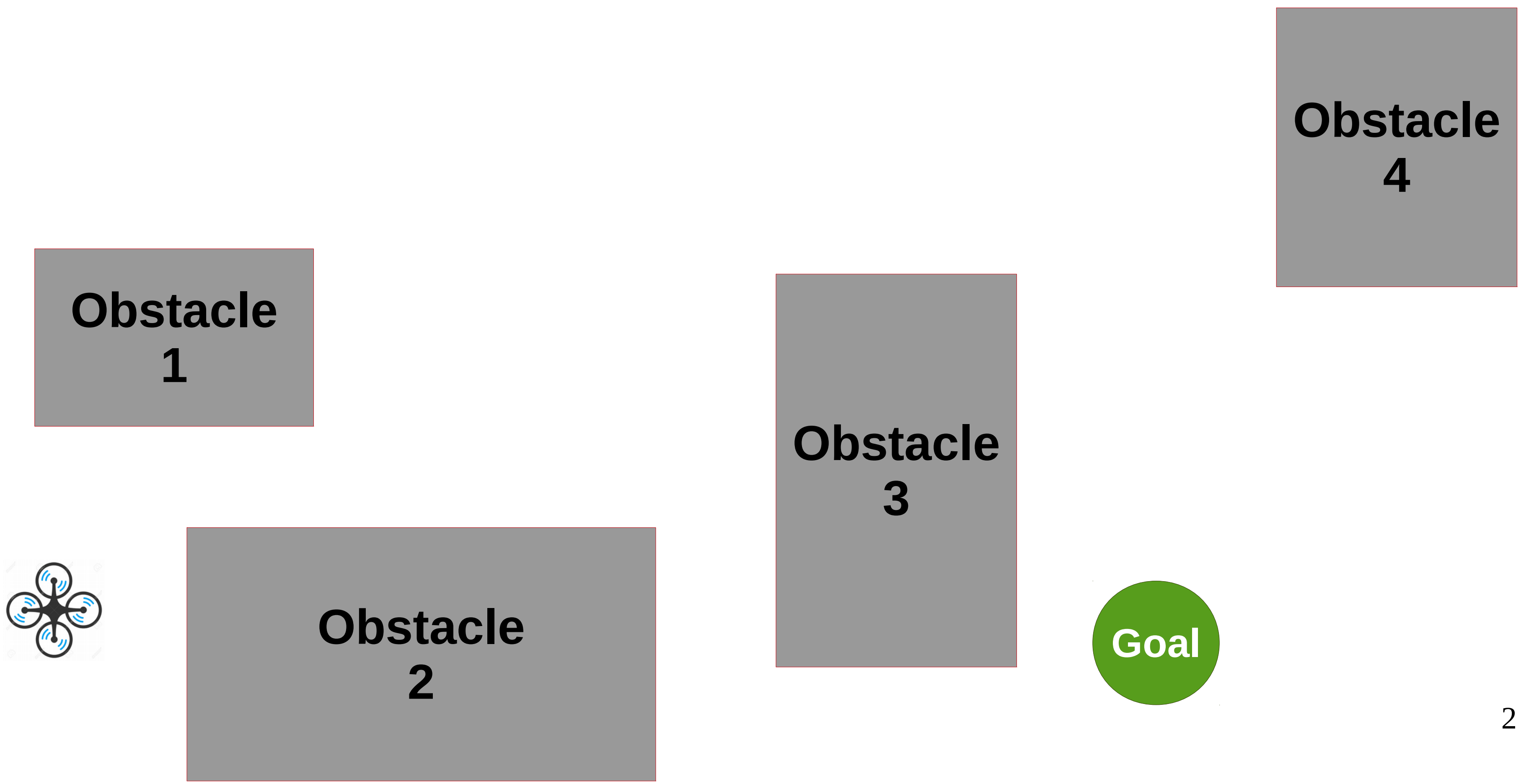
SMT-based optimization applied to nonconvex problems

Iury Bessa
iurybessa@ufam.edu.br

Joint work with: Rodrigo Araújo, Lucas Cordeiro, João Edgar Chaves Filho, and Higo Albuquerque

Motivating Example

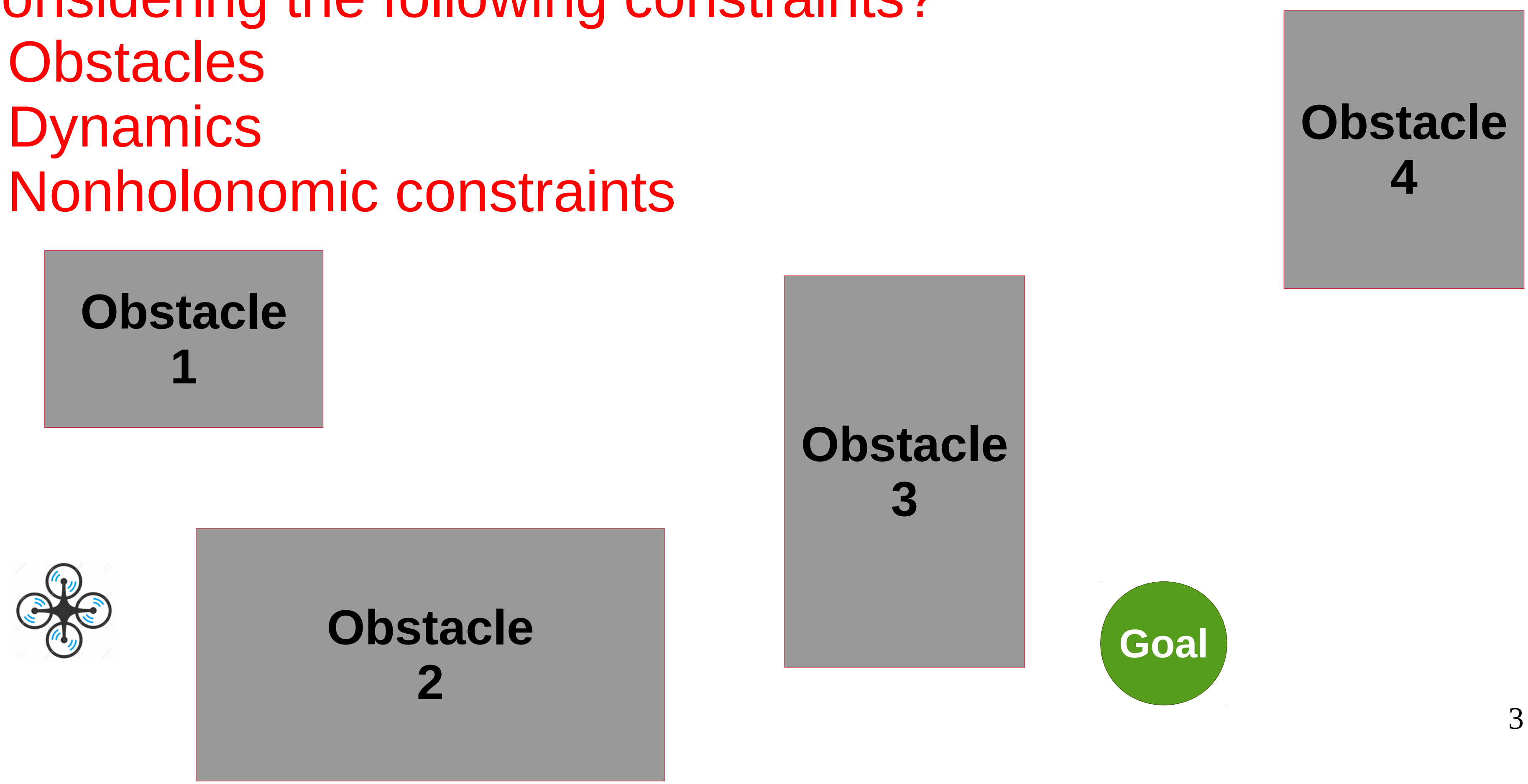
- Consider the following trajectory planning problem:



Motivating Example

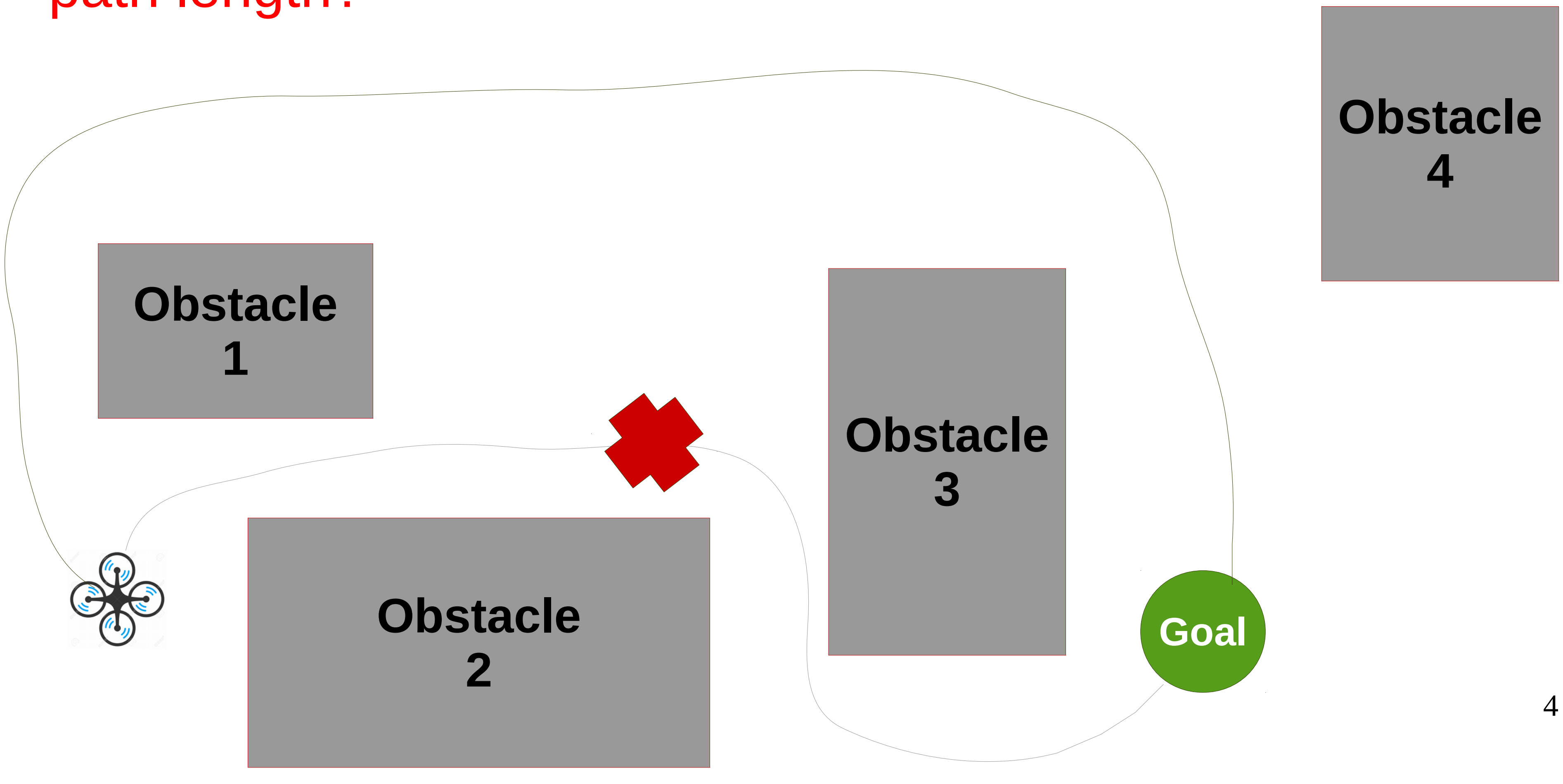
- Consider the following trajectory planning problem:

