Combinatorial Decision Making and Optimization (Module 2)

Last update: 18 May 2024

Academic Year 2023 – 2024

Alma Mater Studiorum · University of Bologna

Contents

1 Satisfiability modulo theory

Satisfiability modulo theory (SMT) Satisfiability of a formula with respect to some background formal theory/theories.

Satisfiability modulo theory (SMT)

SMT extends SAT and exploits domain-specific reasoning (possibly with infinite domains).

1.1 First-order logic for SMT

1.1.1 Syntax

Remark. Only quantifier-free formulas (q.f.f.) are considered in SMT.

Functions The set of all the functions is denoted as $\Sigma^F = \bigcup_{k \geq 0} \Sigma_k^F$ where Σ_k^F denotes the set of k-ary functions.

Functions

Constants Σ_0^F

Predicates The set of all the predicates is denoted as $\Sigma^P = \bigcup_{k \geq 0} \Sigma_k^P$ where Σ_k^P denotes the set of k-ary predicates.

Predicates

Propositional symbols Σ_0^P

Signature The set of the non-logical symbols of FOL is denoted as:

Signature

$$\Sigma = \Sigma^F \cup \Sigma^P$$

Terms The set of terms over Σ is denoted as $\mathbb{T}^\Sigma$:

Terms

$$\begin{aligned}
\mathbb{T}^\Sigma = &\Sigma_0^F \cup \\
&\{f(t_1, \dots, t_k) \mid f \in \Sigma_k^F \land t_1, \dots, t_k \in \mathbb{T}^\Sigma\} \cup \\
&\{\texttt{ite}(\varphi, t_1, t_2) \mid \varphi \in \mathbb{F}^\Sigma \land t_1, t_2 \in \mathbb{T}^\Sigma\}
\end{aligned}$$

Remark. `ite` is an auxiliary function to capture the if-then-else construct.

Formulas The set of formulas over Σ is denoted as $\mathbb{F}^\Sigma$:

Formulas

$$\begin{aligned}
\mathbb{F}^\Sigma = &\{\bot, \top\} \cup \Sigma_0^P \cup \\
&\{t_1 = t_2 \mid t_1, t_2 \in \mathbb{T}^\Sigma\} \cup \\
&\{p(t_1, \dots, t_k) \mid p \in \Sigma_k^P \land t_1, \dots, t_k \in \mathbb{T}^\Sigma\} \cup \\
&\{\neg\varphi \mid \varphi \in \mathbb{F}^\Sigma\} \cup \\
&\{(\varphi_1 \Rightarrow \varphi_2), (\varphi_1 \iff \varphi_2), (\varphi_1 \land \varphi_2), (\varphi_1 \lor \varphi_2) \mid \varphi_1, \varphi_2 \in \mathbb{F}^\Sigma\}
\end{aligned}$$

1.1.2 Semantics

Σ-model Pair $\mathcal{M} = \langle M, (\cdot)^{\mathcal{M}} \rangle$ defined on a given signature Σ where: Σ-model

- M is the universe of $\mathcal{M}$.
- $(\cdot)^{\mathcal{M}}$ is a mapping such that:
  - $\forall f \in \Sigma_k^F : f^{\mathcal{M}} \in \{\varphi \mid \varphi : M^k \to M\}$.
  - $\forall p \in \Sigma_k^P : p^{\mathcal{M}} \in \{\varphi \mid \varphi : M^k \to \{\texttt{true}, \texttt{false}\}\}$.

Interpretation Extension of the mapping function $(\cdot)^{\mathcal{M}}$ to terms and formulas: Interpretation

- $\top^{\mathcal{M}} = \texttt{true}$ and $\bot^{\mathcal{M}} = \texttt{false}$.
- $(f(t_1, \dots, t_k))^{\mathcal{M}} = f^{\mathcal{M}}(t_1^{\mathcal{M}}, \dots, t_k^{\mathcal{M}})$ and $(p(t_1, \dots, t_k))^{\mathcal{M}} = p^{\mathcal{M}}(t_1^{\mathcal{M}}, \dots, t_k^{\mathcal{M}})$.
- $\texttt{ite}(\varphi, t_1, t_2)^{\mathcal{M}} = \begin{cases} t_1^{\mathcal{M}} & \text{if } \varphi^{\mathcal{M}} = \texttt{true} \\ t_2^{\mathcal{M}} & \text{if } \varphi^{\mathcal{M}} = \texttt{false} \end{cases}$.

1.1.3 Σ-theory

Satisfiability A model $\mathcal{M}$ satisfies a formula $\varphi \in \mathbb{F}^{\Sigma}$ if $\varphi^{\mathcal{M}} = \texttt{true}$. Satisfiability

Σ-theory Possibly infinite set $\mathcal{T}$ of Σ-models. Σ-theory

$\mathcal{T}$-satisfiability A formula $\varphi \in \mathbb{F}^{\Sigma}$ is $\mathcal{T}$-satisfiable if there exists a model $\mathcal{M} \in \mathcal{T}$ that satisfies it. $\mathcal{T}$-satisfiability

$\mathcal{T}$-consistency A set of formulas $\{\varphi_1, \dots, \varphi_k\} \subseteq \mathbb{F}^{\Sigma}$ is $\mathcal{T}$-consistent iff $\varphi_1 \wedge \dots \wedge \varphi_k$ is $\mathcal{T}$-satisfiable. $\mathcal{T}$-consistency

$\mathcal{T}$-entailment A set of formulas $\Gamma \subseteq \mathbb{F}^{\Sigma}$ $\mathcal{T}$-entails a formula $\varphi \in \mathbb{F}^{\Sigma}$ ($\Gamma \models_{\mathcal{T}} \varphi$) iff in every model $\mathcal{M} \in \mathcal{T}$ that satisfies Γ, φ is also satisfied. $\mathcal{T}$-entailment

> **Remark.** Γ is $\mathcal{T}$-consistent iff $\Gamma \not\models_{\mathcal{T}} \bot$.

$\mathcal{T}$-validity A formula $\varphi \in \mathbb{F}^{\Sigma}$ is $\mathcal{T}$-valid iff $\varnothing \models_{\mathcal{T}} \varphi$. $\mathcal{T}$-validity

> **Remark.** φ is $\mathcal{T}$-consistent iff $\neg\varphi$ is not $\mathcal{T}$-valid.

Theory lemma $\mathcal{T}$-valid clause $c = l_1 \vee \dots \vee l_k$. Theory lemma

Σ-expansion Given a Σ-model $\mathcal{M} = \langle M, (\cdot)^{\mathcal{M}} \rangle$ and $\Sigma' \supseteq \Sigma$, an expansion $\mathcal{M}' = \langle M', (\cdot)^{\mathcal{M}'} \rangle$ over Σ' is any Σ'-model such that: Σ-expansion

- $M' = M$.
- $\forall s \in \Sigma : s^{\mathcal{M}'} = s^{\mathcal{M}}$

> **Remark.** Given a Σ-theory $\mathcal{T}$, we implicitly consider it to be the theory $\mathcal{T}'$ defined as:
> $$\mathcal{T}' = \{\mathcal{M}' \mid \mathcal{M}' \text{ is an expansion of a } \Sigma\text{-model } \mathcal{M} \text{ in } \mathcal{T}\}$$

Ground $\mathcal{T}$-satisfiability Given a Σ-theory $\mathcal{T}$, determine if a ground formula is $\mathcal{T}$-satisfiable over a Σ-expansion $\mathcal{T}'$. Ground $\mathcal{T}$-satisfiability

Axiomatically defined theory Given a minimal set of formulas (axioms) $\Lambda \subseteq \mathbb{F}^{\Sigma}$, its corresponding theory is the set of all the models that respect Λ. Axiomatically defined theory

Example. Let Σ be defined as:

$$\Sigma_0^F = \{a, b, c, d\} \qquad \Sigma_1^F = \{f, g\} \qquad \Sigma_2^P = \{p\}$$

A Σ-model $\mathcal{M} = \langle [0, 2\pi[, (\cdot)^\mathcal{M} \rangle$ can be defined as follows:

$$\begin{gathered} a^\mathcal{M} = 0 \qquad b^\mathcal{M} = \frac{\pi}{2} \qquad c^\mathcal{M} = \pi \qquad d^\mathcal{M} = \frac{3\pi}{2} \\ f^\mathcal{M} = \sin \qquad g^\mathcal{M} = \cos \qquad p^\mathcal{M}(x, y) \iff x > y \end{gathered}$$

To determine if $p(g(x), f(d))$ is $\mathcal{M}$-satisfiable, we have to expand $\mathcal{M}$ as there are free variables (x). Let $\Sigma' = \Sigma \cup \{x\}$. The expansion $\mathcal{M}'$ such that $x^{\mathcal{M}'} = \frac{\pi}{2}$ makes the formula satisfiable.

1.1.4 Theories of interest

Equality with Uninterpreted Functions theory (EUF) Theory $\mathcal{T}_\text{EUF}$ containing all the possible Σ-models.

Equality with Uninterpreted Functions theory (EUF)

Remark. Also called empty theory as its axiom set is $\varnothing$ (i.e. allows any model).

