Formal Methods in Software Developement

Branch and Bound

Mădălina Erașcu

West University of Timișoara
Faculty of Mathematics and Informatics

Based on slides of the lecture Satisfiability Checking (Erika Ábrahám), RTWH Aachen

December 1, 2018

Integer linear systems

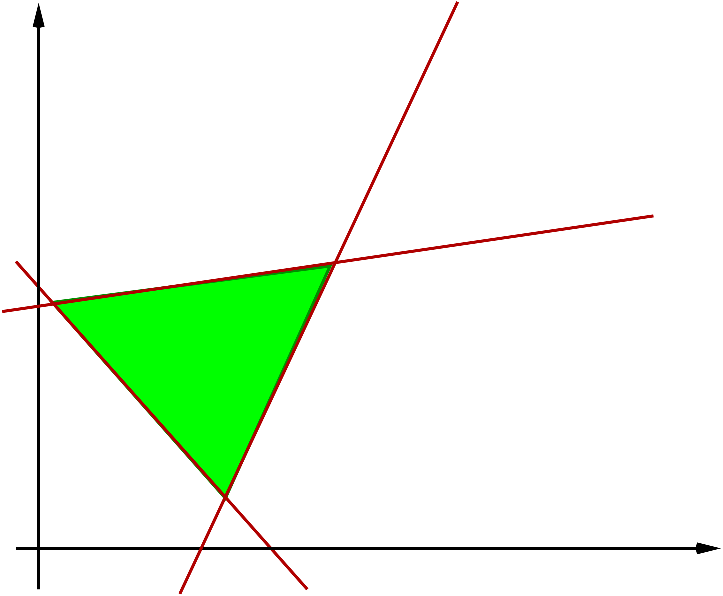
Definition

An integer linear system S is a linear system $Ax = 0$, $\bigwedge_{i=1}^{m} l_i \leq s_i \leq u_i$, with the additional integrality requirement that all variables are of type integer.

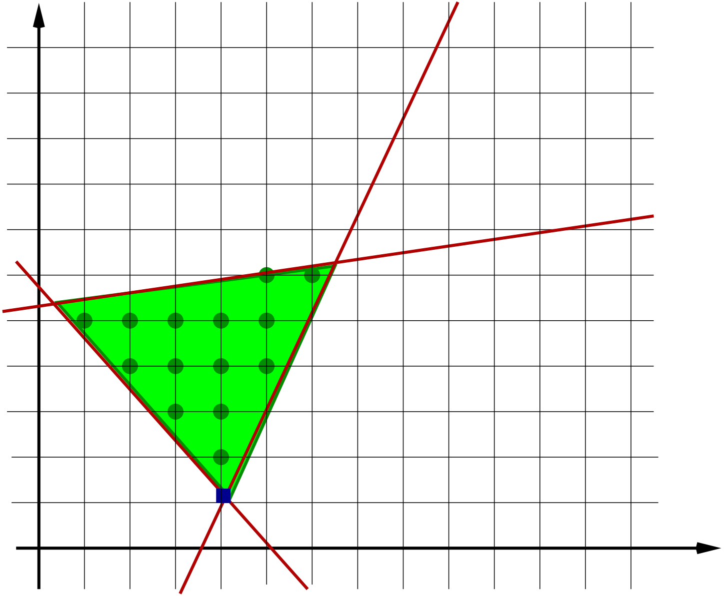
Definition (relaxed system)

Given an integer linear system S, its relaxation $relaxed(S)$ is S without the integrality requirement.