What is the **shortest** trajectory for this UAV considering the following constraints?
- Obstacles
- Dynamics
- Nonholonomic constraints



Motivating Example

How to find a solution that satisfies the constraints and minimizes the path length?



Motivating Example

- The aforementioned trajectory planning problem is represented as an optimization problem:

$$\min_{L} J(L),$$

$$\mathrm{s.}t.\Omega,$$

$$J = \sum_{i=1}^{n-1} \|\vec{R}_{P_i P_{i+1}}\|_2$$

Motivating Example

- The aforementioned trajectory planning problem is represented as an optimization problem:

$$\min_{L} J(L),$$

$$s.t.\ \Omega,$$

The cost function

$$J = \sum_{i=1}^{n-1} \|\vec{R}_{P_i P_{i+1}}\|_2$$

Motivating Example

- The aforementioned trajectory planning problem is represented as an optimization problem:

$$\min_{L} J(L),$$

$$\text{s.t. } \Omega,$$

$$J = \sum_{i=1}^{n-1} \|\vec{R}_{P_i P_{i+1}}\|_2$$

The vector from the *i*-th to the *i+1*-th point of the trajectory

Motivating Example

- The aforementioned trajectory planning problem is represented as an optimization problem:

$$\min_{L} J(L),$$

$$s.t.\ \Omega,$$

$$J = \sum_{i=1}^{n-1} \|\vec{R}_{P_i P_{i+1}}\|_2$$

The trajectory is the sequence of *n* points that solves the problem

Motivating Example

- The aforementioned trajectory planning problem is represented as an optimization problem:

$$\min_{L} J(L),$$

$$\mathrm{s.t.}\ \Omega,$$

$$J = \sum_{i=1}^{n-1} \| \vec{R}_{P_i P_{i+1}} \|_2$$

The set of constraints

Optimization problems

- Optimization problems appear in various research areas, including computer science and engineering
- The more complex problems (e.g. multiobjective or nonconvex) are usually solved by metaheuristic techniques (e.g. genetic algorithm)
- These techniques provide fast solutions for these complex problems, but are usually trapped by local minima

Optimization problems

- Optimization problems appear in various research areas, including computer science and engineering
- The more complex problems (e.g. multiobjective or nonconvex) are usually solved by metaheuristic techniques (e.g. genetic algorithm)
- These techniques provide fast solutions for these complex problems, but are usually trapped by local minima

How to ensure the global optimization more efficiently than metaheuristic techniques?

Objectives

The main objective of this work is to apply SMT-based optimization to globally optimize nonconvex functions

- Develop an SMT-based optimization algorithm
- Optimize nonconvex functions with the proposed SMT-based optimization algorithm
- Compare the results with other traditional optimization techniques using standard benchmarks

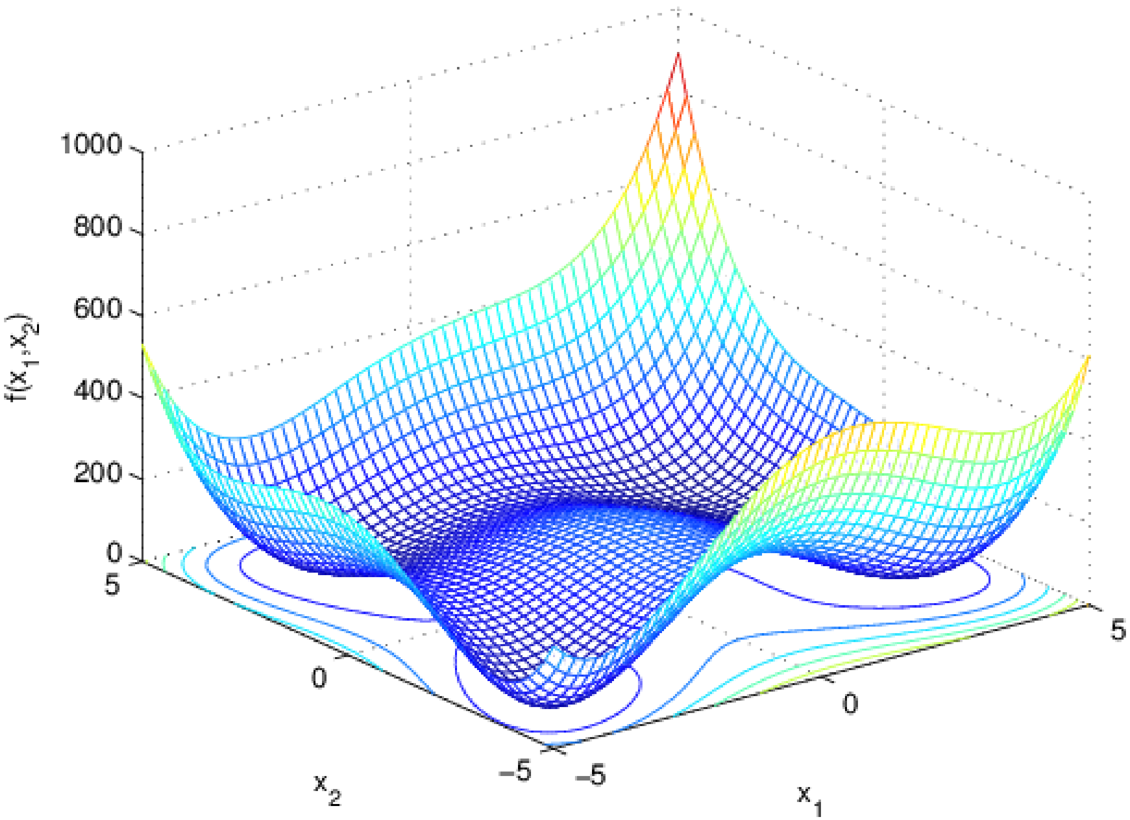
Defining the nonconvex optimization problem

- Let $f: D \rightarrow \mathbb{R}$ be a cost function, such $D \subset \mathbb{R}^n$ is the space of decision variables and $f(x_1, x_2, \ldots, x_n) \equiv f(\mathbf{x})$;
- Let $\Omega \subset \mathbb{R}^n \times \mathbb{R}$ be a set of constraints;
- A multivariable optimization problem consists in finding an optimal vector $\mathbf{x}^*$ which minimizes f considering Ω:

$$\min_{\mathbf{x}} f(\mathbf{x}),$$
$$\mathrm{s.}t.\,\Omega,$$

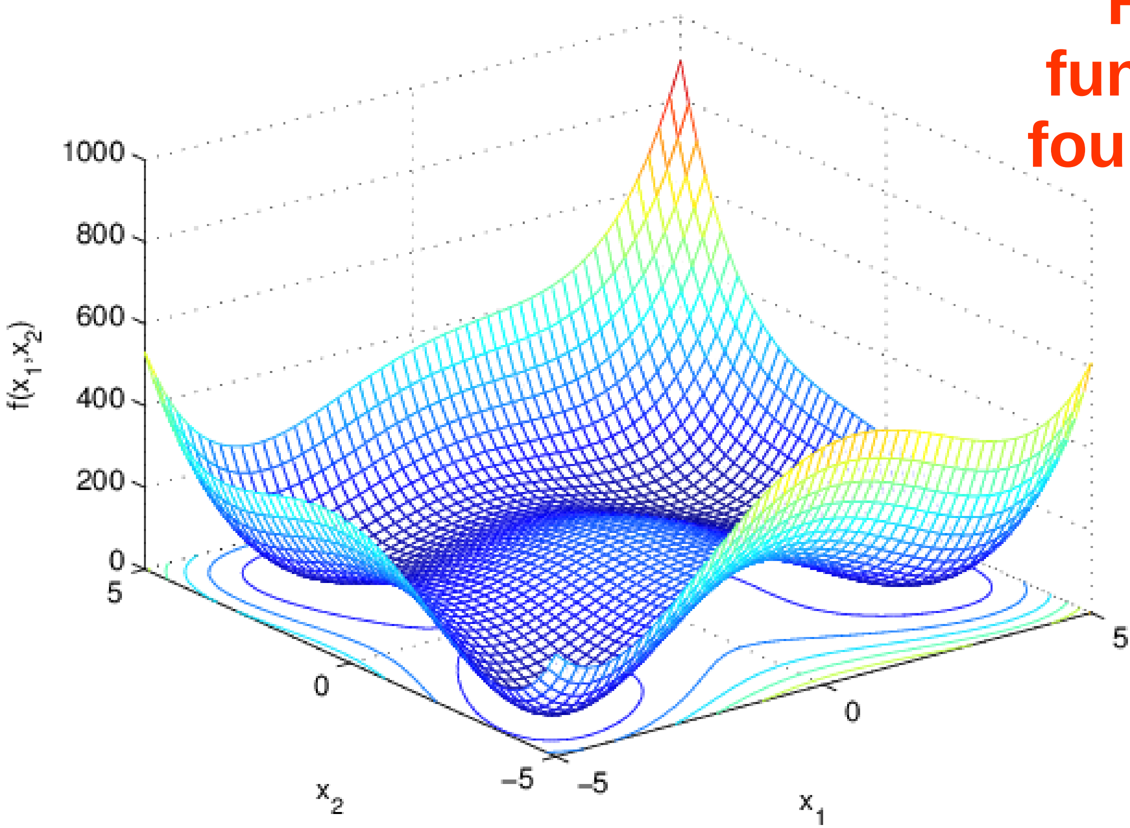
- The above problem will be a nonconvex optimization problem *iff* $f(\mathbf{x})$ is a nonconvex function