Remark. Useful to deal with black-box functions (i.e. prove satisfiability without a specific theory).

Example. The following formula can be proved to be unsatisfiable by only using syntactic manipulations of basic FOL concepts:

$$\begin{gathered} \big(a * (f(b) + f(c)) = d\big) \land \big(b * (f(a) + f(c)) \neq d\big) \land \underline{\big(a = b\big)} \\ \big(\underline{a * (f(a) + f(c))} = d\big) \land \big(\underline{a * (f(a) + f(c))} \neq d\big) \\ \big(g(a, c) = d\big) \land \big(g(a, c) \neq d\big) \end{gathered}$$

Arithmetic theories Theories with $\Sigma = (0, 1, +, -, \leq)$.

Arithmetic theories

Presburger arithmetic Theory $\mathcal{T}_\mathbb{Z}$ that interprets Σ-symbols over integers.

- Ground $\mathcal{T}_\mathbb{Z}$-satisfiability is **NP**-complete.
- Extended with multiplication, $\mathcal{T}_\mathbb{Z}$-satisfiability becomes undecidable.

Real arithmetic Theory $\mathcal{T}_\mathbb{R}$ that interprets Σ-symbols over reals.

- Ground $\mathcal{T}_\mathbb{R}$-satisfiability is in **P**.
- Extended with multiplication, $\mathcal{T}_\mathbb{R}$-satisfiability becomes doubly-exponential.

Remark. In floating points, commutativity still holds, but associativity and distributivity are not guaranteed.

Array theory Let $\Sigma_\mathcal{A}$ be the signature containing two functions:

Array theory

$\texttt{read}(a, i)$ Reads the value of a at index i.

$\texttt{write}(a, i, v)$ Returns an array a' where the value v is at the index i of a.

The theory $\mathcal{T}_\mathcal{A}$ is the set of all models respecting the following axioms:

- $\forall a \, \forall i \, \forall v : \texttt{read}(\texttt{write}(a, i, v), i) = v$.
- $\forall a \, \forall i \, \forall j \, \forall v : (i \neq j) \Rightarrow \Big(\texttt{read}\big(\texttt{write}(a, i, v), j\big) = \texttt{read}(a, j)\Big)$.
- $\forall a \, \forall a' : \big(\forall i : \texttt{read}(a, i) = \texttt{read}(a', i)\big) \Rightarrow (a = a')$.

| **Remark.** The full $\mathcal{T}_{\mathcal{A}}$ theory is undecidable but there are decidable fragments.

Bit-vectors theory Theory $\mathcal{T}_{\mathcal{BV}}$ with vectors of bits of fixed length as constants and operations such as: Bit-vectors theory

- String-like operations (e.g. slicing, concatenation, ...).
- Logical operations (e.g. bit-wise operators).
- Arithmetic operations (e.g. $+$, $-$, ...).

String theory Theory to handle strings of unbounded length. String theory

Theory of word equations Given an alphabet $\mathcal{S}$, a word equation has form $L = R$ where L and R are concatenations of string constants over $\mathcal{S}^*$.

| **Remark.** The general theory of word equations is undecidable.

| **Remark.** The quantifier-free theory of word equations is decidable.

| **Remark.** In practice, many theories are often combined.

1.2 Encoding to SAT

1.2.1 Eager approaches

All the information on the formal theory is used from the beginning to encode an SMT formula φ into an equisatisfiable SAT formula φ' (i.e. SMT is compiled into SAT).

Equisatisfiability Given a Σ-theory $\mathcal{T}$, two formulas φ and φ' are equisatisfiable iff: Equisatisfiability

$$\varphi \text{ is } \mathcal{T}\text{-satisfiable} \iff \varphi' \text{ is } \mathcal{T}\text{-satisfiable}$$

Eager approaches have the following advantages:

- Does not require an SMT solver.
- Once encoded, whichever SAT solver can be used.

Eager approaches have the following disadvantages:

- An ad-hoc encoding is needed for all the theories.
- The resulting SAT formula might be huge.

Algorithm Given an EUF formula φ, to determine if it is $\mathcal{T}_{\text{EUF}}$-satisfiable, the following steps are taken:

1. Replace functions and predicates with constant equalities. Given the terms $f(t_1), \dots, f(t_k)$, possible approaches are:

   Ackermann approach Ackermann approach

   - Each $f(t_i)$ is encoded into a new constant A_i.
   - Add the constraints $(t_i = t_j) \Rightarrow (A_i = A_j)$ for each $i < j$.

   Bryant approach Bryant approach

   - $f(t_1)$ is encoded as A_1.

- $f(t_2)$ is encoded as $\texttt{ite}(t_2 = t_1, A_1, A_2)$.
- $f(t_3)$ is encoded as $\texttt{ite}\big(t_3 = t_1, A_1, \texttt{ite}(t_3 = t_2, A_2, A_3)\big)$.
- $f(t_i)$ is encoded as:

$$\texttt{ite}\Big(t_i = t_1, A_1, \texttt{ite}\Big(t_i = t_2, A_2, \texttt{ite}\big(\dots, \texttt{ite}(t_i = t_{i-1}, A_{i-1}, A_i)\big)\Big)\Big)$$

2. Remove equalities to reduce φ into propositional logic. Possible encodings are:

Small-domain encoding If φ has n distinct variables $\{c_1, \dots, c_n\}$, a possible model $\mathcal{M} = \langle M, (\cdot)^{\mathcal{M}} \rangle$ that satisfies it must have $|M| \leq n$.

Therefore, each $c_i^{\mathcal{M}}$ can be associated to a value in $\{1, \dots, n\}$. In SAT, this mapping from $c_i^{\mathcal{M}}$ to $\{1, \dots, n\}$ can be encoded using $O(\log n)$ bits. Finally, an equality $c_i = c_j$ (or $c_i \neq c_j$) can be encoded by adding bitwise constraints.

Direct encoding Encode each equality $a = b$ with a propositional symbol $P_{a,b}$ and add transitivity constraints of form $(P_{a,b} \wedge P_{b,c}) \Rightarrow P_{a,c}$.

1.2.2 Lazy approaches

Integrate SAT solvers with theory-specific decision procedures.
These approaches are more flexible and modular and avoid an explosion of SAT clauses.
On the other hand, the search becomes SAT-driven and not theory-driven.

Remark. Most SMT solvers follow a lazy approach.

Algorithm Let $\mathcal{T}$ be a theory. Given a conjunction of $\mathcal{T}$-literals $\varphi = \varphi_1 \wedge \dots \wedge \varphi_n$, to determine its $\mathcal{T}$-satisfiability, a generic lazy solver does the following:

1. Each SMT literal φ_i is encoded (abstracted) into a SAT literal l_i to form the abstraction $\Phi = \{l_1, \dots, l_n\}$ of φ.
2. The $\mathcal{T}$-solver sends Φ to the SAT-solver.
   - If the SAT-solver determines that Φ is unsatisfiable, then φ is $\mathcal{T}$-unsatisfiable.
   - Otherwise, the SAT-solver returns a model $\mathcal{M} = \{a_1, \dots, a_n\}$ (an assignment of the literals, possibly partial).
3. The $\mathcal{T}$-solver determines if $\mathcal{M}$ is $\mathcal{T}$-consistent.
   - If it is, then φ is $\mathcal{T}$-satisfiable.
   - Otherwise, update $\Phi = \Phi \cup \neg\mathcal{M}$ and go to Point 2.

Example. Consider the EUF formula φ:

$$\big(g(a) = c\big) \wedge \big((f(g(a)) \neq f(c)) \vee (g(a) = d)\big) \wedge \big(c \neq d\big)$$

- φ abstracted into SAT is:

$$\underbrace{\big(g(a) = c\big)}_{l_1} \wedge \big(\neg \underbrace{\big(f(g(a)) = f(c)\big)}_{l_2} \vee \underbrace{\big(g(a) = d\big)}_{l_3}\big) \wedge \neg \underbrace{\big(c = d\big)}_{l_4}$$

$$l_1 \wedge (\neg l_2 \vee l_3) \wedge \neg l_4$$

  Therefore, $\Phi = \{l_1, (\neg l_2 \vee l_3), \neg l_4\}$

- The $\mathcal{T}$-solver sends Φ to the SAT-solver. Let's say that it return $\mathcal{M} = \{l_1, \neg l_2, \neg l_4\}$.

- The $\mathcal{T}$-solver checks if $\mathcal{M}$ is consistent. Let's say it is not. Let $\Phi' = \Phi \cup \neg\mathcal{M} = \{l_1, (\neg l_2 \vee l_3), \neg l_4, (\neg l_1 \vee l_2 \vee l_4)\}$.
- The $\mathcal{T}$-solver sends Φ' to the SAT-solver. Let's say that it return $\mathcal{M}' = \{l_1, l_2, l_3, \neg l_4\}$.
- The $\mathcal{T}$-solver checks if $\mathcal{M}'$ is consistent. Let's say it is not. Let $\Phi'' = \Phi' \cup \neg\mathcal{M}' = \{l_1, (\neg l_2 \vee l_3), \neg l_4, (\neg l_1 \vee l_2 \vee l_4), (\neg l_1 \vee \neg l_2 \vee \neg l_3 \vee l_4)\}$.
- The $\mathcal{T}$-solver sends Φ'' to the SAT-solver and it detects the unsatisfiability. Therefore, φ is $\mathcal{T}$-unsatisfiable.

