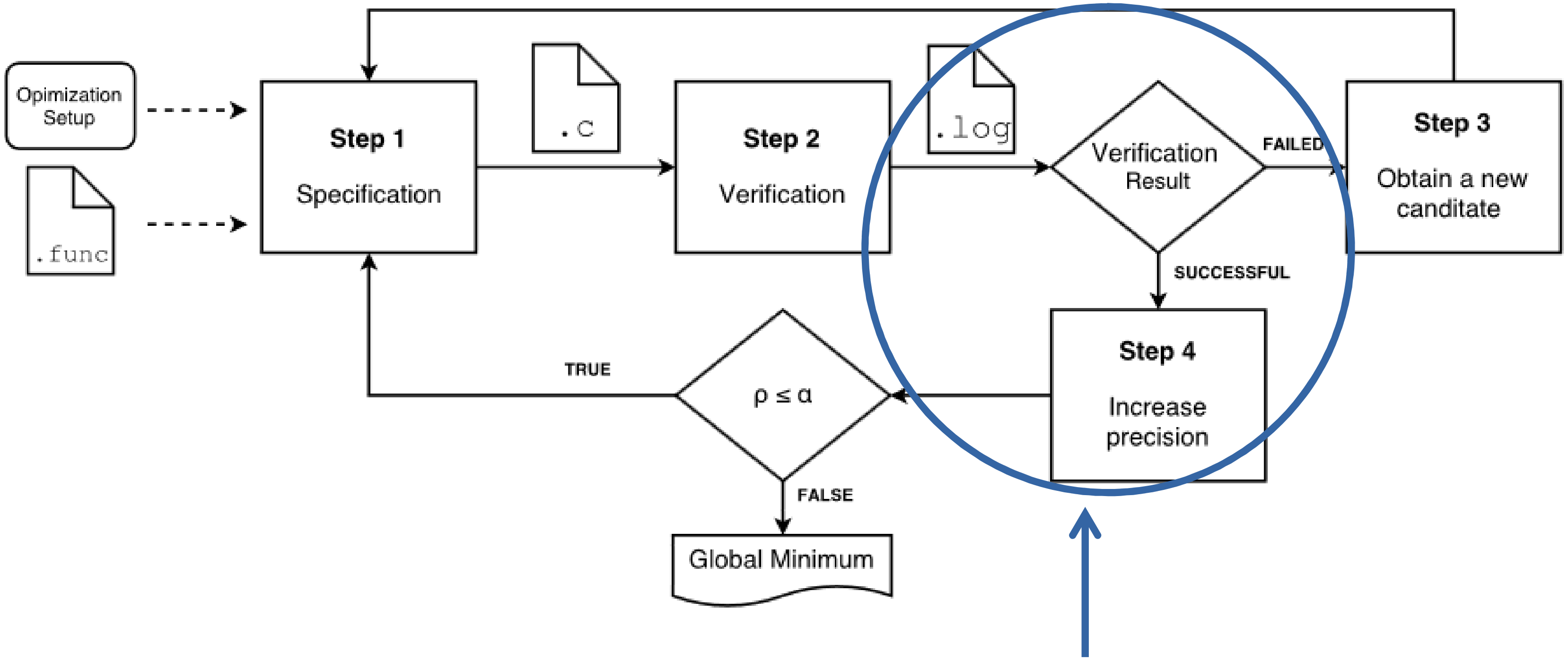
The value found is used to specify a new C file,
The new candidate function value is used as the start of the algorithm