Example of nonconvex functions



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

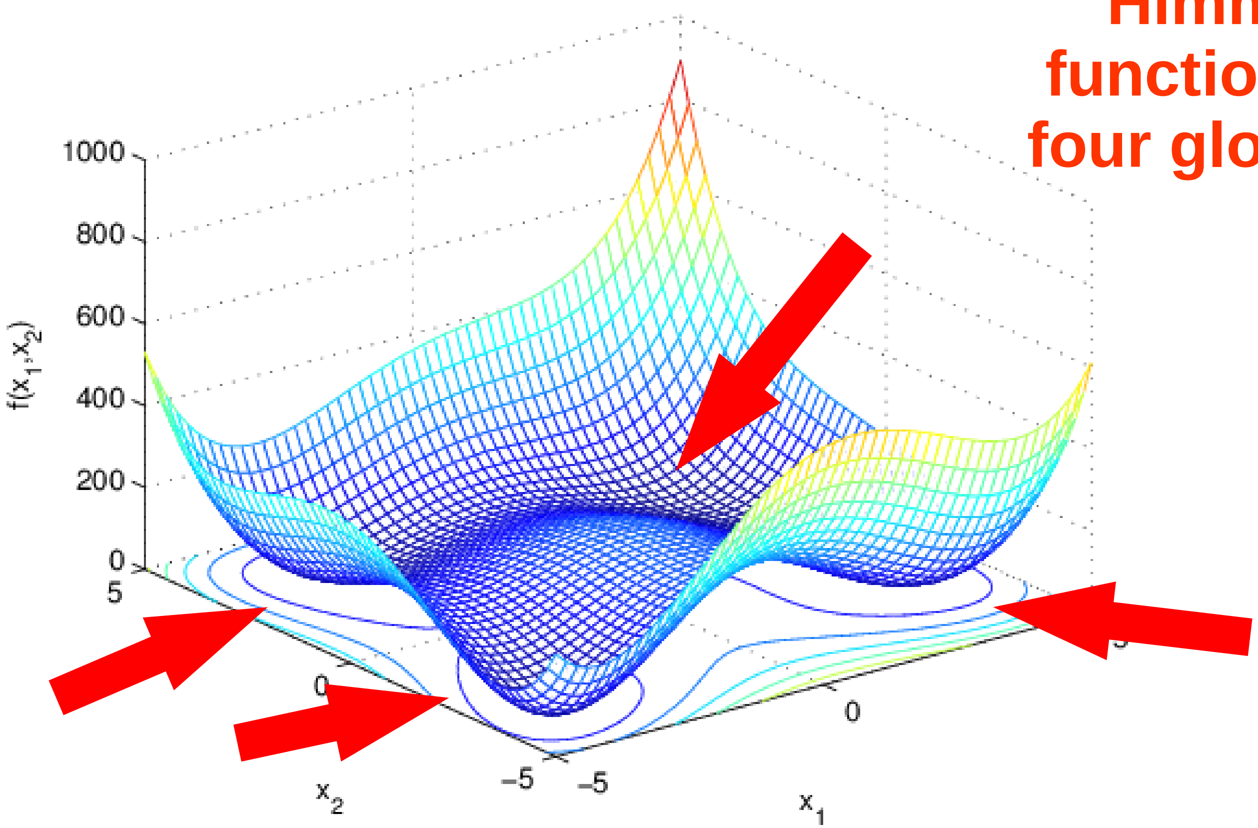
Example of nonconvex functions



Himmelblau's function presents four global minima

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

Example of nonconvex functions



Himmelblau's function presents four global minima

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

Modeling the optimization problem using a model checker

- The directives ASSUME and ASSERT should be employed for modeling optimization problems
  - ASSUME: is used for modeling the knowledge about the problem and the constraints set
  - ASSERT: is used for holding the global optimization condition $l_{optimal}$

$$l_{optimal} \Leftrightarrow f(\mathbf{x}) > f_p$$

Modeling the optimization problem using a model checker

- The ESBMC and its intrisic functions (*__ESBMC_assume* and *__ESBMC_assert*) were used in this work, but any other model checker could be used
- Decision variables are defined as non-deterministic integers
- The verification engine is executed by iteratively increasing the precision and converging to the optimal solution

Modeling the optimization problem using a model checker

- An integer variable controls the precision and discretizes the state-space:

$$p = 10^{n(i)}$$

- The *i*-th verification step stops when:

$$f\left(\mathbf{x}^{(i)}\right) \leq f_p$$

- When it occurs, f_p is updated with the $f\left(\mathbf{x}^{(i)}\right)$ from the counterexample

SMT-based Optimization Algorithm

Input: a cost function *f(x)*, a constraint set Ω, and a desired precision ϵ
Output: the optimal decision variable vector **x***, and the optimal function value $f(\mathbf{x}^*)$

1. Initialize $f(\mathbf{x}^{(0)})$ randomly and the precision variable with $p=1$
2. Declare decision variables (**x**) as non-deterministic integer variables
3. ***while*** $p < \epsilon$ **do**
4. Define the bounds for **x** with `assume`
5. Describe a model for $f(\mathbf{x})$
6. Constrain $f(\mathbf{x}^{(i)}) < f(\mathbf{x}^{(i-1)})$ with `assume`
7. **for** every $f_c \leq f(\mathbf{x}^{(i-1)})$ **do**
8. Check the satisfiability of $\neg l_{optimal}$
9. **if** $\neg l_{optimal}$ is `SAT` **then**
10. Update $f(\mathbf{x}^{(i)})$ and $\mathbf{x}^{(i)}$ from the counterexample
11. Go back to step 6
12. **end**
13. **end**
14. Update the precision variable p = 10p
15. **end**
16. **return** $x^* = x^{(i)}$ and $f(x^*) = f(x^{(i)})$

Illustrative Example

- Let our optimization problem be:

$$\min_{x_1, x_2} \quad f(x_1, x_2)$$

$$\text{s.}t. -7 \leq x_1 \leq 0$$

$$0 \leq x_2 \leq 7$$

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

- This is the Himmelblau's function constrained to the 2nd quadrant

Illustrative Example

```c
int nondet_int();
int main(){
    int p = 1; //precision variable
    float f_ant = 100; // f_ant: previous obj function value
    int v = (int)(f_ant*p + 1);
    int X1 = nondet_int();
    int X2 = nondet_int();
    float x1, x2, fobj, fc;
    assume((X1>=-7*p) && (X1<=0*p));
    assume((X2>=0*p) && (X2<=7*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);
    assume( fobj < f_ant );
    for (int i = 0; i <= v; i++){
        fc = (float) i/p;
        assert( fobj > fc );
    }
    return 0;
}
```

Illustrative Example

The precision variable is started as 10^0

```c
int nondet_int();
int main(){
    int p = 1; //precision variable
    float f_ant = 100; // f_ant: previous obj function value
    int v = (int)(f_ant*p + 1);
    int X1 = nondet_int();
    int X2 = nondet_int();
    float x1, x2, fobj, fc;
    assume((X1>=-7*p) && (X1<=0*p));
    assume((X2>=0*p) && (X2<=7*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);
    assume( fobj < f_ant );
    for (int i = 0; i <= v; i++){
        fc = (float) i/p;
        assert( fobj > fc );
    }
    return 0;
}
```

Illustrative Example

The decision variables are declared as non-deterministic integers

```
int nondet_int();
int main(){
    int p = 1; //precision variable
    float f_ant = 100; // f_ant: previous obj function value
    int v = (int)(f_ant*p + 1);
    int X1 = nondet_int();
    int X2 = nondet_int();
    float x1, x2, fobj, fc;
    assume((X1>=-7*p) && (X1<=0*p));
    assume((X2>=0*p) && (X2<=7*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);
    assume( fobj < f_ant );
    for (int i = 0; i <= v; i++){
        fc = (float) i/p;
        assert( fobj > fc );
    }
    return 0;
}
```

Illustrative Example

```
int nondet_int();
int main(){
    int p = 1; //precision variable
    float f_ant = 100; // f_ant: previous obj function value
    int v = (int)(f_ant*p + 1);
    int X1 = nondet_int();
    int X2 = nondet_int();
    float x1, x2, fobj, fc;
    assume((X1>=-7*p) && (X1<=0*p));
    assume((X2>=0*p) && (X2<=7*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);
    assume( fobj < f_ant );
    for (int i = 0; i <= v; i++){
        fc = (float) i/p;
        assert( fobj > fc );
    }
    return 0;
}
```

Assumptions are used for reducing the state-space and specifying the constraints

Illustrative Example

```
int nondet_int();
int main(){
    int p = 1; //precision variable
    float f_ant = 100; // f_ant: previous obj function value
    int v = (int)(f_ant*p + 1);
    int X1 = nondet_int();
    int X2 = nondet_int();
    float x1, x2, fobj, fc;
    assume((X1>=-7*p) && (X1<=0*p));
    assume((X2>=0*p) && (X2<=7*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);
    assume( fobj < f_ant );
    for (int i = 0; i <= v; i++){
        fc = (float) i/p;
        assert( fobj > fc );
    }
    return 0;
}
```