Optimizations

- Check $\mathcal{T}$-consistency on partial assignments.
- Given a $\mathcal{T}$-inconsistent assignment μ, find a smaller $\mathcal{T}$-inconsistent assignment $\eta \subseteq \mu$ and add $\neg\eta$ to Φ instead of $\neg\mu$.
- When reaching $\mathcal{T}$-inconsistency, backjump to a $\mathcal{T}$-consistent point in the computation.

1.3 CDCL($\mathcal{T}$)

Lazy solver based on CDCL for SAT extended with a $\mathcal{T}$-solver. The $\mathcal{T}$-solver does the following: CDCL($\mathcal{T}$)

- Checks the $\mathcal{T}$-consistency of a conjunction of literals.
- Performs deduction of unassigned literals.
- Explains $\mathcal{T}$-inconsistent assigments.
- Allows to backtrack.

State transition Transition system to describe the reasoning of SAT or SMT solvers. A transition has form: State transition

$$(\mu \| \varphi) \to (\mu' \| \varphi')$$

where:

- φ and φ' are $\mathcal{T}$-formulas.
- μ and μ' are (partial) boolean assignments to atoms of φ and φ', respectively.
- $(\mu \| \varphi)$ and $(\mu' \| \varphi')$ are states.

Transition rule Determine the possible transitions.

Derivation Sequence of transitions.

Initial state $(\varnothing \| \varphi)$.

$\mathcal{T}$-consistency Given a $\mathcal{T}$-formula φ and a full assignment μ of φ, φ is $\mathcal{T}$-consistent ($\mu \models_{\mathcal{T}} \varphi$) if there is a derivation from $(\varnothing \| \varphi)$ to $(\mu \| \varphi)$.

$\mathcal{T}$-propagation Deduce the assignment of an unassigned literal l using some knowledge of the theory. $\mathcal{T}$-propagation

$\mathcal{T}$-consequence If: $\mathcal{T}$-consequence

- $\mu \models_{\mathcal{T}} l$,
- l or $\neg l$ occur in φ,
- l and $\neg l$ do not occur in μ,

then:

$$(\mu \| \varphi) \rightarrow (\mu \cup \{l\} \| \varphi)$$

Example. Given the formula φ:

$$\big(g(a) = c\big) \wedge \Big(\big(f(g(a)) \neq f(c)\big) \vee \big(g(a) = d\big)\Big) \wedge \big(c \neq d\big)$$

A possible derivation for some theory $\mathcal{T}$ (i.e. $\mathcal{T}$-propagation are decided arbitrarily) is:

1. $\varnothing \| \varphi$ (initial state).
2. $\varnothing \| \varphi \rightarrow \{l_1\} \| \varphi$ (Unit propagation).
3. $\{l_1\} \| \varphi \rightarrow \{l_1, l_2\} \| \varphi$ ($\mathcal{T}$-propagation).
4. $\{l_1, l_2\} \| \varphi \rightarrow \{l_1, l_2, l_3\} \| \varphi$ (Unit propagation).
5. $\{l_1, l_2, l_3\} \| \varphi \rightarrow \{l_1, l_2, l_3, l_4\} \| \varphi$ ($\mathcal{T}$-propagation).
6. $\{l_1, l_2, l_3, l_4\} \| \varphi \rightarrow$ `fail` (Failure).

As we are at decision level 0 (as no decision literal has been fixed), we can conclude that φ is unsatisfiable.

Remark. Unit and theory propagation are alternated (see algorithm description).

Algorithm Given a $\mathcal{T}$-formula φ and a (partial) $\mathcal{T}$-assignment μ (i.e. initial decisions), CDCL($\mathcal{T}$) does the following:

Algorithm 1 CDCL($\mathcal{T}$)

```
def cdclT(φ, μ):
    if preprocess(φ, μ) == CONFLICT: return UNSAT
    φ^p, μ^p = SMT_to_SAT(φ), SMT_to_SAT(μ)
    level = 0
    l = None

    while True:
        status = propagate(φ^p, μ^p, l)
        if status == SAT:
            return SAT_to_SMT(μ^p)
        elif status == UNSAT:
            η^p, jump_level = analyzeConflict(φ^p, μ^p)
            if jump_level < 0: return UNSAT
            backjump(jump_level, φ^p ∧ ¬η^p, μ^p)
        elif status == UNKNOWN:
            l = decideNextLiteral(φ^p, μ^p)
            level += 1
```

Where:

`preprocess` Preprocesses φ with μ through operations such as simplifications, $\mathcal{T}$-specific rewritings, ...

`SMT_TO_SAT` Provides the boolean abstraction of an SMT formula.

`SAT_TO_SMT` Reverses the boolean abstraction of an SMT formula.

`propagate` Iteratively apply:

- Unit propagation,
- $\mathcal{T}$-consistency check,
- $\mathcal{T}$-propagation.

Returns `SAT`, `UNSAT` or `UNKNOWN` (when no deductions are possible and there are still free variables).

`analyzeConflict` Performs conflict analysis:

- If the conflict is detected by SAT boolean propagation ($\mu^p \wedge \varphi^p \models_p \perp$), a boolean conflict set η^p is outputted (as in standard CDCL).
- If the conflict is detected by $\mathcal{T}$-propatation ($\mu \wedge \phi \models_{\mathcal{T}} \perp$), a theory conflict η is produced and its boolean abstraction η^p is outputted.

Moreover, the earliest decision level at which a variable of η^p is unassigned is returned.

As in standard CDCL, $\neg\eta^p$ is added to φ^p and the algorithm backjumps to a previous decision level (if possible).

`decideNextLiteral` Decides the assignment of an unassigned variable (as in standard CDCL). Theory information might be exploited.

Implication graph As in the standard CDCL algorithm, an implication graph is used to explain conflicts. Implication graph

Nodes Decisions, derived literals or conflicts.

Edges If v allows to unit/theory propagate w, then there is an edge $v \rightarrow w$.

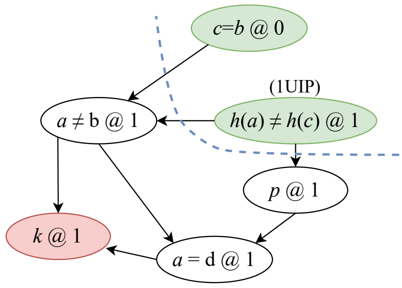
Example. Given the $\mathcal{T}$-formula φ:

$$\big(h(a) = h(c) \vee p\big) \wedge \big(a = b \vee \neg p \vee a = d\big) \wedge \big(a \neq d \vee a = b\big)$$

and an initial decision $(c = b) \in \mu$, CDCL($\mathcal{T}$) does the following:

1. As no propagation is possible, the decision $h(a) \neq h(c)$ is added to μ.
2. Unit propagate p due to the clause $\big(h(a) = h(c) \vee p\big)$ and the decision at the previous step.
3. $\mathcal{T}$-propagate $(a \neq b)$ due to the current assigments: $\{c = b, h(a) \neq h(c)\} \models_{\mathcal{T}} a \neq b$.
4. Unit propagate $(a = d)$ due to the clause $\big(a = b \vee \neg p \vee a = d\big)$ and the current knowledge base (p and $a \neq b$).
5. There is a conflict between $(a \neq d)$ and $(a = d)$.

By building the conflict graph, one can identify the 1UIP as the decision $h(a) \neq h(c)$.



A cut in front of the 1UIP that separates decision nodes and the conflict node (as in standard CDCL) is made to obtain the conflict set $\eta = \{h(a) \neq h(c), c = b\}$. $\big((h(a) = h(c)) \vee (c \neq b)\big)$ is added as a clause and the algorithm backjumps at the decision level 0.

1.4 Theory solvers

Decide satisfiability of theory-specific formulas.

1.4.1 EUF theory

Congruence closure Given a conjunction of EUF literals Φ, its satisfiability can be decided in polynomial time as follows: Congruence closure

1. Add a new variable c and replace each $p(t_1, \dots, t_k)$ with $f_p(t_1, \dots, t_k) = c$.
2. Partition input literals into the sets of equalities E and disequalities D.
3. Define E^* as the congruence closure of E. It is the smallest equivalence relation $\equiv_E$ over terms such that:
   - $(t_1 = t_2) \in E \Rightarrow (t_1 \equiv_E t_2)$.
   - For each $f(s_1, \dots, s_k)$ and $f(t_1, \dots, t_k)$ occurring in E, if $s_i \equiv_E t_i$ for each $i \in \{1, \dots, k\}$, then $f(s_1, \dots, s_k) \equiv_E f(t_1, \dots, t_k)$.
4. Φ is satisfiable iff $\forall (t_1 \neq t_2) \in D \Rightarrow (t_1 \not\equiv_E t_2)$.

Remark. In practice, congruence closure is usually implemented using a DAG to represent terms and union-find for the E^* class.