OptCE: Architecture



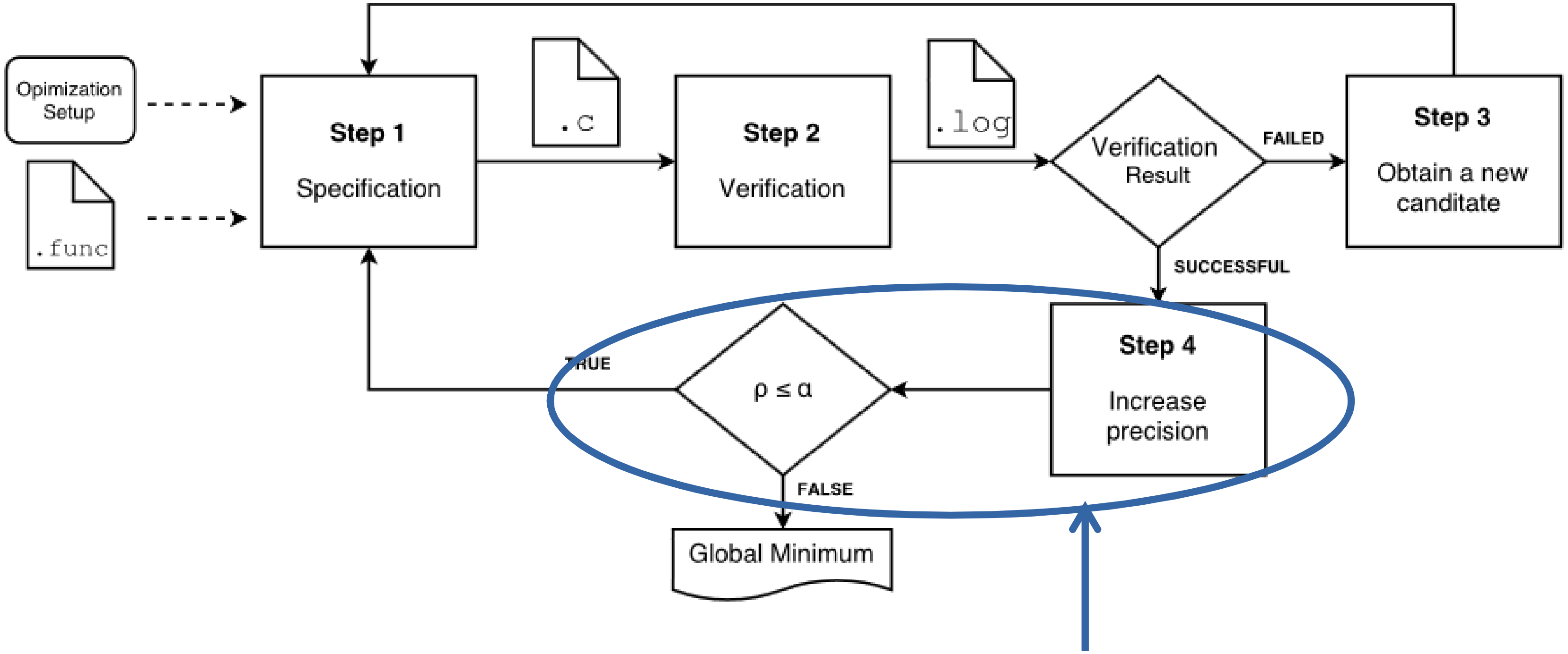
This cycle remains until the check is SUCCESSFUL.

OptCE: Architecture



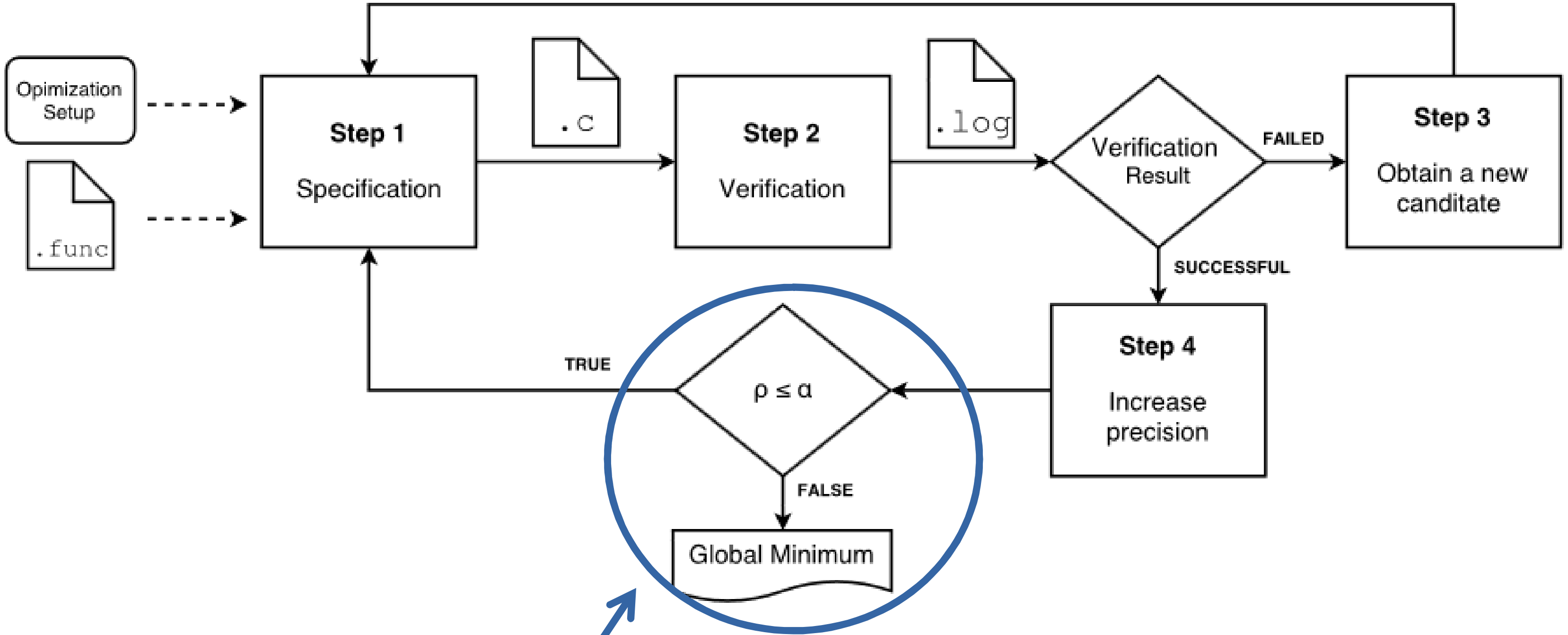
When the check is SUCCESSFUL, it means that we have found the global minimum of the function with the defined precision

