Verificare Formala

Aplicatii ale SMT solving

Mădălina Erașcu

West University of Timișoara
Faculty of Mathematics and Informatics
Department of Computer Science

Bazat pe suportul de curs: *Satisfiability Checking (Erika Ábrahám), RTWH Aachen*
Bazat pe cartile: (1) The calculus of computation - Z. Manna, A. Bradley; (2) Decision Procedures - An algorithmic Point of View - D. Kroening, O. Strichman

WS 2023/2024

Outline

Contents

Optimization

General form:
Minimize/maximize $f_1(x, y, ...), f_2(x, y, ...), ..., f_n(x, y, ...)$ subject to $C(x, y, ...)$.

Optimization

General form:
Minimize/maximize $f_1(x, y, ...), f_2(x, y, ...), ..., f_n(x, y, ...)$ subject to $C(x, y, ...)$.

- Single optimization: $n \geq 1$

Optimization

General form:
Minimize/maximize $f_1(x, y, ...), f_2(x, y, ...), ..., f_n(x, y, ...)$ subject to $C(x, y, ...)$.

- Single optimization: $n \geq 1$
- Multiple optimization: $n \geq 2$

Optimization

General form:
Minimize/maximize $f_1(x, y, ...), f_2(x, y, ...), ..., f_n(x, y, ...)$ subject to $C(x, y, ...)$.

- Single optimization: $n \geq 1$
- Multiple optimization: $n \geq 2$

In the SMT solver Z3 (https://github.com/Z3Prover/z3), there are three ways to combine objective functions (assume we want to maximize):

Optimization

General form:
Minimize/maximize $f_1(x, y, ...), f_2(x, y, ...), ..., f_n(x, y, ...)$ subject to $C(x, y, ...)$.

- Single optimization: $n \geq 1$
- Multiple optimization: $n \geq 2$

In the SMT solver Z3 (https://github.com/Z3Prover/z3), there are three ways to combine objective functions (assume we want to maximize):

1. $Box(x, y): \quad v_x := max\{x|\varphi(x, y)\},\ v_y := max\{y|\varphi(x, y)\}$

Optimization

General form:
Minimize/maximize $f_1(x, y, ...), f_2(x, y, ...), ..., f_n(x, y, ...)$ subject to $C(x, y, ...)$.

- Single optimization: $n \geq 1$
- Multiple optimization: $n \geq 2$

In the SMT solver Z3 (https://github.com/Z3Prover/z3), there are three ways to combine objective functions (assume we want to maximize):

1. $Box(x, y): \quad v_x := max\{x|\varphi(x, y)\},\ v_y := max\{y|\varphi(x, y)\}$
2. $Lex(x, y): \quad v_x := max\{x|\varphi(x, y)\},\ v_y := max\{y|\varphi(v_x, y)\}$

Optimization

General form:
Minimize/maximize $f_1(x, y, ...), f_2(x, y, ...), ..., f_n(x, y, ...)$ subject to $C(x, y, ...)$.

- Single optimization: $n \geq 1$
- Multiple optimization: $n \geq 2$

In the SMT solver Z3 (https://github.com/Z3Prover/z3), there are three ways to combine objective functions (assume we want to maximize):

1. $Box(x, y): \quad v_x := max\{x|\varphi(x, y)\},\ v_y := max\{y|\varphi(x, y)\}$
2. $Lex(x, y): \quad v_x := max\{x|\varphi(x, y)\},\ v_y := max\{y|\varphi(v_x, y)\}$

Optimization

General form:
Minimize/maximize $f_1(x, y, ...), f_2(x, y, ...), ..., f_n(x, y, ...)$ subject to $C(x, y, ...)$.

- Single optimization: $n \geq 1$
- Multiple optimization: $n \geq 2$

In the SMT solver Z3 (https://github.com/Z3Prover/z3), there are three ways to combine objective functions (assume we want to maximize):

1. $Box(x, y): \quad v_x := max\{x|\varphi(x, y)\},\ v_y := max\{y|\varphi(x, y)\}$
2. $Lex(x, y): \quad v_x := max\{x|\varphi(x, y)\},\ v_y := max\{y|\varphi(v_x, y)\}$

where x, y are the decision variables, $\varphi(x, y)$ are the constraints to be fulfilled.

Optimization

General form:
Minimize/maximize $f_1(x, y, ...), f_2(x, y, ...), ..., f_n(x, y, ...)$ subject to $C(x, y, ...)$.