1.4.2 Arithmetic theories

Linear real arithmetic LRA theory has signature $\Sigma_{\text{LRA}} = (\mathbb{Q}, +, -, *, \leq)$ where the multiplication $*$ is only linear.

Fourier-Motzkin elimination Given an LRA formula, its satisfiability can be decided as follows: Fourier-Motzkin elimination

1. Replace:
   - $(t_1 \neq t_2)$ with $(t_1 < t_2) \vee (t_2 < t_1)$.
   - $(t_1 \leq t_2)$ with $(t_1 < t_2) \vee (t_1 = t_2)$.
2. Eliminate equalities and apply the Fourier-Motzkin elimination[1] method on all variables to determine satisfiability.

[1] https://en.wikipedia.org/wiki/Fourier%E2%80%93Motzkin_elimination

> **Remark.** Not practical on a large number of constraints. The simplex algorithm is more suited.

Linear integer arithmetic LIA theory has signature $\Sigma_{\text{LRA}} = (\mathbb{Z}, +, -, *, \leq)$ where the multiplication $*$ is only linear.

Fourier-Motzkin can be applied to check satisfiability. Simplex and branch & bound is usually better.

1.4.3 Difference logic theory

Difference logic (DL) atomic formulas have form $(x - y \leq k)$ where x, y are variables and k is a constant.

> **Remark.** Constraints with form $(x - y \bowtie k)$ where $\bowtie \in \{=, \neq, <, \geq, >\}$ can be rewritten using $\leq$.

> **Remark.** Unary constraints $x \leq k$ can be rewritten as $x - z_0 \leq k$ with the assignment $z_0 = 0$.

Theorem 1.4.1. By allowing $\neq$ and with domain in $\mathbb{Z}$, deciding satisfiability becomes NP-hard.

Proof. Any graph k-coloring instance can be poly-reduced to a difference logic formula. □

Graph consistency Given DL literals Φ, it is possible to build a weighted graph $\mathcal{G}_\Phi$ where: Graph consistency

Nodes Variables occurring in Φ.

Edges $x \xrightarrow{k} y$ for each $(x - y \leq k) \in \Phi$.

Theorem 1.4.2. Φ is inconsistent $\iff$ $\mathcal{G}_\Phi$ has a negative cycle (i.e. cycle whose cost is negative).

> **Remark.** A negative cycle acts as an inconsistency explanation (not necessarily minimal).

> **Remark.** From the consistency graph, if there is a path from x to y with cost k, then $(x - y \leq k)$ can be deduced.

1.5 Combining theories

Given $\mathcal{T}_i$-solvers for theories $\mathcal{T}_1, \dots, \mathcal{T}_n$, a general approach to combine them into a $\bigcup_i^n \mathcal{T}_i$-solver is the following:

1. Purify the formula so that each literal belongs to a single theory. New constants can be introduced.

   Interface equalities Equalities involving shared constants across solvers should be the same for all solvers.

2. Iteratively run the following:
   a) Each $\mathcal{T}_i$-solver checks the satisfiability of $\mathcal{T}_i$-formulas. If one detects unsatisfiability, the whole formula is unsatisfiable.
   b) When a $\mathcal{T}_i$-solver deduces a new literal, it sends it to the other solvers.

Example. Consider the formula:

$$\Big(f\big(f(x) - f(y)\big) = a\Big) \land \Big(f(a) = a + 2\Big) \land \Big(x = y\Big)$$

where the theories of EUF and linear arithmetic (LA) are involved.
To determine satisfiability, the following steps are taken:

1. The formula is purified to obtain the literals:

LA	**EUF**
	$f(e_1) = a$
$e_1 = e_2 - e_3$	$e_2 = f(x)$
$e_4 = 0$	$e_3 = f(y)$
$e_5 = a + 2$	$f(e_4) = e_5$
	$x = y$

   where $e_1, \dots, e_5$ are new constants.

2. Both EUF-solver and LA-solver determine `SAT`. Moreover, the EUF-solver deduces that $\{x = y, f(x) = e_2, f(y) = e_3\} \models_{EUF} (e_2 = e_3)$ and sends it to the LA-solver.

LA	**EUF**
$e_1 = e_2 - e_3$	$f(e_1) = a$
$e_4 = 0$	$e_2 = f(x)$
$e_5 = a + 2$	$e_3 = f(y)$
$\underline{e_2 = e_3}$	$f(e_4) = e_5$
	$x = y$

3. Both EUF-solver and LA-solver determine `SAT`. Moreover, the LA-solver deduces that $\{e_2 - e_3 = e_1, e_4 = 0, e_2 = e_3\} \models_{LA} (e_1 = e_4)$ and sends it to the EUF-solver.

LA	**EUF**
	$f(e_1) = a$
$e_1 = e_2 - e_3$	$e_2 = f(x)$
$e_4 = 0$	$e_3 = f(y)$
$e_5 = a + 2$	$f(e_4) = e_5$
$e_2 = e_3$	$x = y$
	$\underline{e_1 = e_4}$

$$\vdots$$

4. The EUF-solver determines `SAT` but the LA-solver determines `UNSAT`. Therefore, the formula is unsatisfiable.

1.5.1 Deterministic Nelson-Oppen

Let $\mathcal{T}_1$ be a Σ_1-theory and $\mathcal{T}_2$ be a Σ_2-theory. $(\mathcal{T}_1 \cup \mathcal{T}_2)$-satisfiability can be checked with the deterministic Nelson-Oppen if $\mathcal{T}_1$ and $\mathcal{T}_2$ are: Deterministic Nelson-Oppen

Signature-disjoint $\Sigma_1 \cap \Sigma_2 = \varnothing$.

Stably-infinite $\mathcal{T}_i$ is stably-infinite iff every $\mathcal{T}_i$-satisfiable formula φ has a corresponding $\mathcal{T}_i$-model with a universe of infinite cardinality that satisfies it.

Convex For each set of $\mathcal{T}_i$-literals S, it holds that:

$$\big(S \models_{\mathcal{T}_i} (a_1 = b_1) \vee \cdots \vee (a_n = b_n)\big) \Rightarrow \big(S \models_{\mathcal{T}_i} (a_k = b_k)\big) \text{ for some } k \in \{1, \dots, n\}$$

> **Example.** $\mathcal{T}_\mathbb{Z}$ is not convex. Consider the following formula φ:
>
> $$(1 \leq z) \wedge (z \leq 2) \wedge (u = 1) \wedge (v = 2)$$
>
> We have that:
>
> $$\varphi \models_{\mathcal{T}_\mathbb{Z}} (z = u) \vee (z = v)$$
>
> But it is not true that:
>
> $$\varphi \not\models_{\mathcal{T}_\mathbb{Z}} z = u \qquad \varphi \not\models_{\mathcal{T}_\mathbb{Z}} z = v$$

Algorithm Given a $(\mathcal{T}_1 \cup \mathcal{T}_2)$-formula S, the deterministic Nelson-Oppen algorithm works as follows:

1. Purify S into S_1 and S_2. Let E be the set of interface equalities between S_1 and S_2 (i.e. it contains all the equalities that involve shared constants).
2. If $S_1 \models_{\mathcal{T}_1} \bot$ or $S_2 \models_{\mathcal{T}_2} \bot$, then S is unsatisfiable.
3. If $S_1 \models_{\mathcal{T}_1} (x = y)$ with $(x = y) \in (E \setminus S_2)$, then $S_2 \leftarrow S_2 \cup \{x = y\}$. Go to Point 2.
4. If $S_2 \models_{\mathcal{T}_2} (x = y)$ with $(x = y) \in (E \setminus S_1)$, then $S_1 \leftarrow S_1 \cup \{x = y\}$. Go to Point 2.
5. S is satisfiable.

1.5.2 Non-deterministic Nelson-Oppen

Non-deterministic Nelson-Oppen

Extension of the deterministic Nelson-Oppen algorithm to non-convex theories.
Works by doing case splitting on pairs of shared variables and has exponential time complexity.

1.6 SMT extensions

1.6.1 Layered solvers

Layered solvers

Stratify the problem into layers of increasing complexity. The satisfiability of each layer is determined by a different solver of increasing expressivity and complexity.

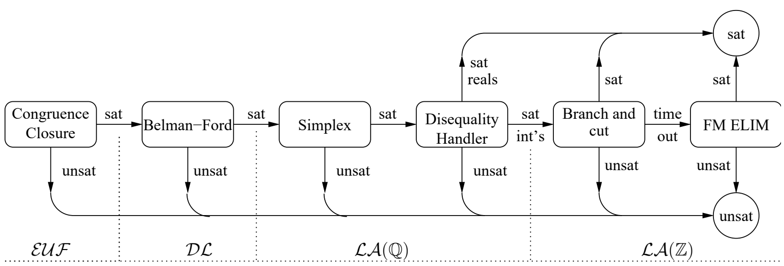


Figure 1.1: Example of layered solvers

1.6.2 Case splitting

Case reasoning on free variables. Case splitting

Example. Given the formula:

$$y = \texttt{read}\big(\texttt{write}(A, i, x), j\big)$$

A solver can explore the case when $i = j$ and $i \neq j$.