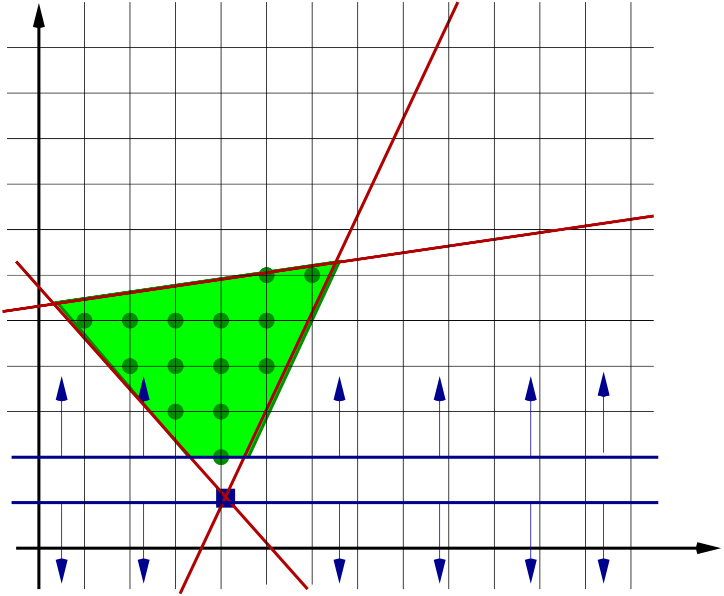
Geometric interpretation



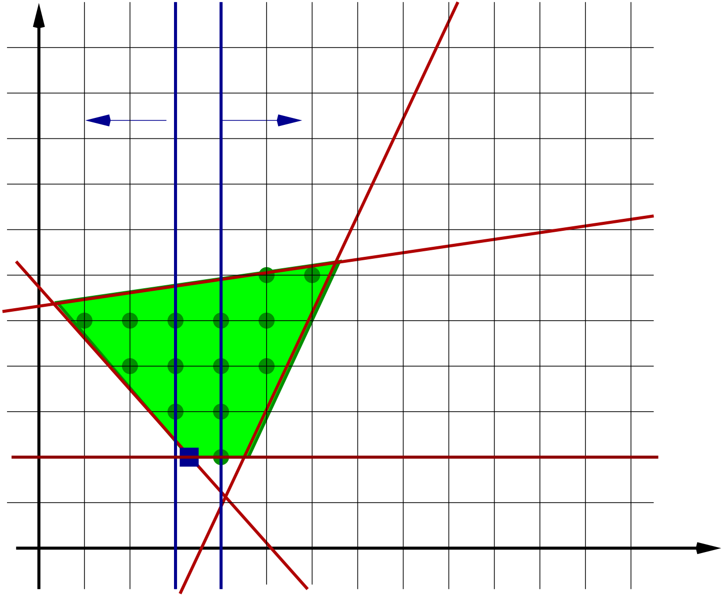
Geometric interpretation



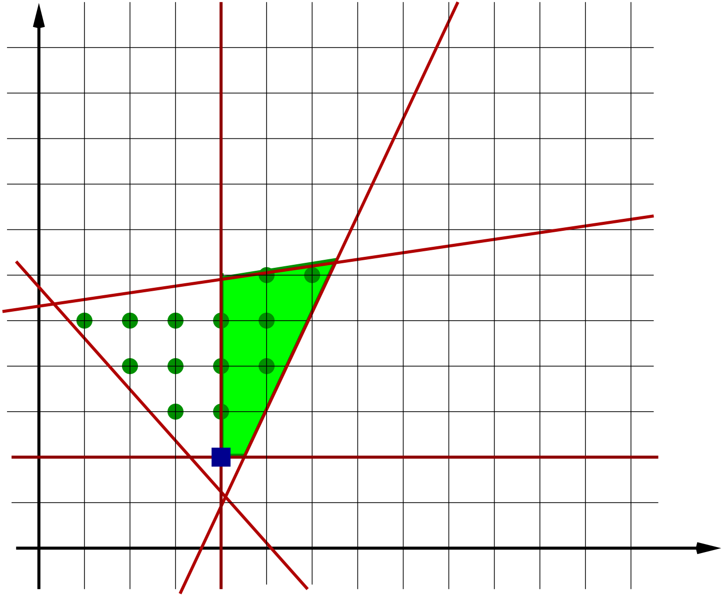
Geometric interpretation



Geometric interpretation



Geometric interpretation



Branch and bound algorithm

Input: An integer linear system S
Output: SAT if S is satisfiable, UNSAT otherwise

procedure Branch-and-Bound(S) {

Branch and bound algorithm

Input: An integer linear system S
Output: SAT if S is satisfiable, UNSAT otherwise

```
procedure Branch-and-Bound(S) {
  res = LP(relaxed(S));          /*LP could be Fourier-Motzkin*/
```

Branch and bound algorithm

Input: An integer linear system S
Output: SAT if S is satisfiable, UNSAT otherwise

```
procedure Branch-and-Bound(S) {
  res = LP(relaxed(S));          /*LP could be Fourier-Motzkin*/
  if (res==UNSAT) return UNSAT;
```

Branch and bound algorithm

Input: An integer linear system S
Output: SAT if S is satisfiable, UNSAT otherwise

```
procedure Branch-and-Bound(S) {
  res = LP(relaxed(S));          /*LP could be Fourier-Motzkin*/
  if (res==UNSAT) return UNSAT;
  else if (res is integral) return SAT;
  else {
```

Branch and bound algorithm

Input: An integer linear system S
Output: SAT if S is satisfiable, UNSAT otherwise

```
procedure Branch-and-Bound(S) {
  res = LP(relaxed(S));          /*LP could be Fourier-Motzkin*/
  if (res==UNSAT) return UNSAT;
  else if (res is integral) return SAT;
  else {
    Select a variable v that is assigned a non-integral value r;
```

Branch and bound algorithm

Input: An integer linear system S
Output: SAT if S is satisfiable, UNSAT otherwise

```
procedure Branch-and-Bound(S) {
  res = LP(relaxed(S));          /*LP could be Fourier-Motzkin*/
  if (res==UNSAT) return UNSAT;
  else if (res is integral) return SAT;
  else {
    Select a variable v that is assigned a non-integral value r;
    if (Branch-and-Bound(S ∪ (v ≤ ⌊r⌋))==SAT) return SAT;
```