OptCE: Architecture



Precision is incremented and checked if it still belongs to the desired precision limit

OptCE: Architecture



If not (FALSE), we find the global minimum wanted with that precision.
If yes (TRUE), we update the precision in the algorithm at runtime to generate a new specification

OptCE: Input File

$$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ & \cdots \\ x_{n1} & x_{n2} \end{bmatrix}$$

- Format adopted for constraint matrices
- Ex: Input file for function *adjiman*

    Fobj = cos2(x1)*sin2(x2) – (x1/(x2*x2+1));

    #

    A = [-1 2; -1 1];

- Mathematical functions have been rewritten to simplify the verification process
- The user can write the math function and insert it into the OptCE math library

OptCE Features

- **BMC Configuration:** CBMC or ESBMC
- **Solver Configuration:** Boolector, Z3, MathSAT, MiniSAT
- **Algorithm Configuration:** CEGIO-G, CEGIO-S, CEGIO-F
- **Initialization:** Set the optimization start point
- **Insert Library:** Insert personal libraries with math functions
- **Timeout:** configures the time limit, in seconds
- **Precision:** set the desired precision, number of decimal places of a solution

Optimizing via OptCE

Call	Set Properties						
Binary + Function	BMC	Solver	Algorithm	Initialization	Library	Timeout	Precision
./optCE name.func	--esbmc --cbmc	--mathsat --boolector --z3 --minisat	--generalized --positive --convex	--start-value=?	--library=name	--timeout=?	--timeout=?

Experimental Evaluation

- Objectives
  - Evaluate the performance of the proposed algorithms
  - Check the performance of the SAT and SMT solvers for optimizing the functions
  - Compare the methodology with traditional techniques, such as: genetic algorithm, particle swarm, pattern search, simulated annealing and nonlinear programming

Experimental Evaluation

- Configuration of Experiments
  - A set of 10 functions used for testing optimization algorithms. These have different characteristics, such as: differentiable or non-differentiable, separable or non-separable, unimodal or multimodal etc.

#	Benchmark	Domain	Global Minimum
1	Alpine 1	$-10 \leq xi \leq 10$	$f(0,0) = 0$
2	Cosine	$-1 \leq xi \leq 1$	$f(0,0) = -0,2$
3	Styblinski Tang	$-5 \leq xi \leq 5$	$f(2.903,2.903) = -78.332$
4	Zirilli	$-10 \leq xi \leq 10$	$f(1.046,0) \approx -0,3523$
5	Booth	$-10 \leq xi \leq 10$	$f(1,3) = 0$
6	Himmelblau	$-5 \leq xi \leq 5$	$f(3,2) = 0$
7	Leon	$-2 \leq xi \leq 2$	$f(1,1) = 0$
8	Zettl	$-5 \leq xi \leq 10$	$f(0.029,0) = -0.0037$
9	Sum Square	$-10 \leq xi \leq 10$	$f(0,0) = 0$
10	Rotated Ellipse	$-500 \leq xi \leq 500$	$f(0,0) = 0$

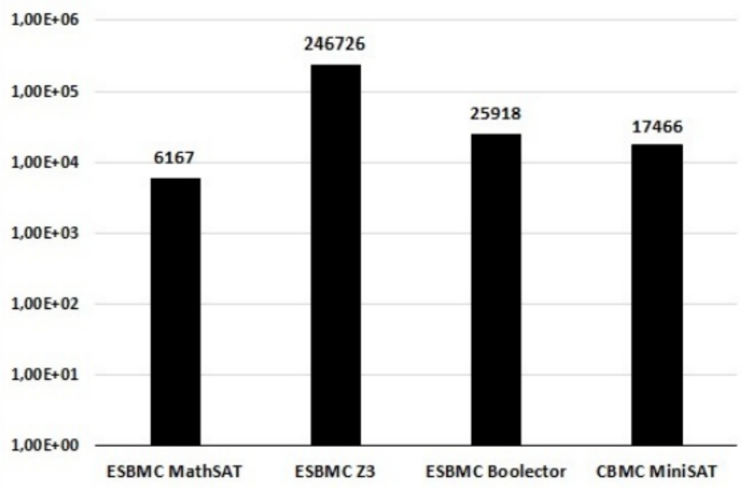
Experimental Evaluation

- Configuration of Experiments
  - CEGIO-G Algorithm { --generalized} - was employed in all functions
  - CEGIO-S Algorithm { --positive} - was applied to functions Booth, Himmelblau and Leon
  - CEGIO-F Algorithm { --convex} - was used for functions Zettl, Rotated Ellipse and Sum Square