$\mathcal{T}$-solver case reasoning The $\mathcal{T}$solver internally detects inconsistencies through case reasoning.

SAT solver case reasoning The $\mathcal{T}$-solver encodes the case reasoning and sends it to the SAT solver.

Example. Given the formula:

$$y = \texttt{read}\big(\texttt{write}(A, i, x), j\big)$$

The $\mathcal{T}$-solver sends to the SAT solver the following:

$$y = \texttt{read}\big(\texttt{write}(A, i, x), j\big) \wedge (i = j) \Rightarrow y = x$$
$$y = \texttt{read}\big(\texttt{write}(A, i, x), j\big) \wedge (i \neq j) \Rightarrow y = \texttt{read}(A, j)$$

1.6.3 Optimization modulo theory

Extension of SMT so that it finds a model that simultaneously satisfies the input formula φ and optimizes an objective function f_{obj}. Optimization modulo theory

φ belongs to a theory $\mathcal{T} = \mathcal{T}_{\preceq} \cup \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_n$ where $\mathcal{T}_{\preceq}$ contains a predicate $\preceq$ (e.g. $\leq$) representing a total order.

Offline OTM($\mathcal{LRA}$) Approach that does not require to modify the SMT solver.

- **Linear search** Repeatedly solve the problem and, at each iteration, add the constraint $\texttt{cost} < c_i$ where c_i is the cost found at the i-th iteration.
- **Binary search** Given the cost domain $[l_i, u_i]$, repeatedly pick a pivot $p_i \in [l_i, u_i]$ and add the constraint $\texttt{cost} < p_i$. If a model is found, recurse in the domain $[l_i, p_i]$, otherwise recurse in $[p_i, u_i]$.

Inline OTM($\mathcal{LRA}$) SMT solver that integrates an optimization procedure.

2 Linear programming

Linear programming (LP) Optimization problem defined through a system of linear constraints (equalities and inequalities) and an objective function.

Linear programming (LP)

In the Cartesian plane, equalities represent hyperplanes and inequalities are half-spaces.

> **Remark.** LP is useful for optimal allocation with limited number of resources.

Feasible solution Assignment satisfying all the constraints.

Feasible solution

In the Cartesian plane, it is represented by any point within the intersection of all half-spaces defined by the inequalities.

Feasible region Set of all feasible solutions. It is a convex polyhedron that can be empty, bounded or unbounded.

Feasible region

> **Remark.** The optimal solution is always at one of the intersection points of the constraints within the feasible region (i.e. vertexes of the polyhedron).
> The number of vertexes is finite but might grow exponentially.

Canonical form A linear programming problem is in canonical form if it is defined as:

Canonical form

$$\max \sum_{j=1}^{n} c_j x_j \text{ subject to } \sum_{j=1}^{n} a_{i,j} x_j \leq b_i \text{ for } 1 \leq i \leq m \land$$
$$x_j \geq 0 \text{ for } 1 \leq j \leq n$$

where:

- m is the number of constraints.
- n is the number of non-negative variables.
- $a_{i,j}, b_i, c_j \in \mathbb{R}$ are known parameters (given by the definition of the problem).
- $\sum_{j=1}^{n} c_j x_j$ is the objective function to maximize.
- $\sum_{j=1}^{n} a_{i,j} x_j \leq b_i$ are m linear inequalities.

In matrix form:

$$\max\{\mathbf{c}\mathbf{x}\} \text{ subject to } \boldsymbol{A}\mathbf{x} \leq \mathbf{b} \land \mathbf{x} \geq \bar{\mathbf{0}}$$

where:

- $\mathbf{c} = \begin{bmatrix} c_1 & \dots & c_n \end{bmatrix}$.
- $\mathbf{x} = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}^T$.
- $\mathbf{b} = \begin{bmatrix} b_1 & \dots & b_m \end{bmatrix}^T$.
- $\boldsymbol{A} = \begin{bmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \dots & a_{m,n} \end{bmatrix}$

Standard form A linear programming problem is in standard form if it only has equality constraints (excluded those on single variables):

Standard form

$$\max \sum_{j=1}^{n} c_j x_j \text{ subject to } \sum_{j=1}^{n} a_{i,j} x_j = b_i \text{ for } 1 \leq i \leq m \land$$
$$x_j \geq 0 \text{ for } 1 \leq j \leq n$$

In matrix form: $\max\{\mathbf{cx}\}$ subject to $\boldsymbol{A}\mathbf{x} = \mathbf{b} \land \mathbf{x} \geq \bar{\mathbf{0}}$.

Remark (Canonical to standard form)**.** Any LP problem with m constraints in canonical form has an equivalent standard form with m slack variables $y_1, \dots, y_m \geq 0$ such that:

$$\forall i \in \{1, \dots, m\} : \left(\sum_{j=1}^{n} a_{i,j} x_j \leq b_i \right) \Rightarrow \left(\sum_{j=1}^{n} a_{i,j} x_j + y_i = b_i \land y_i \geq 0 \right)$$

Example (Brewery problem)**.** The definition of the problem in canonical form is the following:

$$\begin{array}{lccccc}
\max & 13A & + & 23B & & \\
\text{subj. to} & 5A & + & 15B & \leq & 480 \\
 & 4A & + & 4B & \leq & 160 \\
 & 35A & + & 20B & \leq & 1190 \\
 & A & , & B & \geq & 0
\end{array}$$

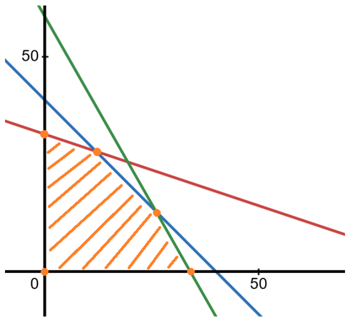


Figure 2.1: Feasible regions and vertexes of the polyhedron

In standard form, it becomes:

$$\begin{array}{lccccccccccc}
\max & 13A & + & 23B & & & & & & & & \\
\text{subj. to} & 5A & + & 15B & + & S_C & & & & & = & 480 \\
 & 4A & + & 4B & & & + & S_H & & & = & 160 \\
 & 35A & + & 20B & & & & & + & S_M & = & 1190 \\
 & A & , & B & , & S_C & , & S_H & , & S_M & \geq & 0
\end{array}$$

2.1 Simplex algorithm

Algorithm that starts from an extreme point of the polyhedron and iteratively moves to a neighboring vertex as long as the objective function does not decrease.

2.1.1 Basis

Basis Given an LP problem $\mathcal{P}$ in standard form with m constraints and n variables (note: in standard form, it holds that $m \leq n$) and its constraint matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, a (ordered) basis $\mathcal{B} = \{x_{i_1}, \dots, x_{i_m}\}$ is a subset of m of the n variables such that the columns $\boldsymbol{A}_{i_1}, \dots, \boldsymbol{A}_{i_m}$ of $\boldsymbol{A}$ form a $m \times m$ invertible matrix $\boldsymbol{A}_\mathcal{B}$.

Basis

Variables in $\mathcal{B}$ are basic variables while $\mathcal{N} = \{x_1, \dots, x_n\} \smallsetminus \mathcal{B}$ are non-basic variables.

$\mathcal{P}$ can be rewritten by separating basic and non-basic variables:

$$\max\{\mathbf{c}_\mathcal{B}\mathbf{x}_\mathcal{B} + \mathbf{c}_\mathcal{N}\mathbf{x}_\mathcal{N}\} \text{ subject to } \boldsymbol{A}_\mathcal{B}\mathbf{x}_\mathcal{B} + \boldsymbol{A}_\mathcal{N}\mathbf{x}_\mathcal{N} = \mathbf{b} \wedge \mathbf{x}_\mathcal{B}, \mathbf{x}_\mathcal{N} \geq \bar{\mathbf{0}}$$

Basic solution By constraining $\mathbf{x}_\mathcal{N} = \bar{\mathbf{0}}$, an LP problem becomes:

Basic solution

$$\max\{\mathbf{c}_\mathcal{B}\mathbf{x}_\mathcal{B}\} \text{ subject to } \boldsymbol{A}_\mathcal{B}\mathbf{x}_\mathcal{B} = \mathbf{b} \wedge \mathbf{x}_\mathcal{B} \geq \bar{\mathbf{0}}$$

As $\boldsymbol{A}_\mathcal{B}$ is invertible by definition, it holds that:

$$\mathbf{x}_\mathcal{B} = \boldsymbol{A}_\mathcal{B}^{-1}\mathbf{b} \qquad \text{and} \qquad \max\{\mathbf{c}_\mathcal{B}\mathbf{x}_\mathcal{B}\} = \max\{\mathbf{c}_\mathcal{B}\boldsymbol{A}_\mathcal{B}^{-1}\mathbf{b}\}$$

$\mathbf{x}_\mathcal{B}$ is a basic solution for $\mathcal{B}$.

Basic feasible solution (BFS) Given a basic solution $\mathbf{x}_\mathcal{B}$ for $\mathcal{B}$, it is feasible iff:

Basic feasible solution (BFS)

$$\forall_{i=1}^m \mathbf{x}_{\mathcal{B}_i} \geq 0$$

Non-degenerate BFS A basic feasible solution is non-degenerate iff $\forall_{i=1}^m \mathbf{x}_{\mathcal{B}_i} > 0$.