Branch and bound algorithm

Input: An integer linear system S
Output: SAT if S is satisfiable, UNSAT otherwise

```
procedure Branch-and-Bound(S) {
  res = LP(relaxed(S));          /*LP could be Fourier-Motzkin*/
  if (res==UNSAT) return UNSAT;
  else if (res is integral) return SAT;
  else {
    Select a variable v that is assigned a non-integral value r;
    if (Branch-and-Bound(S ∪ (v ≤ ⌊r⌋))==SAT) return SAT;
    else if (Branch-and-Bound(S ∪ (v ≥ ⌈r⌉))==SAT) return SAT;
```

Branch and bound algorithm

Input: An integer linear system S
Output: SAT if S is satisfiable, UNSAT otherwise

```
procedure Branch-and-Bound(S) {
  res = LP(relaxed(S));          /*LP could be Fourier-Motzkin*/
  if (res==UNSAT) return UNSAT;
  else if (res is integral) return SAT;
  else {
    Select a variable v that is assigned a non-integral value r;
    if (Branch-and-Bound(S ∪ (v ≤ ⌊r⌋))==SAT) return SAT;
    else if (Branch-and-Bound(S ∪ (v ≥ ⌈r⌉))==SAT) return SAT;
    else return UNSAT;
  }
}
```

Example

- Let $x_1, ..., x_4$ be the variables of S. Assume that LP returns the solution $(1, 0.7, 2.5, 3)$ (line 2);

Example

- Let $x_1, ..., x_4$ be the variables of S. Assume that LP returns the solution $(1, 0.7, 2.5, 3)$ (line 2);
- Branch-and-Bound algorithm, chooses, in line 7 between x_2 and x_3; assume we choose x_2;

Example

- Let $x_1, ..., x_4$ be the variables of S. Assume that LP returns the solution $(1, 0.7, 2.5, 3)$ (line 2);
- Branch-and-Bound algorithm, chooses, in line 7 between x_2 and x_3; assume we choose x_2;
- In line 8, S (the linear system solved at the current recursion level) is then augmented with the constraint $x_2 \leq 0$ and sent for solving at a deeper recursion level. If no solution is found in this branch, S is augmented instead with $x_2 \geq 1$ and, once again, is sent to a deeper recursion level. If both these calls do not return integral values, UNSAT is returned. If one call returns SAT, then the same is performed for x_3.

Branch and bound

- The algorithm is incomplete.
- Example: $1 \leq 3x - 3y \leq 2$ has unbounded real solutions but no integer solutions $\rightarrow$ the algorithm loops forever.
- The algorithm can be made complete for formulae with the small-model property: if there is a solution, then there is also a solution within a (computable) finite bound.
- The algorithm can be extended to mixed integer linear programming, where some of the variables are integer-valued while the others are real-valued.

Branch and bound

- Branch: Split the search space
- Bound: Exclude unsatisfiable sub-spaces
- We have seen: Depth-first search
- Also possible: Breadth-first search

Branch and bound

Optimizations:

- Constraints can be removed: $x_1 + x_2 \leq 2$, $x_1 \leq 1$, $x_2 \leq 1$.

Branch and bound

Optimizations:

- Constraints can be removed: $x_1 + x_2 \leq 2$, $x_1 \leq 1$, $x_2 \leq 1$.
  First constraint is redundant.

Branch and bound

Optimizations:

- Constraints can be removed: $x_1 + x_2 \leq 2$, $x_1 \leq 1$, $x_2 \leq 1$.
  First constraint is redundant.
- Bounds can be tightened: $2x_1 + x_2 \leq 2$, $x_2 \geq 4$, $x_1 \leq 3$

Branch and bound

Optimizations:

- Constraints can be removed: $x_1 + x_2 \leq 2$, $x_1 \leq 1$, $x_2 \leq 1$.
  First constraint is redundant.
- Bounds can be tightened: $2x_1 + x_2 \leq 2$, $x_2 \geq 4$, $x_1 \leq 3$
  From the first two constraints we get $x_1 \leq -1$

Branch and bound

Optimizations:

- Constraints can be removed: $x_1 + x_2 \leq 2$, $x_1 \leq 1$, $x_2 \leq 1$.
  First constraint is redundant.
- Bounds can be tightened: $2x_1 + x_2 \leq 2$, $x_2 \geq 4$, $x_1 \leq 3$
  From the first two constraints we get $x_1 \leq -1$
- Assume a constraint $\sum_i a_i x_i \leq b$ with $l_i \leq x_i \leq u_i$.
  If $a_k > 0$, we have $x_k \leq (b - \sum_{i \neq k} a_i l_i)/a_k$.
  If $a_k < 0$, we have $x_k \geq (b - \sum_{i \neq k} a_i u_i)/a_k$.