- Single optimization: $n \geq 1$
- Multiple optimization: $n \geq 2$

In the SMT solver Z3 (https://github.com/Z3Prover/z3), there are three ways to combine objective functions (assume we want to maximize):

1. $Box(x, y): \quad v_x := max\{x|\varphi(x, y)\},\ v_y := max\{y|\varphi(x, y)\}$
2. $Lex(x, y): \quad v_x := max\{x|\varphi(x, y)\},\ v_y := max\{y|\varphi(v_x, y)\}$

where x, y are the decision variables, $\varphi(x, y)$ are the constraints to be fulfilled.
More details at
`https://theory.stanford.edu/~nikolaj/programmingz3.html`.

Optimization (cont'd)

In Z3, the default optimization is lexicographic.
One can set up other types of optimization by adding (SMT-LIB version, but there exists also the corresponding commands for Python API):

- (set-option :opt.priority pareto)
- (set-option :opt.priority box)

Pareto Optimization

In multi-objective optimization, it is typically the the case that the objective functions are competing.

Pareto Optimization

In multi-objective optimization, it is typically the the case that the objective functions are competing.
In this case, we talk about a set of optimal solutions instead of one optimal solution since no solution is better than the other.

Pareto Optimization

In multi-objective optimization, it is typically the the case that the objective functions are competing.
In this case, we talk about a set of optimal solutions instead of one optimal solution since no solution is better than the other.
These optimal solutions are called Pareto-optimal solutions.

Pareto Optimization

In multi-objective optimization, it is typically the the case that the objective functions are competing.
In this case, we talk about a set of optimal solutions instead of one optimal solution since no solution is better than the other.
These optimal solutions are called Pareto-optimal solutions.
Given a set of feasible solutions and different objective functions, Pareto improvement is a movement from one feasible solution to another that can make at least one objective function to return a better value with no other objective function becoming worse.

Pareto Optimization

In multi-objective optimization, it is typically the the case that the objective functions are competing.

In this case, we talk about a set of optimal solutions instead of one optimal solution since no solution is better than the other.

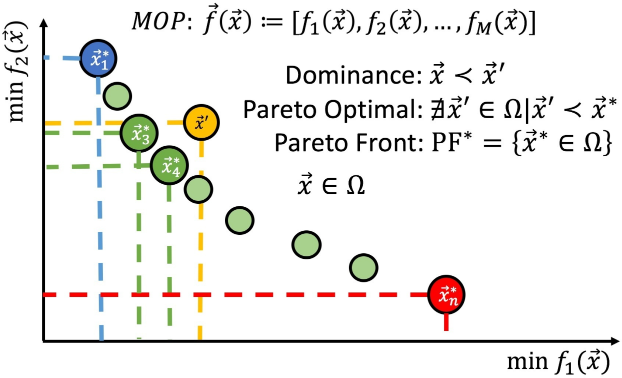
These optimal solutions are called Pareto-optimal solutions.

Given a set of feasible solutions and different objective functions, Pareto improvement is a movement from one feasible solution to another that can make at least one objective function to return a better value with no other objective function becoming worse.

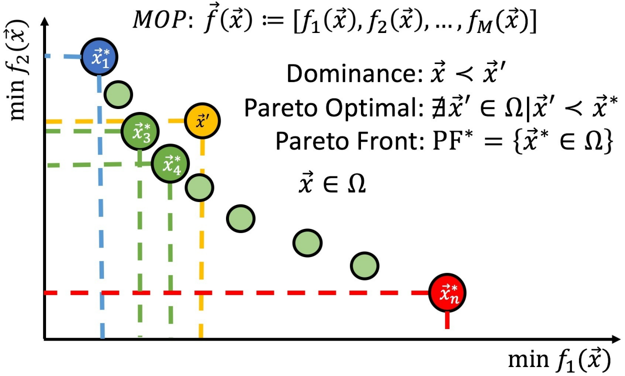
A set of feasible solutions are Pareto efficient/optimal when no further Pareto improvements can be made.

Pareto Optimality

Solutions along the line are all non-dominated solutions.

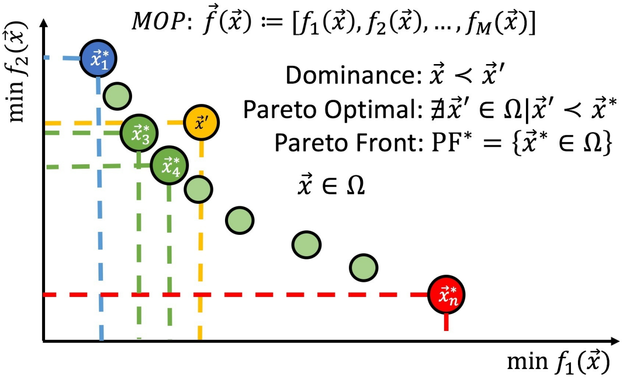


Solutions along the line are all non-dominated solutions.
Dominated solutions are inside the line as there is another solution on the line with at least one objective that is better.