Non-degenerate BFS

> **Remark.** A non-degenerate BFS is represented by a unique basis.

> **Remark.** The simplex algorithm iteratively moves through basic feasible solutions $\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \dots$ such that $\mathbf{c}\tilde{\mathbf{x}}_k \geq \mathbf{c}\tilde{\mathbf{x}}_{k-1}$.

2.1.2 Tableau

Tabular representation to describe the steps of the simplex algorithm. A variable Z is introduced to represent the value of the objective function (which can be seen as a conversion of the objective function into a constraint).
The tableau of an LP problem in standard form is divided into three sections:

Tableau

1. Objective function, where the coefficients of the variables are called reduced costs.
2. Equality constraints.
3. Variable constraints.

Reduced costs

> **Example** (Brewery problem)**.** In standard form, the brewery problem is defined as:
>
> $$\begin{array}{lccccccccccc} \max & 13A & + & 23B & & & & & & & & \\ \text{subj. to} & 5A & + & 15B & + & S_C & & & & & = & 480 \\ & 4A & + & 4B & & & + & S_H & & & = & 160 \\ & 35A & + & 20B & & & & & + & S_M & = & 1190 \\ & A & , & B & , & S_C & , & S_H & , & S_M & \geq & 0 \end{array}$$
>
> As a tableau, assuming an initial basis $\mathcal{B} = \{S_C, S_H, S_M\}$, the problem is represented as:

$$
\begin{array}{rcrcrcrcrcrcr}
\hline
13A & + & 23B & & & & & & & - & Z & = & 0 \\
\hline
5A & + & 15B & + & S_C & & & & & & & = & 480 \\
4A & + & 4B & & & + & S_H & & & & & = & 160 \\
35A & + & 20B & & & & & + & S_M & & & = & 1190 \\
\hline
A & , & B & , & S_C & , & S_H & , & S_M & & & \geq & 0 \\
\hline
\end{array}
$$

The reduced costs are $\{13, 23, 0, 0, 0\}$.

2.1.3 Pivoting

Given a basis $\mathcal{B}$, it is possible to insert a new variable $x^{\text{in}} \notin \mathcal{B}$ into it and remove an old one $x^{\text{out}} \in \mathcal{B}$ to increase (or leave unchanged) the objective function:

Entering variable

Entering variable x^{in} should be the variable in $\mathcal{N}$ with the highest improvement on the objective function.

Leaving variable

Leaving variable x^{out} should be chosen to ensure that the new basis $\mathcal{B}' = \mathcal{B} \cup \{x^{\text{in}}\} \setminus \{x^{\text{out}}\}$ is still a feasible basis.

Minimum ratio rule

Minimum ratio rule For each constraint i (i.e. i-th row of the system $\boldsymbol{A}\mathbf{x} = \mathbf{b}$), it is possible to compute the ratio:

$$\frac{\mathbf{b}_i}{\boldsymbol{\alpha}_i^{\text{in}}}$$

where:

- $\boldsymbol{\alpha}_i^{\text{in}}$ is the coefficient associated to the entering variable x^{in} in the i-th constraint.
- $\mathbf{b}_i$ is the known term of the i-th constraint.

The index i of the leaving variable $x^{\text{out}} = \mathcal{B}_i$ is determined as:

$$\arg\min_i \frac{\mathbf{b}_i}{\boldsymbol{\alpha}_i^{\text{in}}}$$

Once x^{in} and $x^{\text{out}} = \mathcal{B}_i$ has been determined, x^{in} is isolated in the equation of the i-th constraint and it is substituted in all the others.

Example (Brewery problem)**.** The initial tableau of the brewery problem with $\mathcal{B} = \{S_C, S_H, S_M\}$ and $\mathcal{N} = \{A, B\}$ is:

$$
\begin{array}{rcrcrcrcrcrcr}
\hline
13A & + & 23B & & & & & & & - & Z & = & 0 \\
\hline
5A & + & 15B & + & S_C & & & & & & & = & 480 \\
4A & + & 4B & & & + & S_H & & & & & = & 160 \\
35A & + & 20B & & & & & + & S_M & & & = & 1190 \\
\hline
A & , & B & , & S_C & , & S_H & , & S_M & & & \geq & 0 \\
\hline
\end{array}
$$

1. It can be easily seen that $x^{\text{in}} = B$ should be the entering variable.

   For the leaving variable, the ratios are:

$$\arg\min\left\{\frac{480}{15}, \frac{160}{4}, \frac{1190}{20}\right\} = \arg\min\{32, 40, 59.5\} = 1$$

Therefore, the leaving variable is $x^{\text{out}} = \mathcal{B}_1 = S_C$.

We now isolate B from the first constraint:

$$\begin{aligned} 5A + 15B + S_C = 480 &\iff \frac{1}{3}A + B + \frac{1}{15}S_C = 32 \\ &\iff B = 32 - \frac{1}{3}A - \frac{1}{15}S_C \end{aligned}$$

The tableau with $\mathcal{B}' = \{B, S_H, S_M\}$ and $\mathcal{N}' = \{A, S_C\}$ is updated as:

$$\begin{array}{cccccccccccc} \hline \frac{16}{3}A & & & - & \frac{23}{15}S_C & & & & & - & Z & = & -736 \\ \hline \frac{1}{3}A & + & B & + & \frac{1}{15}S_C & & & & & & & = & 32 \\ \frac{8}{3}A & & & - & \frac{4}{15}S_C & + & S_H & & & & & = & 32 \\ \frac{85}{3}A & & & - & \frac{4}{3}S_C & & & + & S_M & & & = & 550 \\ \hline A & , & B & , & S_C & , & S_H & , & S_M & & & \geq & 0 \\ \hline \end{array}$$

2. Now, $x^{\text{in}} = A$ is the variable that increases the objective function the most.

   For the leaving variable, the ratios are:

$$\arg\min \left\{ \frac{32}{1/3}, \frac{32}{8/3}, \frac{550}{85/3} \right\} = 2$$

   Therefore, the leaving variable is $x^{\text{out}} = \mathcal{B}'_2 = S_H$.

   A isolated in the second constraint brings to:

$$A = \frac{3}{8}(32 + \frac{4}{15}S_C - S_H) \iff A = 12 + \frac{1}{10}S_C - \frac{3}{8}S_H$$

   The tableau with $\mathcal{B}'' = \{A, B, S_M\}$ and $\mathcal{N}'' = \{S_C, S_H\}$ is updated as:

$$\begin{array}{cccccccccccc} \hline & & & - & S_C & - & 2S_H & & & - & Z & = & -800 \\ \hline & & B & + & \frac{1}{10}S_C & + & \frac{1}{8}S_H & & & & & = & 28 \\ A & & & - & \frac{1}{10}S_C & + & \frac{3}{8}S_H & & & & & = & 12 \\ & & & - & \frac{25}{6}S_C & - & \frac{85}{8}S_H & + & S_M & & & = & 210 \\ \hline A & , & B & , & S_C & , & S_H & , & S_M & & & \geq & 0 \\ \hline \end{array}$$

   We cannot continue anymore as the reduced costs $\{0, 0, -1, -2, 0\}$ are ≤ 0 (i.e. cannot improve the objective function anymore).

2.1.4 Optimality

Optimality

When any substitution worsens the objective function, the current assignment is optimal. In the tableau, this happens when all the reduced costs are ≤ 0.

Remark. For any optimal solution, there is at least a basis such that the reduced costs are ≤ 0. Therefore, this is a sufficient condition.

Remark. The fact that reduced costs are ≤ 0 is not a necessary condition.

Example (Brewery problem)**.** The tableau at the last iteration, with $\mathcal{B}'' = \{A, B, S_M\}$ and $\mathcal{N}'' = \{S_C, S_H\}$, is the following:

$$
\begin{array}{ccccccccccc}
\hline
 & & - & S_C & - & 2S_H & & & - & Z & = & -800 \\
\hline
 & B & + & \frac{1}{10}S_C & + & \frac{1}{8}S_H & & & & & = & 28 \\
A & & - & \frac{1}{10}S_C & + & \frac{3}{8}S_H & & & & & = & 12 \\
 & & - & \frac{25}{6}S_C & - & \frac{85}{8}S_H & + & S_M & & & = & 210 \\
\hline
A & , B & , & S_C & , & S_H & , & S_M & & & \geq & 0 \\
\hline
\end{array}
$$

All reduced costs are ≤ 0 and an optimal solution has been reached:

- $S_C = 0$ and $S_H = 0$ (variables in $\mathcal{N}''$).
- $A = 12$, $B = 28$ and $S_M = 210$ (variables in $\mathcal{B}''$. Obtained by isolating them in the constraints).
- $-S_C - 2S_H - Z = -800 \iff Z = 800 - S_C - 2S_H \iff Z = 800$ (objective function).