The objective function is evaluated until the previous iteration solution

Illustrative Example

```
int nondet_int();
int main(){
    int p = 1; //precision variable
    float f_ant = 100; // f_ant: previous obj function value
    int v = (int)(f_ant*p + 1);
    int X1 = nondet_int();
    int X2 = nondet_int();
    float x1, x2, fobj, fc;
    assume((X1>=-7*p) && (X1<=0*p));
    assume((X2>=0*p) && (X2<=7*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);
    assume( fobj < f_ant );
    for (int i = 0; i <= v; i++){
        fc = (float) i/p;
        assert( fobj > fc );
    }
    return 0;
}
```

When this condition is false, the optimal candidate is updated and the verification is repeated

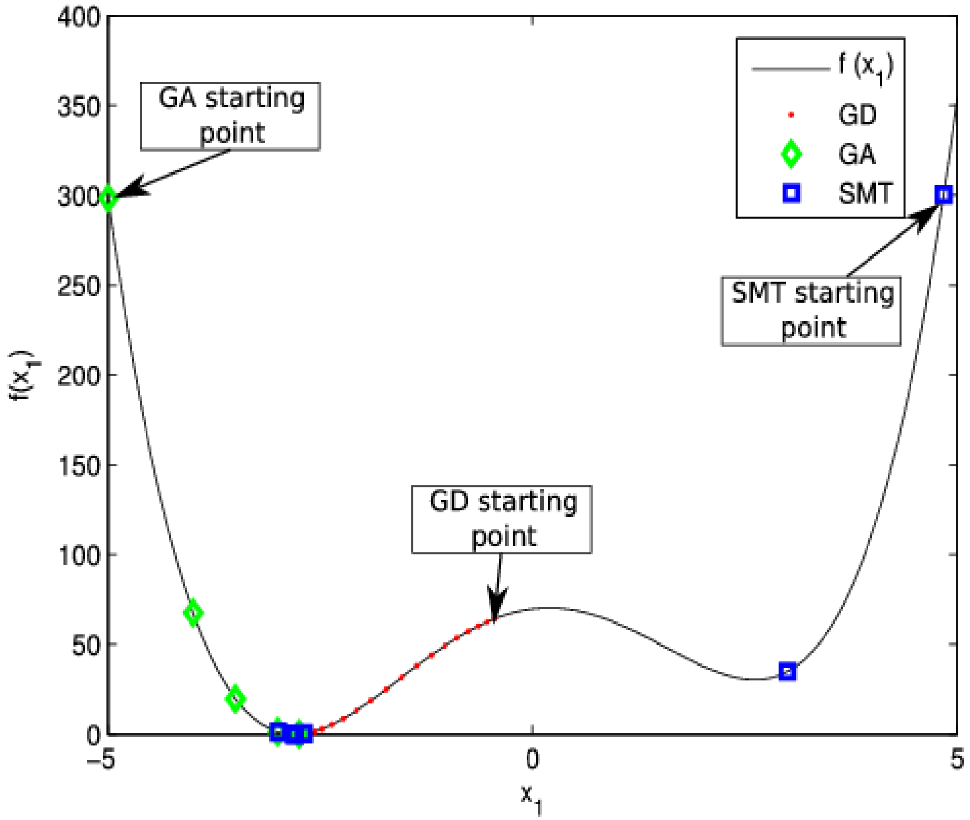
Illustrative Example

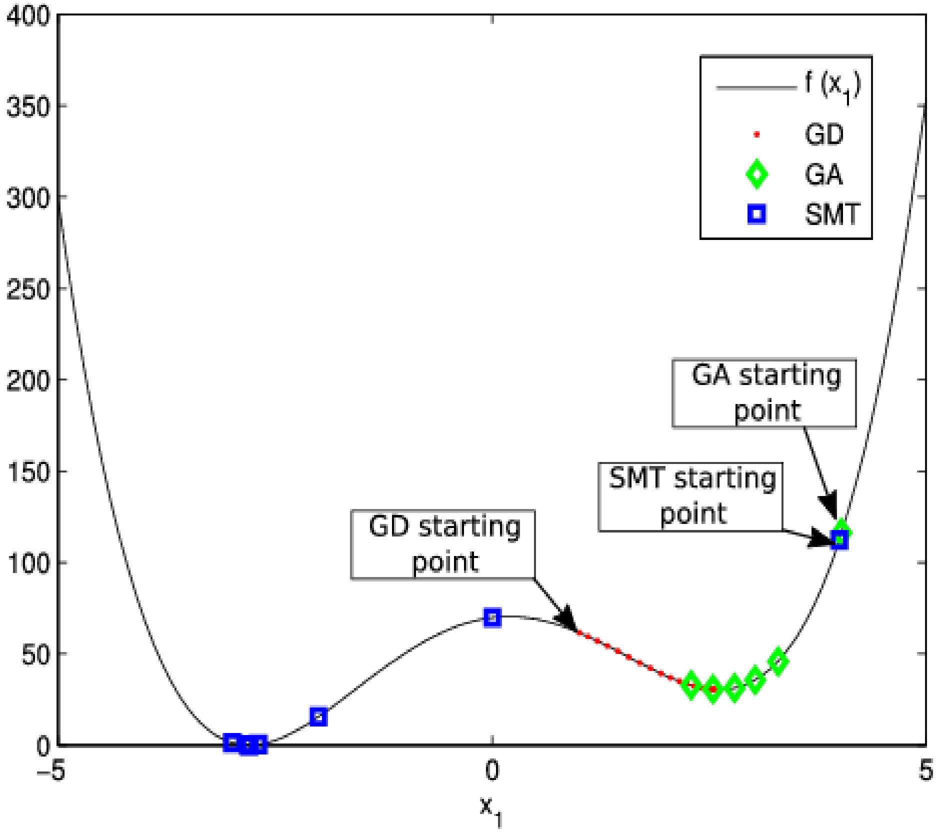
```c
int nondet_int();
int main(){
    int p = 1; //precision variable
    float f_ant = 100; // f_ant: previous obj function value
    int v = (int)(f_ant*p + 1);
    int X1 = nondet_int();
    int X2 = nondet_int();
    float x1, x2, fobj, fc;
    assume((X1>=-7*p) && (X1<=0*p));
    assume((X2>=0*p) && (X2<=7*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);
    assume( fobj < f_ant );
    for (int i = 0; i <= v; i++){
        fc = (float) i/p;
        assert( fobj > fc );
    }
    return 0;
}
```

If the assertion is maintained, then the optimal value is already known

Local minima avoidance

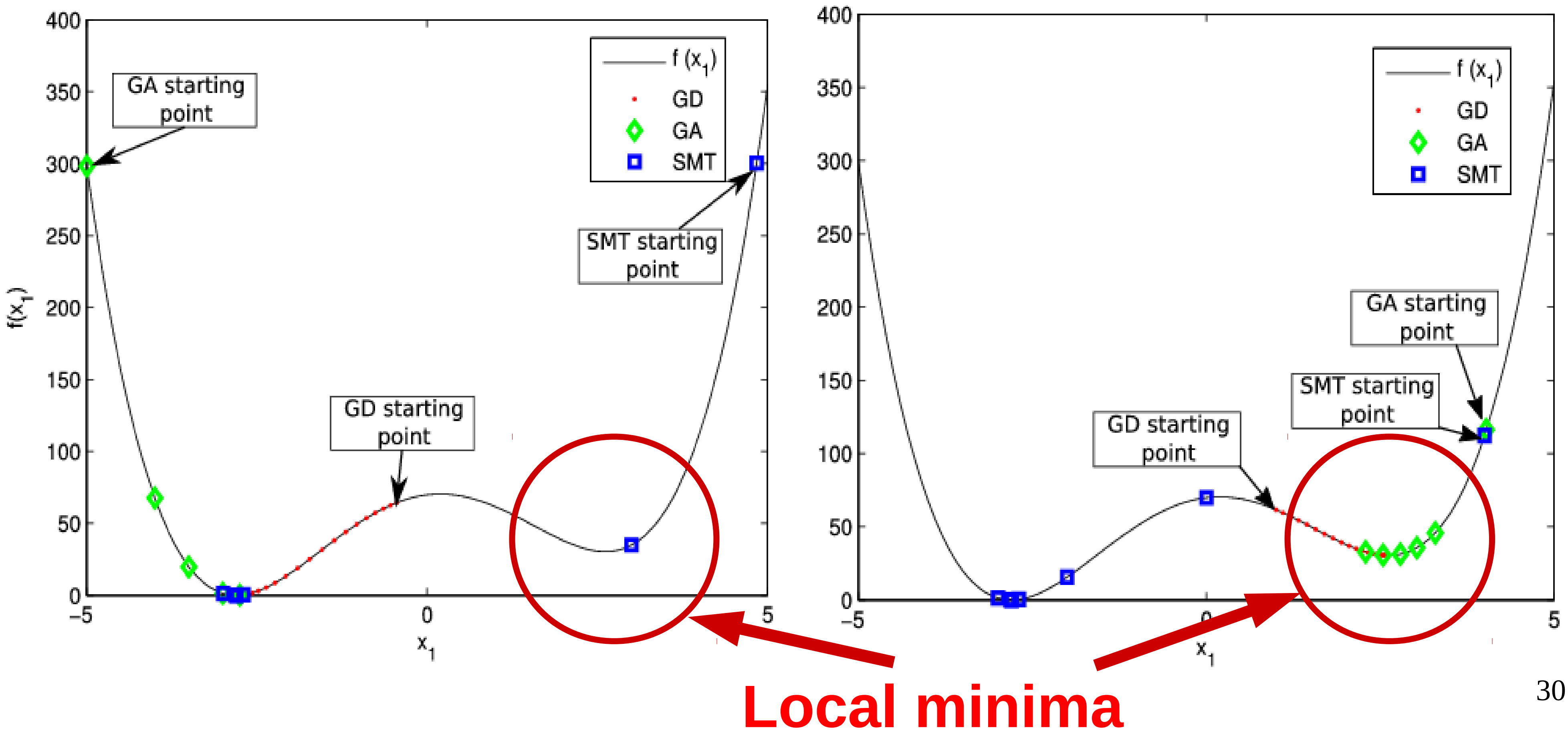
- Metaheuristic techniques depend on initialization and cannot ensure the global optimization