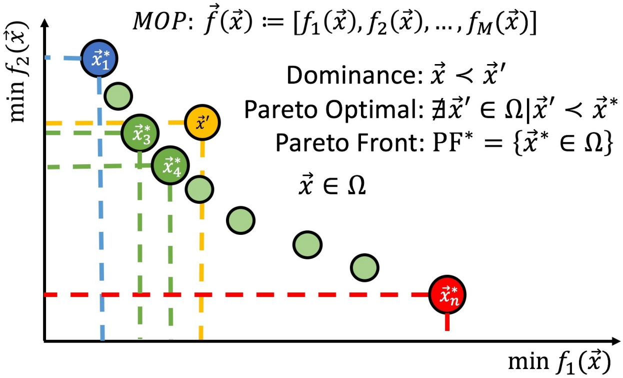
Pareto Optimality

Solutions along the line are all non-dominated solutions.
Dominated solutions are inside the line as there is another solution on the line with at least one objective that is better.
The line is the Pareto-optimal front and the solutions on it are called Pareto-optimal.



Pareto Optimality

Solutions along the line are all non-dominated solutions.
Dominated solutions are inside the line as there is another solution on the line with at least one objective that is better.
The line is the Pareto-optimal front and the solutions on it are called Pareto-optimal. All Pareto optimal solutions are non-dominated.



Contents

Virtual Machine Placement Problem

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

1. Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)
2. Minimize the number of servers used.

Virtual Machine Placement Problem

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

1. Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)
2. Minimize the number of servers used.

Solution. We first formalize the problem. Then we can translate it into the SMT-LIB format and use an SMT solver to find a solution.

Virtual Machine Placement Problem

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

1. Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)
2. Minimize the number of servers used.

Solution. We first formalize the problem. Then we can translate it into the SMT-LIB format and use an SMT solver to find a solution.
Let x_{ij} denote that VM i is placed on the server j and y_j denote that server j is in use.
We need to express the followings.

Virtual Machine Placement Problem (cont'd)

1. Implicit constraints

2. Explicit constraints

3. Optimization functions

Virtual Machine Placement Problem (cont'd)

1. Implicit constraints
   - A VM is on exactly one server:

$$x_{i1} + x_{i2} + x_{i3} = 1, \quad \forall i, j = \overline{1,3}$$

2. Explicit constraints

3. Optimization functions

Virtual Machine Placement Problem (cont'd)

1. Implicit constraints
   - A VM is on exactly one server:

$$x_{i1} + x_{i2} + x_{i3} = 1, \quad \forall i, j = \overline{1,3}$$

   - A used server has at least a VM on it:

$$(x_{1j} = 1) \vee ... \vee (x_{3j} = 1) \Rightarrow (y_j = 1), j = \overline{1,3}$$

2. Explicit constraints

3. Optimization functions

Virtual Machine Placement Problem (cont'd)

1. Implicit constraints
   - A VM is on exactly one server:

$$x_{i1} + x_{i2} + x_{i3} = 1, \quad \forall i, j = \overline{1,3}$$

   - A used server has at least a VM on it:

$$(x_{1j} = 1) \vee ... \vee (x_{3j} = 1) \Rightarrow (y_j = 1), j = \overline{1,3}$$

2. Explicit constraints
   - Capacity constraints:
     $100x_{11} + 50x_{21} + 15x_{31} \leq 100y_1$
     $100x_{12} + 50x_{22} + 15x_{32} \leq 75y_2$
     $100x_{13} + 50x_{23} + 15x_{33} \leq 200y_3$
3. Optimization functions

Virtual Machine Placement Problem (cont'd)

1. Implicit constraints
   - A VM is on exactly one server:

$$x_{i1} + x_{i2} + x_{i3} = 1, \quad \forall i, j = \overline{1,3}$$

   - A used server has at least a VM on it:

$$(x_{1_j} = 1) \vee ... \vee (x_{3_j} = 1) \Rightarrow (y_j = 1), j = \overline{1,3}$$