Remark. The points in the feasible region of a problem $\mathcal{P}$ are:

$$\mathcal{F}_\mathcal{P} = \{\mathbf{x} \in \mathbb{R}^n \mid \boldsymbol{A}\mathbf{x} \leq \mathbf{b} \land \mathbf{x} \geq \bar{\mathbf{0}}\}$$

Optimal region For an LP problem $\mathcal{P}$, its set of solutions is defined as:

Optimal region

$$\mathcal{O}_\mathcal{P} = \{x^* \in \mathcal{F}_\mathcal{P} \mid \forall \mathbf{x} \in \mathcal{F}_\mathcal{P} : \mathbf{c}\mathbf{x}^* \geq \mathbf{c}\mathbf{x}\}$$

Trivially, it holds that $\mathcal{O}_\mathcal{P} \subseteq \mathcal{F}_\mathcal{P}$ and $\mathcal{F}_\mathcal{P} = \varnothing \Rightarrow \mathcal{O}_\mathcal{P} = \varnothing$.

Theorem 2.1.1. If $\mathcal{O}_\mathcal{P}$ is finite, then $|\mathcal{O}_\mathcal{P}| = 1$ (therefore, if $|\mathcal{O}_\mathcal{P}| > 1$, then $\mathcal{O}_\mathcal{P}$ is infinite).

Unbounded problem An LP problem $\mathcal{P}$ is unbounded if it does not have an optimal solution (i.e. $\mathcal{F}_\mathcal{P}$ is an unbounded polyhedron).

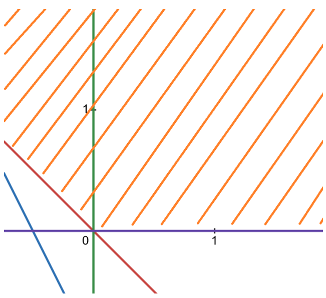
Unbounded problem

With the tableau formulation, if a column has reduced cost > 0 and all the constraint coefficients are ≤ 0, then the problem is unbounded.

Example. Given the following tableau:

$$
\begin{array}{cccccccccc}
\hline
 & x & - & y & & & & & - & Z & = & -1 \\
\hline
- & x & - & y & + & S_1 & & & & & = & 0 \\
- & 2x & - & y & & & + & S_2 & & & = & 1 \\
\hline
 & x & , & y & , & S_1 & , & S_2 & & & \geq & 0 \\
\hline
\end{array}
$$

The unboundedness of the problem can be detected from the first column.



2.1.5 Algorithm

Given an LP problem $\mathcal{P}$ in standard form, the steps of the simplex algorithm are:

1. Set $k = 0$, find a feasible basis $\mathcal{B}_k$ and reformulate $\mathcal{P}$ according to it.
2. If the basis feasible solution is optimal, return.
3. If $\mathcal{P}$ is unbounded, return.
4. Select an entering variable x^{in}.
5. Select a leaving variable x^{out}.
6. Let $\mathcal{B}_{k+1} = \mathcal{B}_k \cup \{x^{\text{in}}\} \smallsetminus \{x^{\text{out}}\}$ and reformulate $\mathcal{P}$ according to the new basis.
7. Set $k = k + 1$ and go to Point 2.

Properties

- If all basis feasible solutions are non-degenerate, the simplex algorithm always terminates (as the solution is always strictly improving).
- In the general case, the algorithm might stall by ending up in a loop.
- The worst-case time complexity is $O(2^n)$. The average case is polynomial.

Remark. If the problem has lots of vertexes, the interior point method (polynomial complexity) or approximation algorithms should be preferred.

2.1.6 Two-phase method

Method that solves an LP problem by first finding an initial feasible basis $\mathcal{B}_0$ and determining if the LP problem is unsatisfiable. Two-phase method

Given an LP problem $\mathcal{P}$ ($\max\{\mathbf{c}\mathbf{x}\}$ subject to $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ with m constraints and n variables), the two-phase method works as follows:

Phase 1 Define a new artificial problem $\mathcal{P}'$ from $\mathcal{P}$ with new variables $s_1, \dots, s_m$ as follows:

$$\max\left\{-\sum_{i=1}^{m} s_i\right\} \text{ subject to } \begin{aligned}[t] &\sum_{j=1}^{n} a_{i,j}x_j + s_i = b_i \text{ for } i \in \{k \in \{1,\dots,m\} \mid b_k \geq 0\} \,\land \\ &\sum_{j=1}^{n} a_{i,j}x_j - s_i = b_i \text{ for } i \in \{k \in \{1,\dots,m\} \mid b_k < 0\} \,\land \\ &s_i, x_j \geq 0 \end{aligned}$$

Remark. It holds that $-\sum_{i=1}^{m} s_i \leq 0$ and $\mathcal{B}' = \{s_1, \dots, s_m\}$ is always a feasible basis corresponding to the basis feasible solution $x_j = 0$, $s_i = |b_i|$

The problem $\mathcal{P}'$ with basis $\mathcal{B}'$ can be solved through the simplex method.

Theorem 2.1.2. Let $\mathcal{F}_\mathcal{P}$ be the feasible region of $\mathcal{P}$. It holds that:

$$\mathcal{F}_\mathcal{P} \neq \varnothing \iff \sum_{i=1}^{m} s_i = 0$$

In other words:

- If the optimal solution of $\mathcal{P}'$ is < 0, then $\mathcal{P}$ is unsatisfiable.
- Otherwise, the basis $\mathcal{B}_{\mathcal{P}'}$ corresponding to the optimal solution of $\mathcal{P}'$ can be used as the initial basis of $\mathcal{P}$, after removing the variables s_i.

Phase 2 Solve $\mathcal{P}$ through the simplex algorithm using as initial basis $\mathcal{B}_{\mathcal{P}'}$.

2.1.7 Duality

Dual problem Given the primal problem $\mathcal{P}$ defined as:

$$P : \max\{\mathbf{cx}\} \text{ subject to } \boldsymbol{A}\mathbf{x} = \mathbf{b} \wedge \mathbf{x} \geq \bar{\mathbf{0}}$$

with $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{x} \in \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, its dual problem $\mathcal{D}(P)$ is defined as:

$$\mathcal{D}(P) : \min\{\mathbf{by}\} \text{ subject to } \boldsymbol{A}^T\mathbf{y} \geq \mathbf{c} \wedge \mathbf{y} \geq \bar{\mathbf{0}}$$

where:

- $\mathbf{y} \in \mathbb{R}^m$ has a variable $\mathbf{y}_i$ for each constraint $\sum_{j=1}^{n} \boldsymbol{A}_{i,j}\mathbf{x}_j = \mathbf{b}_i$ of P,
- $\sum_{i=1}^{m} \boldsymbol{A}_{j,i}\mathbf{y}_i \geq \mathbf{c}_j$ is a new constraint defined for each variable $\mathbf{x}_j$ of P.

Remark. The constraint $\mathbf{y} \geq \bar{\mathbf{0}}$ comes out naturally from the conversion from primal to dual. Therefore, it is not strictly necessary to explicitly put it.

Example. Given the primal problem:

max	$1x_1$	+	$1x_2$	+	$0x_3$	+	$0x_4$	+	$0x_5$		
subj. to	$3x_1$	+	$2x_2$	+	$1x_3$	+	$0x_4$	+	$0x_5$	$=$	5
	$4x_1$	+	$5x_2$	+	$0x_3$	+	$1x_4$	+	$0x_5$	$=$	4
	$0x_1$	+	$1x_2$	+	$0x_3$	+	$0x_4$	+	$1x_5$	$=$	2
	x_1	,	x_2	,	x_3	,	x_4	,	x_5	$\geq$	0

Its dual is:

min	$5y_1$	+	$4y_2$	+	$2y_3$		
subj. to	$3y_1$	+	$4y_2$	+	$0y_3$	$\geq$	1
	$2y_1$	+	$5y_2$	+	$1y_3$	$\geq$	1
	$1y_1$	+	$0y_2$	+	$0y_3$	$\geq$	0
	$0y_1$	+	$1y_2$	+	$0y_3$	$\geq$	0
	$0y_1$	+	$0y_2$	+	$1y_3$	$\geq$	0

Theorem 2.1.3. For any primal problem $\mathcal{P}$, it holds that $\mathcal{D}(\mathcal{D}(\mathcal{P})) = \mathcal{P}$.

Theorem 2.1.4 (Weak duality)**.** The cost of any feasible solution of the primal $\mathcal{P}$ is less or equal than the cost of any solution of the dual $\mathcal{D}(\mathcal{P})$: Weak duality

$$\forall \mathbf{x} \in \mathcal{F}_{\mathcal{P}}, \forall \mathbf{y} \in \mathcal{F}_{\mathcal{D}(\mathcal{P})} : \mathbf{cx} \leq \mathbf{by}$$

In other words, **by** is an upper bound for $\mathcal{P}$ and **cx** is a lower bound for $\mathcal{D}(P)$.