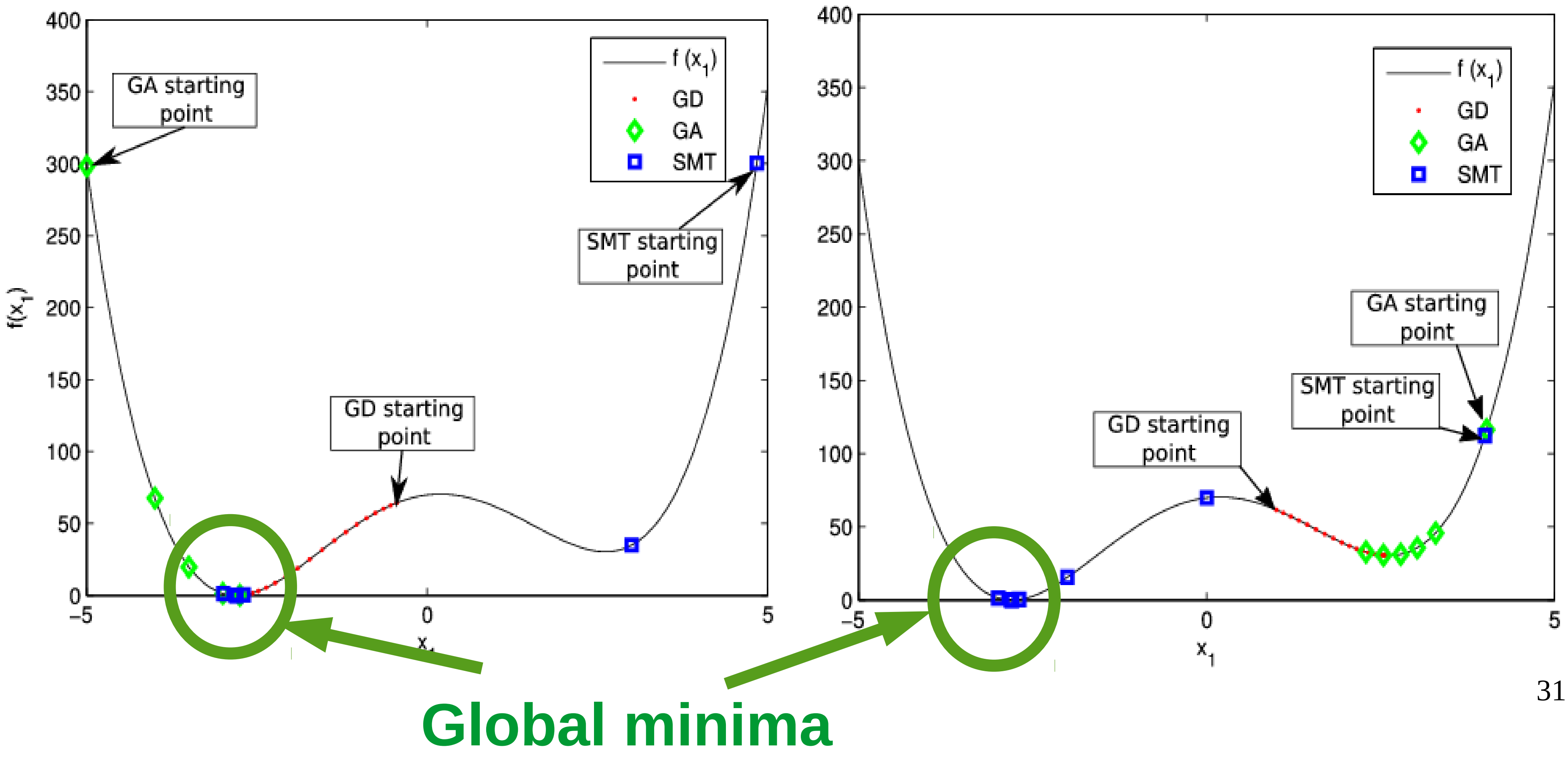
Local minima avoidance

- Metaheuristic techniques depend on initialization and cannot ensure the global optimization



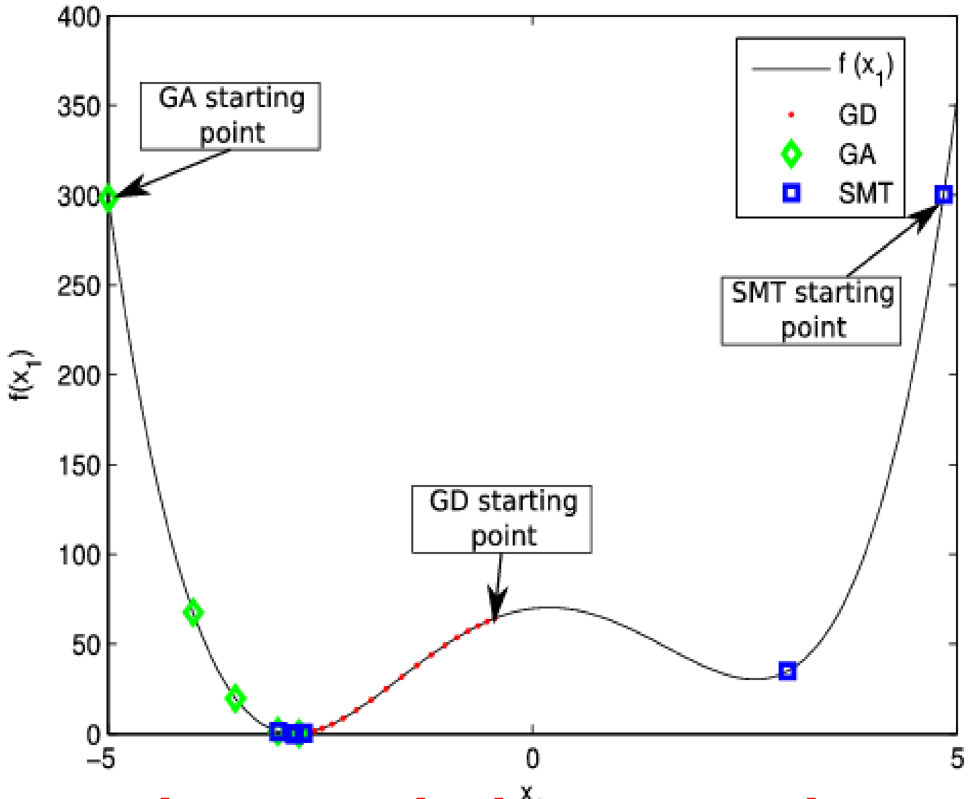
Local minima avoidance

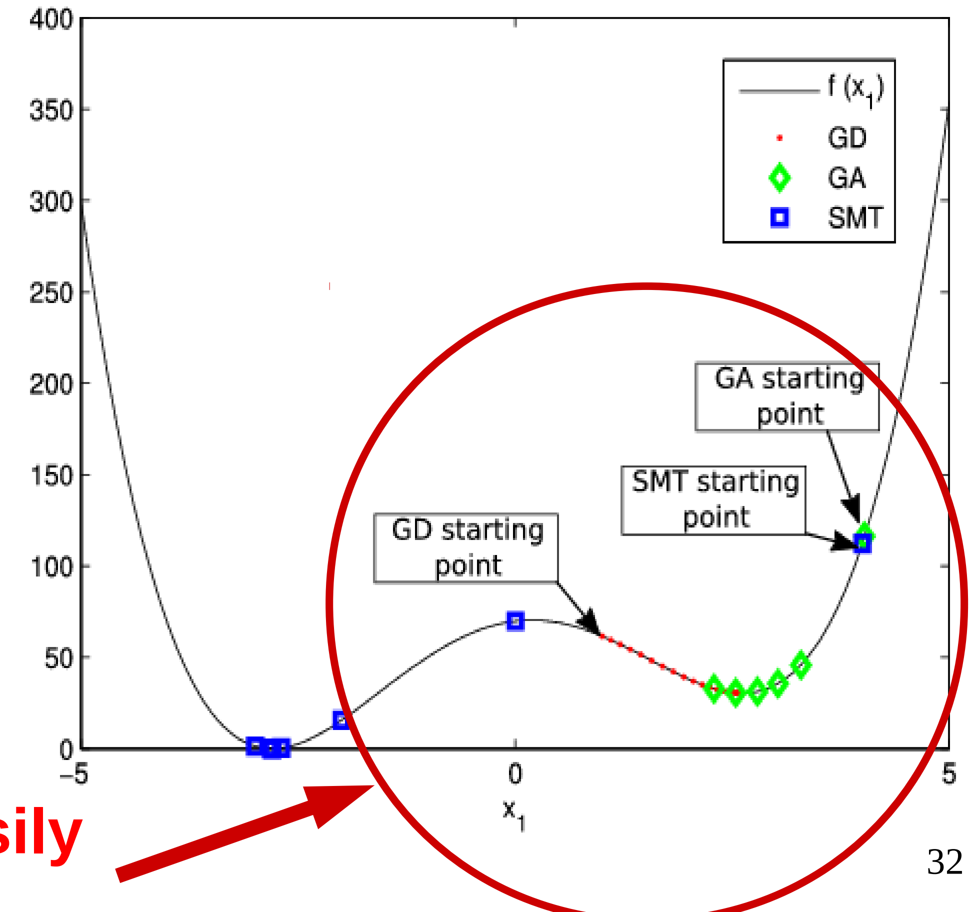
- Metaheuristic techniques depend on initialization and cannot ensure the global optimization



Local minima avoidance

- Metaheuristic techniques depend on initialization and cannot ensure the global optimization