2. Explicit constraints
   - Capacity constraints:
     $100x_{11} + 50x_{21} + 15x_{31} \leq 100y_1$
     $100x_{12} + 50x_{22} + 15x_{32} \leq 75y_2$
     $100x_{13} + 50x_{23} + 15x_{33} \leq 200y_3$
3. Optimization functions
   - $10y_1 + 5y_2 + 20y_3$

Virtual Machine Placement Problem (cont'd)

1. Implicit constraints
   - A VM is on exactly one server:

$$x_{i1} + x_{i2} + x_{i3} = 1, \quad \forall i, j = \overline{1,3}$$

   - A used server has at least a VM on it:

$$(x_{1j} = 1) \vee \ldots \vee (x_{3j} = 1) \Rightarrow (y_j = 1), j = \overline{1,3}$$

2. Explicit constraints
   - Capacity constraints:
     $100x_{11} + 50x_{21} + 15x_{31} \leq 100y_1$
     $100x_{12} + 50x_{22} + 15x_{32} \leq 75y_2$
     $100x_{13} + 50x_{23} + 15x_{33} \leq 200y_3$
3. Optimization functions
   - $10y_1 + 5y_2 + 20y_3$
   - $y_1 + y_2 + y_3$

Virtual Machine Placement Problem (cont'd)

There are various ways of encoding the variables of the problem, for example:

(Variant 1) as integers

(Variant 2) as real

(Variant 3) as bool

(Variant 4) using `assert-soft` constraints

Virtual Machine Placement Problem (cont'd)

There are various ways of encoding the variables of the problem, for example:

(Variant 1) as integers

(Variant 2) as real

(Variant 3) as bool

(Variant 4) using `assert-soft` constraints

Variant 1.

Virtual Machine Placement Problem (cont'd)

There are various ways of encoding the variables of the problem, for example:

(Variant 1) as integers

(Variant 2) as real

(Variant 3) as bool

(Variant 4) using `assert-soft` constraints

Variant 1. We declare each variable as integer, e.g.:

```
(declare-const x11 Int)
```

We also need to ensure that variables are 0/1, e.g.:

```
(assert (and (>= x11 0) (>= x12 0) (>= x13 0) (>= x21 0)...
(assert (and (<= y1 1) (<= y2 1) (<= y3 1)))
```

Another variant for encoding the 0/1 integers is:

```
(assert (or (>= x11 0) (<= x11 1)
(assert (or (>= x12 0) (<= x12 1)...
```

Virtual Machine Placement Problem (cont'd)

There are various ways of encoding the variables of the problem, for example:

(Variant 1) as integers

(Variant 2) as real

(Variant 3) as bool

(Variant 4) using `assert-soft` constraints

Variant 1. Constraint of type 2 can be encoded in 2 ways. For example:

```
(assert (and (>= y1 x11) (>= y1 x21) (>= y1 x31)))
...
```

Or as:

```
(assert (implies (= y1 1) (or (= x11 1) (= x21 1) (= x31 1) ))
...
```

Virtual Machine Placement Problem (cont'd)

There are various ways of encoding the variables of the problem, for example:

(Variant 1) as integers

(Variant 2) as real

(Variant 3) as bool

(Variant 4) using `assert-soft` constraints

Variant 2. We declare each variable as real. The constraints should be the same as for the integer encoding.

Virtual Machine Placement Problem (cont'd)

There are various ways of encoding the variables of the problem, for example:

(Variant 1) as integers

(Variant 2) as real

(Variant 3) as bool

(Variant 4) using `assert-soft` constraints

Variant 3. We declare each variable as bool. The capability constraints require only integer/real variables, so we need to transform the bool variables into integer/real. This can be done by declaring a function as follows:

```
(define-fun bool_to_int ((b Bool)) Int (ite b 1 0) )
```

and cast the bool variables to int/real, e.g.

```
(assert (<= (+ (* 100 (bool_to_int x11)) ...  )  (...)))
```

Virtual Machine Placement Problem (cont'd)

There are various ways of encoding the variables of the problem, for example:

(Variant 1) as integers

(Variant 2) as real

(Variant 3) as bool

(Variant 4) using `assert-soft` constraints

Variant 4. Use `assert-soft` constraints (soft constraints) (see `https://rise4fun.com/Z3/tutorialcontent/optimization`). For our problem, soft constraints can be used to encode the optimization goals:

`(assert-soft (not y1) :id num_servers)` ...

`(assert-soft (not y1) :id costs :weight 10)` ...

The `assert-soft` command represents MaxSMT (maximize the number of constraints which can be satisfied) which tries to maximize the weighted sum of boolean expressions belonged to the same `id`. Since we are doing minimization, negation is needed to take advantage of MaxSMT support.