Experimental Evaluation

- Experimental Results - CEGIO-G { --generalized}

#	ESBMC			CBMC
	MathSAT (s)	Z3(s)	Boolector(s)	MiniSAT(s)
1	1068	105192	3387	5344
2	4130	80481	5003	8509
3	443	37778	2027	2438
4	468	387	190	1143
5	7	1244	4016	2
6	12	14205	6217	4
7	5	2443	212	2
8	13	753	389	9
9	18	4171	4438	13
10	3	72	39	2



- Considering the proposed combinations, the optimization time varies significantly, where ESBMC + MathSAT is 2.8 times faster than CBMC + MiniSAT, while ESBMC + Z3 presents higher execution time.

Experimental Evaluation

- Experimental Results - CEGIO-S { --positive}

#	--positive				--generalized			
	ESBMC			CBMC	ESBMC			CBMC
	MathSAT (s)	Z3(s)	Boolector(s)	MiniSAT(s)	MathSAT(s)	Z3(s)	Boolector(s)	MiniSAT(s)
5	3	<1	1	3	7	1244	4016	2
6	4	1	1	2	12	14205	6217	4
7	3	<1	1	2	5	2443	212	2

- The benchmarks executed with the --positive flag had the time reduced considerably, therefore, no checks are made in the negative domain, which reduces the search space.

Experimental Evaluation

- Experimental Results – CEGIO-F { --convex}

#	--convex				--generalized			
	ESBMC			CBMC	ESBMC			CBMC
	MathSAT (s)	Z3 (s)	Boolector (s)	MiniSAT (s)	MathSAT (s)	Z3 (s)	Boolector (s)	MiniSAT (s)
8	15	6	21	5	13	753	389	9
9	14	3	19	5	18	4171	4438	13
10	3	1	2	2	3	72	39	2

- The tests with the benchmarks 8,9,10 using the flag -convex presented a significant reduction in the optimization time, this because, with each step of the verification the search space is reduced.

Experimental Evaluation

- Experimental Results – CEGIO algorithms x traditional techniques

#	OptCE			GA		ParSwarm		PatSearch		SA		NLP	
	Configuration	R%	T(s)	R%	T(s)	R%	T(s)	R%	T(s)	R%	T(s)	R%	T(s)
1	G + ESBMC + MathSAT	100	1068	29.1	1	22.2	3	16	4	0.4	1	4.8	9
2	G + ESBMC + MathSAT	100	4130	100	9	9.8	1	96.7	3	88.5	2	28.4	2
3	G + ESBMC + MathSAT	100	443	68.1	9	47.8	1	51.8	3	99.5	1	35.8	2
4	G + ESBMC + Boolector	100	190	95.7	9	53.9	1	98.8	3	74.4	1	62.5	2
5	P + ESBMC + Z3	100	< 1	100	10	100	2	100	6	93.5	1	100	2
6	P + ESBMC + Z3	100	1	42.4	9	43.9	1	26	3	21	1	35	2
7	P + ESBMC + Z3	100	< 1	84.4	1	80.3	2	1	7	24.3	1	100	4
8	C + CBMC + MiniSAT	100	5	100	9	48.1	1	99.8	4	26.4	1	100	3
9	C + ESBMC + Z3	100	3	100	9	71.5	1	100	4	96.9	1	100	2
10	C + ESBMC + Z3	100	1	100	9	100	2	100	7	99.8	1	100	2

- The great difference of the OptCE in relation to the other techniques is the rate of success. While the other technicians get stuck in local minima, OptCE finds the global minimum.

Conclusion

- The OptCE tool formalizes a new optimization proposal, which is based on the counter-example analysis of software verifiers.
- This work allowed to implement the GEGIOs algorithms.
- The comparisons show that the approach evolved among the CEGIO algorithms, proposing better and more specific solutions in the case of convex and non-negative functions.
- It is also seen that the tool hit rate is higher than the other analysis techniques.

Future work

- Incorporate checks using other solvers with the MiniSAT.
- Adapt the tool to run in different cores, increasing the optimization time linearly.
- Improve the input file.