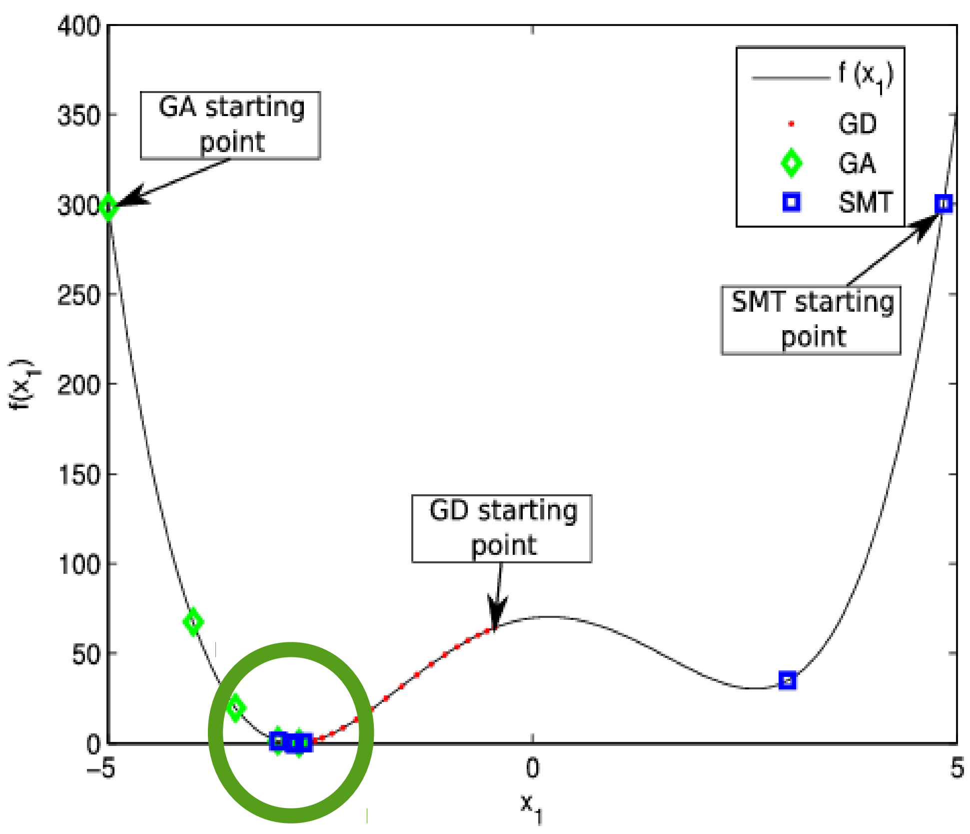


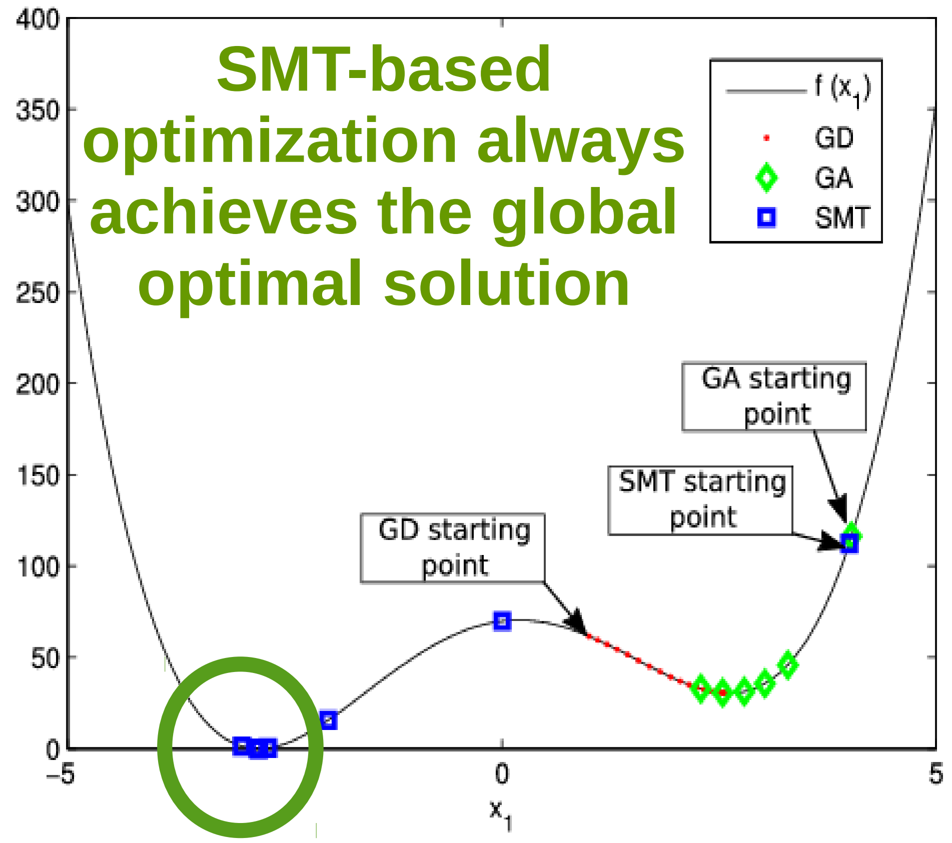


These techniques can be easily trapped by local minima

Local minima avoidance

- Metaheuristic techniques depend on initialization and cannot ensure the global optimization





Experimental Evaluation

- Objectives:
  - Check the performance of the SMT-based optimization algorithm
  - Compare with other traditional optimization methods
- Three functions are employed for evaluating our present method:
  - Himmelblau
  - Styblinski-Tang
  - Goldstein-Price

Experimental Evaluation

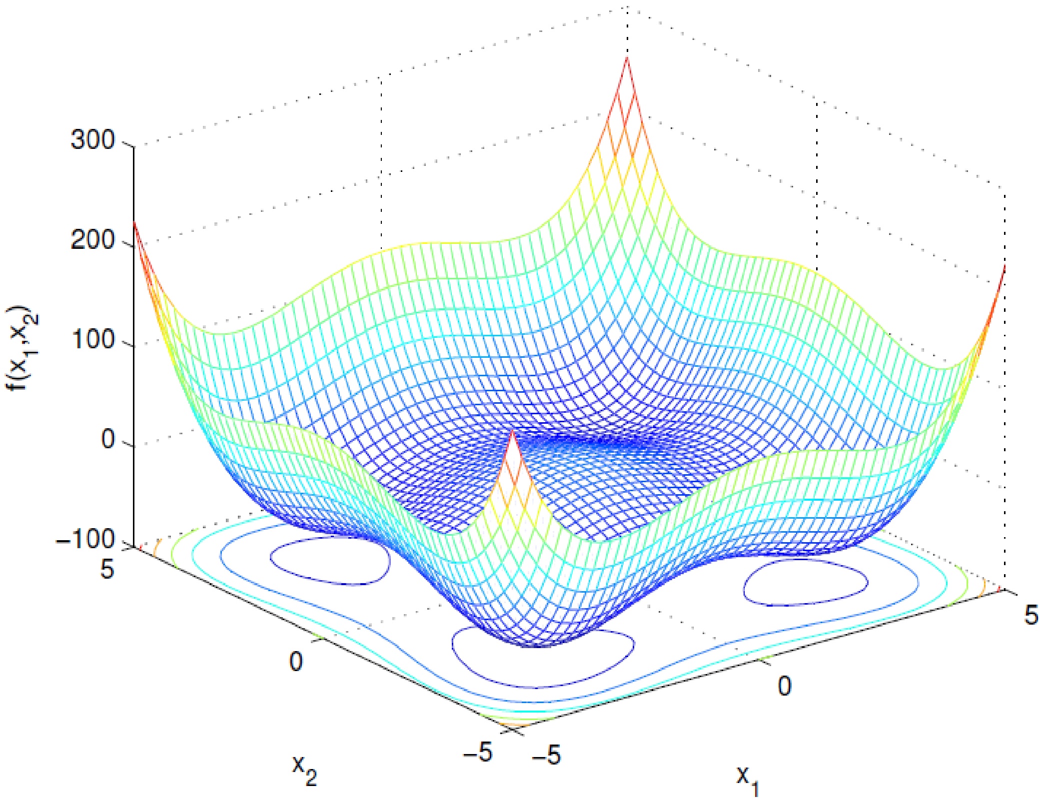
- The SMT-based optimization is compared to other two traditional techniques (genetic algorithm and gradient descent)
- Genetic algorithm (GA)
  - Population: 10
  - Generations: 50
- Gradient descent (GD)
  - Stop criteria: gradient less than 0.1
  - Learning rate: 0.01 (5e-5 for Goldstein-Price)

Experimental Setup

- Model checker: ESBMC 3.0 64-bits
- SMT Solver: Boolector v2.1.1
- Fedora 21 64-bits
- Dell Inspiron 5000, 16 GB RAM, Intel i7-5500U 3 GHz
- The time for the GA and GD are measured using an appropriate MATLAB function
- The time for the SMT-based optimization technique is measured with the UNIX time command

Stiblinski-Tang's function

$$f(x_1, x_2) = \frac{1}{2}\left(x_1^4 - 16x_1^2 + 5x_1 + x_2^4 - 16x_2^2 + 5x_2\right)$$



- Optimum point:

$$\mathbf{x}^* = (-2.903, -2.903)$$

$$f(\mathbf{x}^*) = -78.332$$

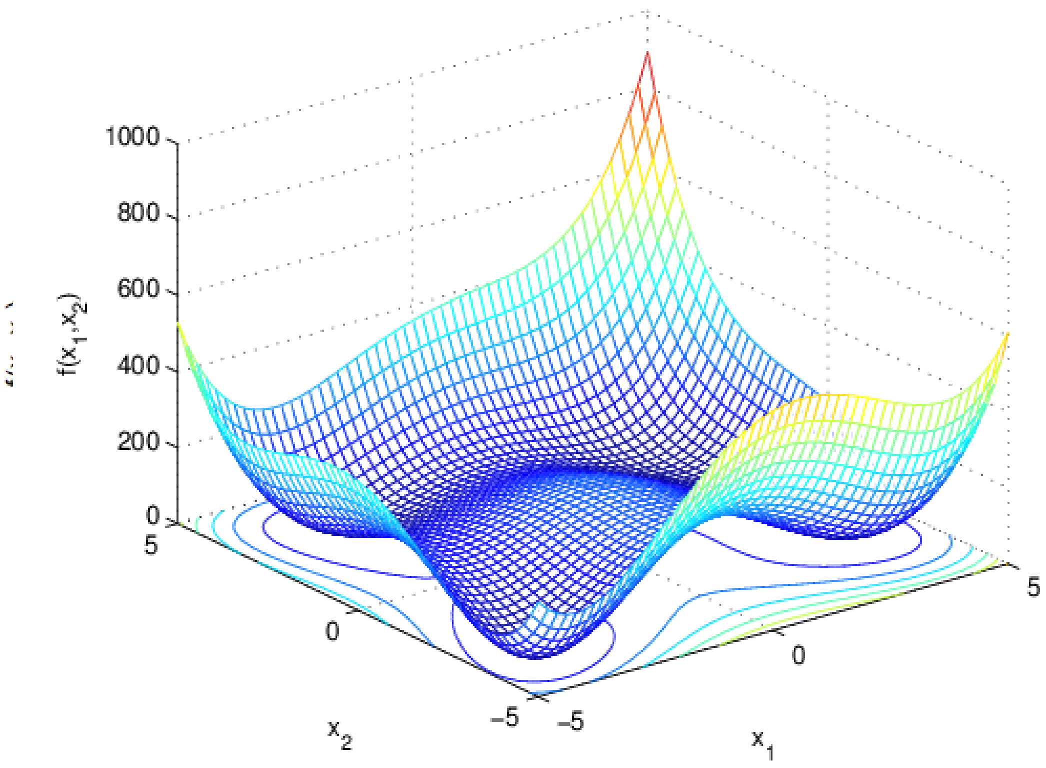
- Domain:

$$x_1 \in [-5, 5]$$

$$x_2 \in [-5, 5]$$

Himmelblau's function #1

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



- Optimum point:

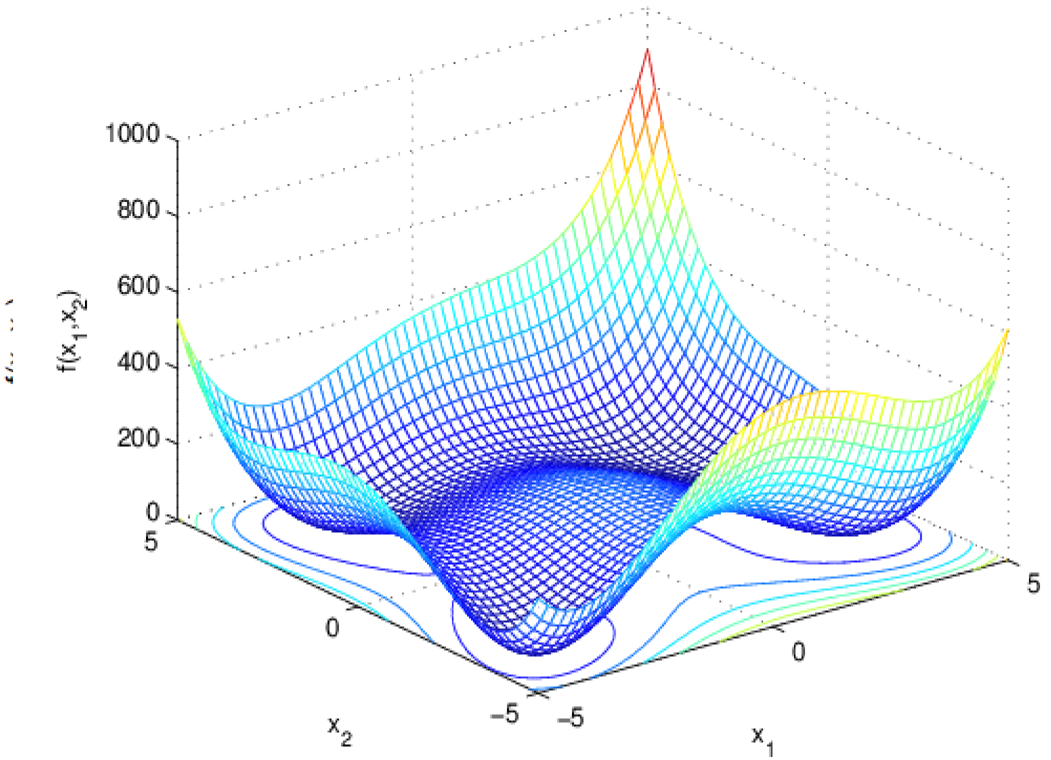
$$\mathbf{x}^* = (-2.805, 3.131)$$
$$f(\mathbf{x}^*) = 0$$

- Domain:

$$x_1 \in [-7, 0]$$
$$x_2 \in [0, 7]$$

Himmelblau's function #2

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



- Optima points:

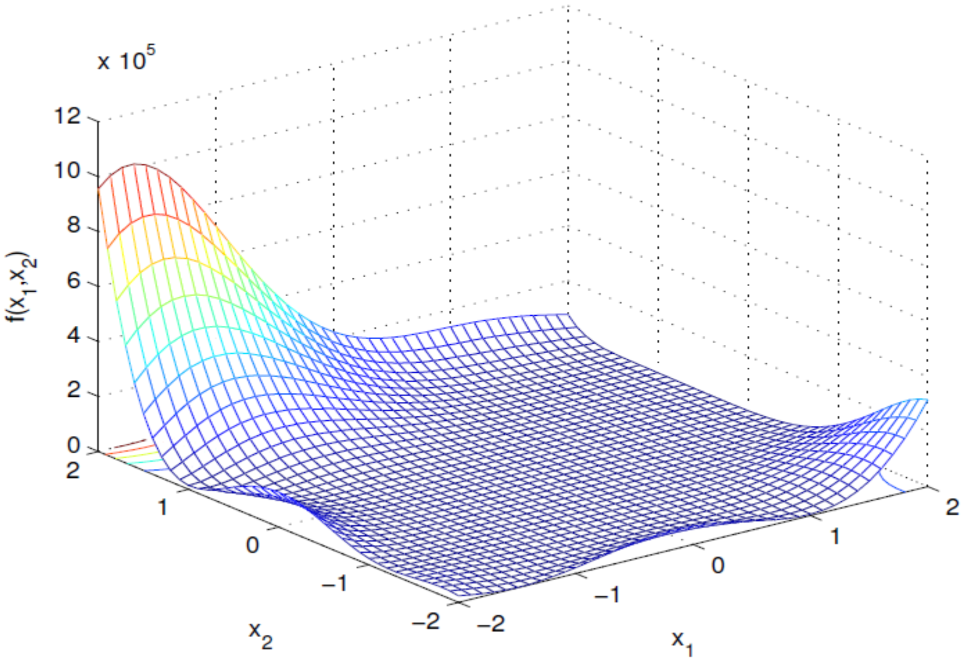
$$\mathbf{x}^* = (3, 2)$$
$$\mathbf{x}^* = (-2.805, 3.131)$$
$$\mathbf{x}^* = (-3.779, -3.283)$$
$$\mathbf{x}^* = (3.584, -1.848)$$
$$f(\mathbf{x}^*) = 0$$

- Domain:

$$x_1 \in [-5, 5]$$
$$x_2 \in [-5, 5]$$

Goldstein-Price's function

$$f(x_1, x_2) = [1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)]$$
$$[30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)]$$

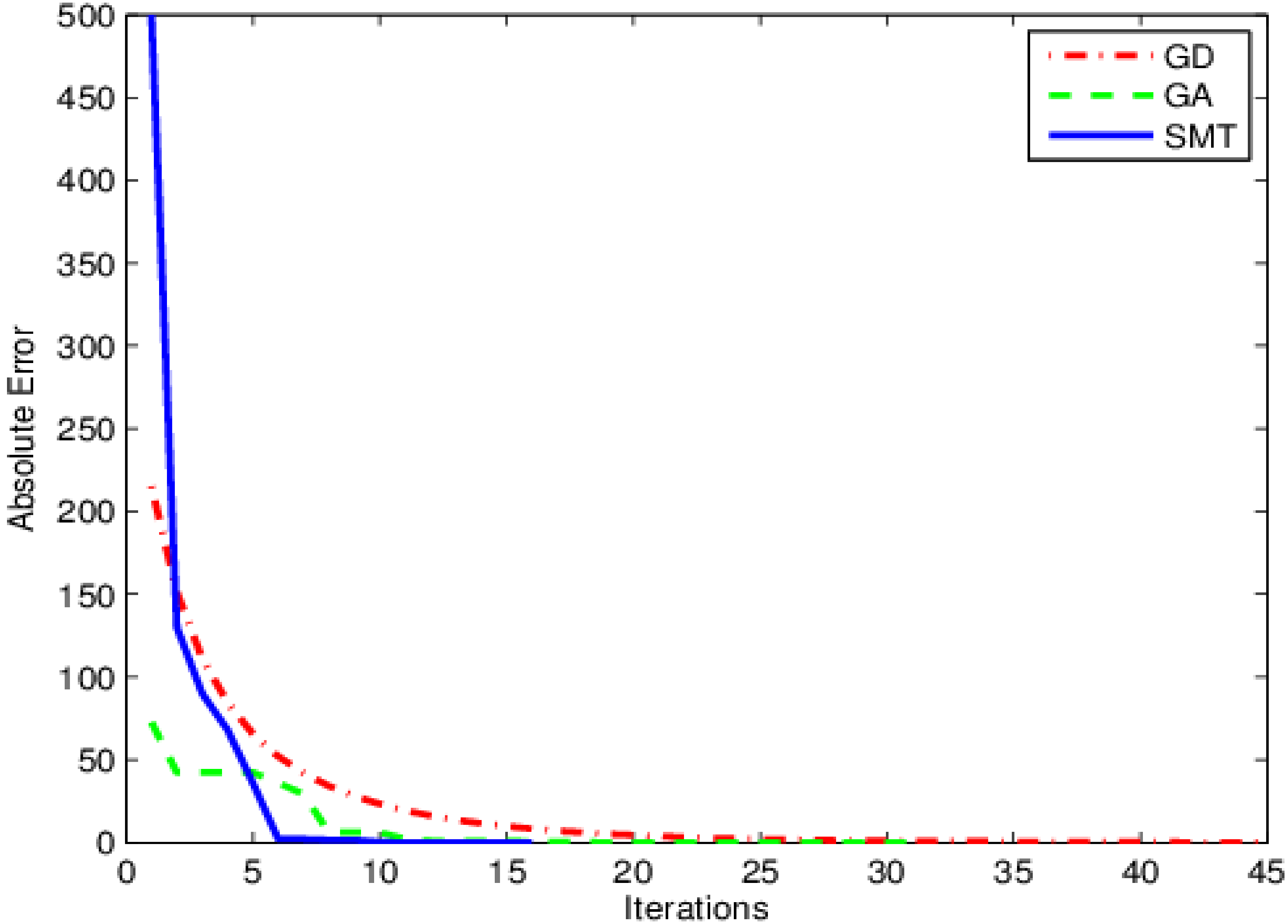


- Optimum point:
  $$\mathbf{x}^* = (0, -1)$$
  $$f(\mathbf{x}^*) = 3$$
- Domain:
  $$x_1 \in [-2, 2]$$
  $$x_2 \in [-2, 2]$$

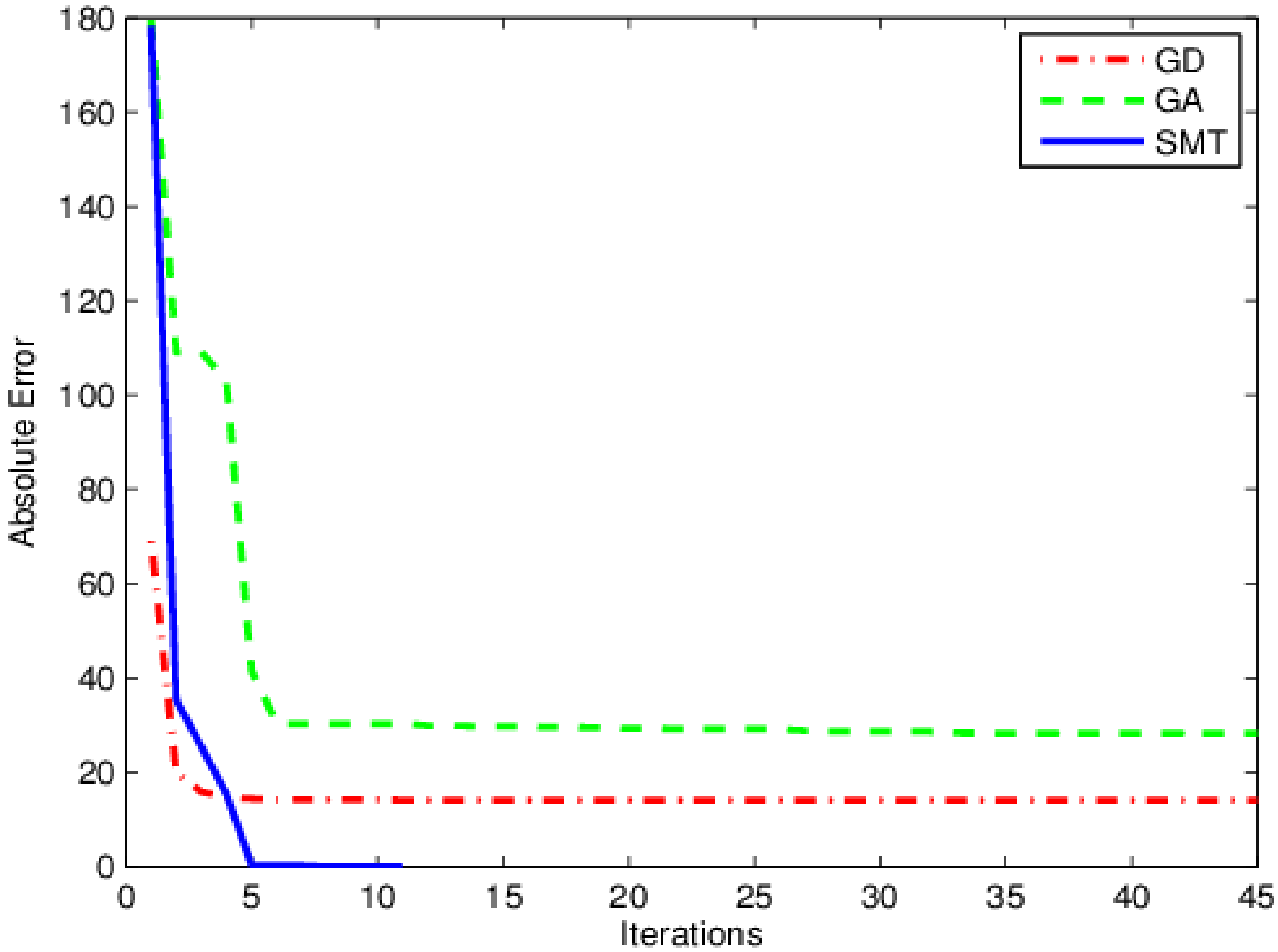
Experimental Results

Function	Method	Correct Answer (%)	Execution Time (s)
Himmelblau #1	GD	55	<1
	GA	**100**	**<1**
	SMT	100	1622
Himmelblau #2	**GD**	**100**	**<1**
	GA	**100**	**<1**
	SMT	100	4
Styblinski-Tang	GD	21	<1
	GA	9	<1
	SMT	**100**	**1045**
Goldstein-Price	GD	0	1
	GA	69	<1
	SMT	**100**	**14**

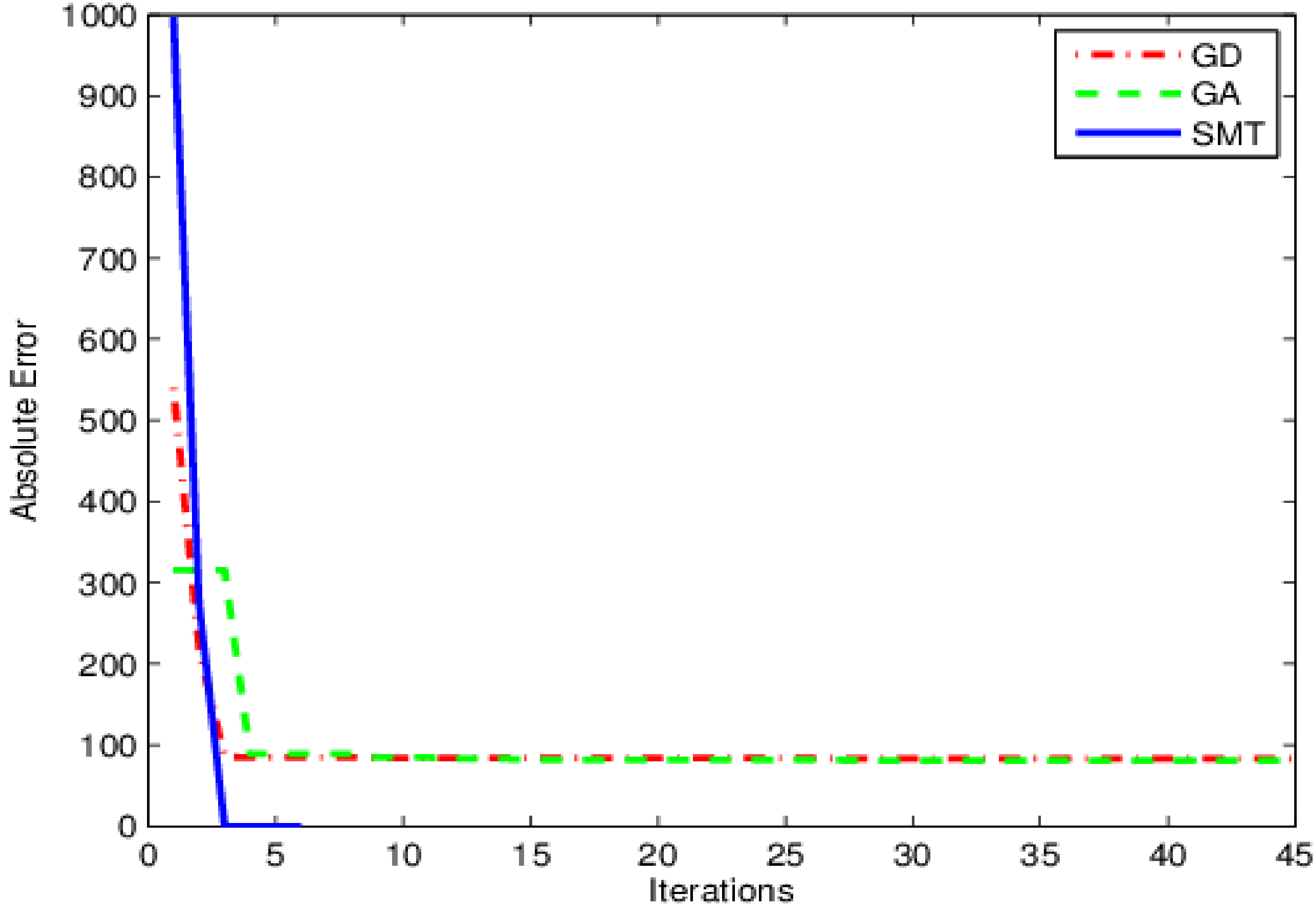
Absolute error X iteration (Himmelblau #2)



Absolute error X iteration (Styblinski-Tang)



Absolute error X iteration (Goldstein-Price's)



Conclusions

- We presented an SMT-based optimization method applied to nonconvex optimization problems
- The proposal ensures the global optimization but it takes longer time than GD and GA
- SMT-based optimization is a flexible technique and can be used for any class of function
- Further work:
  - Multiobjective optimization
  - UAV trajectory planning and mission planning
  - Parallelize the optimization process