Corollary 2.1.4.1. If $\mathcal{P}$ is unbounded, then $\mathcal{D}(\mathcal{P})$ is unfeasible:

$$\mathcal{F}_{\mathcal{P}} \neq \varnothing \wedge \mathcal{O}_{\mathcal{P}} = \varnothing \;\Rightarrow\; \mathcal{F}_{\mathcal{D}(\mathcal{P})} = \varnothing$$

Corollary 2.1.4.2. If $\mathcal{D}(\mathcal{P})$ is unbounded, then $\mathcal{P}$ is unfeasible:

$$\mathcal{F}_{\mathcal{D}(\mathcal{P})} \neq \varnothing \wedge \mathcal{O}_{\mathcal{D}(\mathcal{P})} = \varnothing \;\Rightarrow\; \mathcal{F}_{\mathcal{P}} = \varnothing$$

Strong duality

Theorem 2.1.5 (Strong duality)**.** If the primal and the dual are feasible, then they have the same optimal values:

$$\left(\mathcal{F}_{\mathcal{P}} \neq \varnothing \wedge \mathcal{F}_{\mathcal{D}(\mathcal{P})} \neq \varnothing\right) \Rightarrow \left(\forall \mathbf{x}^* \in \mathcal{O}_{\mathcal{P}}, \forall \mathbf{y}^* \in \mathcal{O}_{\mathcal{D}(\mathcal{P})} : \mathbf{c}\mathbf{x}^* = \mathbf{b}\mathbf{y}^*\right)$$

Dual simplex

Dual simplex Move from optimal basis (which can be unfeasible) to feasible basis, while preserving optimality.

> **Remark.** The traditional primal simplex moves from feasible to optimal basis, while preserving feasibility.

Properties

- Duality makes the time complexity of finding a feasible or optimal solution the same.
- The dual allows to prove the unfeasibility of the primal.
- Primal and dual provide a bounding of the objective function.

2.1.8 Sensitive analysis

Sensitive analysis

Study how the optimal solution of a problem $\mathcal{P}$ is affected if $\mathcal{P}$ is perturbed.
Given a problem $\mathcal{P}$ with optimal solution $\mathbf{x}^*$, a perturbed problem $\bar{\mathcal{P}}$ can be obtained by altering:

Known terms Change of form:

$$\mathbf{b} \rightsquigarrow \bar{\mathbf{b}} = \mathbf{b} + \Delta\mathbf{b}$$

This can affect the feasibility and optimality of $\mathbf{x}^*$.

> **Remark.** Changing the known terms of $\mathcal{P}$ changes the objective function of $\mathcal{D}(\mathcal{P})$.

Objective function coefficients Change of form:

$$\mathbf{c} \rightsquigarrow \bar{\mathbf{c}} = \mathbf{c} + \Delta\mathbf{c}$$

This can affect the optimality of $\mathbf{x}^*$.

Constraint coefficients Change of form:

$$\boldsymbol{A} \rightsquigarrow \bar{\boldsymbol{A}} = \boldsymbol{A} + \Delta\boldsymbol{A}$$

- If the change involves a variable $\mathbf{x}_j^* = 0$, then feasibility is not changed but optimality can be affected.
- If the change involves a variable $\mathbf{x}_j^* \neq 0$, the problem needs to be re-solved.

2.2 (Mixed) integer linear programming

Integer linear programming (ILP) Linear programming problem where variables are all integers:

$$\max \sum_{j=1}^{n} c_j x_j \text{ subject to } \sum_{j=1}^{n} a_{i,j} x_j = b_i \quad \text{for } 1 \leq i \leq m \;\land$$
$$x_j \geq 0 \land x_j \in \mathbb{Z} \quad \text{for } 1 \leq j \leq n$$

Integer linear programming (ILP)

Mixed-integer linear programming (MILP) Linear programming problem with n variables where $k < n$ variables are integers.

Mixed-integer programming (MILP)

Theorem 2.2.1. Finding a feasible solution of a mixed-integer linear programming problem is NP-complete.

Proof.

MILP in NP) A certificate contains an assignment of the variables. It is sufficient to check if all constraints are satisfied. Both steps are polynomial.

MILP is NP-hard) Any SAT problem can be poly-reduced to MILP.

□

Remark. Due to its NP-completeness, MILP problems can be solved by:

Exact algorithms Guarantee optimal solution with exponential time complexity.

Approximate algorithms Guarantee polynomial time complexity but might provide sub-optimal solutions with an approximation factor ρ.

Heuristic algorithms Empirically find a good solution in a reasonable time (no theoretical guarantees).

2.2.1 Linear relaxation

Given a MILP problem $\mathcal{P}$, its linear relaxation $\mathcal{L}(\mathcal{P})$ removes the constraints $x_j \in \mathbb{Z}$. However, solving $\mathcal{L}(\mathcal{P})$ as an LP problem and rounding the solution does not guarantee feasibility or optimality.

Linear relaxation

Theorem 2.2.2. It holds that $\mathcal{F}_{\mathcal{L}(\mathcal{P})} = \varnothing \Rightarrow \mathcal{F}_{\mathcal{P}} = \varnothing$.
Therefore, the linear relaxation of a MILP problem can be used to verify unsatisfiability.

Remark. If $\mathcal{F}_{\mathcal{L}(\mathcal{P})}$ is unbounded, then $\mathcal{P}$ can either be bounded, unbounded or unsatisfiable.

2.2.2 Branch-and-bound

Given an ILP problem $\mathcal{P}$, the branch-and-bound algorithm solves it with a divide et impera approach.
The algorithm does the following:

Branch-and-bound

1. Set the current best optimal value $z^* = -\infty$ and put $\mathcal{P}$ as the root of a search tree.

2. Solve $\mathcal{P}_0 = \mathcal{L}(\mathcal{P})$ to obtain a solution $\{x_1 = \beta_1, \dots, x_n = \beta_n\}$.

3. If each β_i is an integer, the solution is optimal and terminate.

4. Pick a variable x_k such that its assignment $\beta_k \notin \mathbb{Z}$ and branch the problem:

$$\begin{cases} \mathcal{P}_1 = \mathcal{P}_0 \cup \{x_k \leq \lfloor \beta_k \rfloor\} \\ \mathcal{P}_2 = \mathcal{P}_0 \cup \{x_k \geq \lceil \beta_k \rceil\} \end{cases}$$

5. Add $\mathcal{P}_1$ and $\mathcal{P}_2$ as children of $\mathcal{P}$ in the search tree. Solve the linear relaxations $\mathcal{L}(\mathcal{P}_1)$ and $\mathcal{L}(\mathcal{P}_2)$:
   - If $\mathcal{L}(\mathcal{P}_k)$ has an integral solution, it is optimal for the subproblem. The best optimal value z^* is updated if the current objective value z_k is higher.

     In the search tree, $\mathcal{P}_k$ becomes a leaf.
   - If $\mathcal{L}(\mathcal{P}_k)$ does not have an integral solution, continue branching as in Point 4. If this is not possible, $\mathcal{P}_k$ becomes a leaf in the search tree.

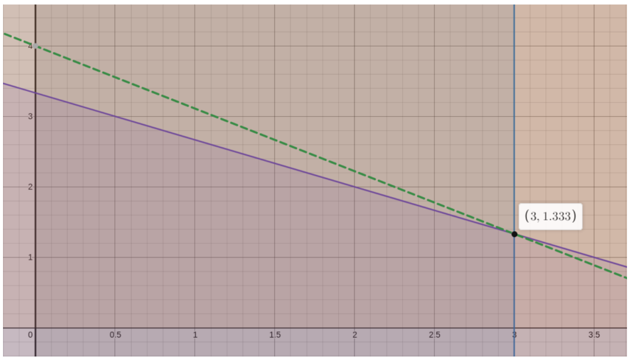
Fathomed node Leaf of the search tree.

Incumbent solution Leaf whose solution is optimal.

Example (Bakery problem)**.** Consider the problem $\mathcal{P}$:

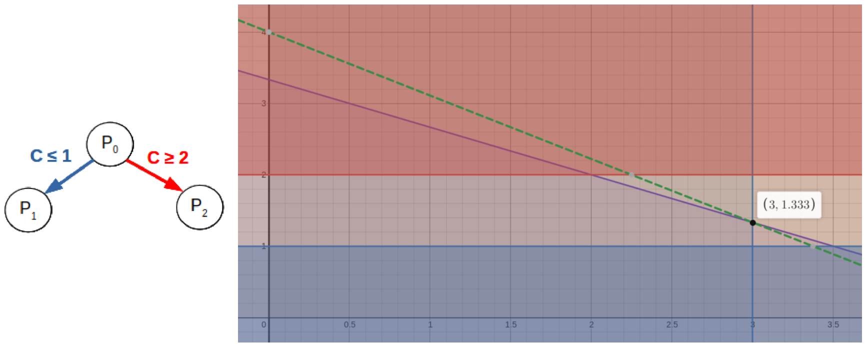
$$\begin{array}{lccccc} \max & 400B & + & 450C & & \\ \text{subj. to} & 250B & + & 200C & \leq & 4000 \\ & 2B & & & \leq & 6 \\ & 75B & + & 150C & \leq & 2000 \\ & 100B & + & 150C & \leq & 500 \\ & & & 75C & \leq & 500 \\ & B & , & C & \in & \{1, 2, \dots, 100\} \end{array}$$

1. The solution of $\mathcal{P}_0 = \mathcal{L}(\mathcal{P})$ is $\{B = 3, C = \frac{4}{3}\}$.

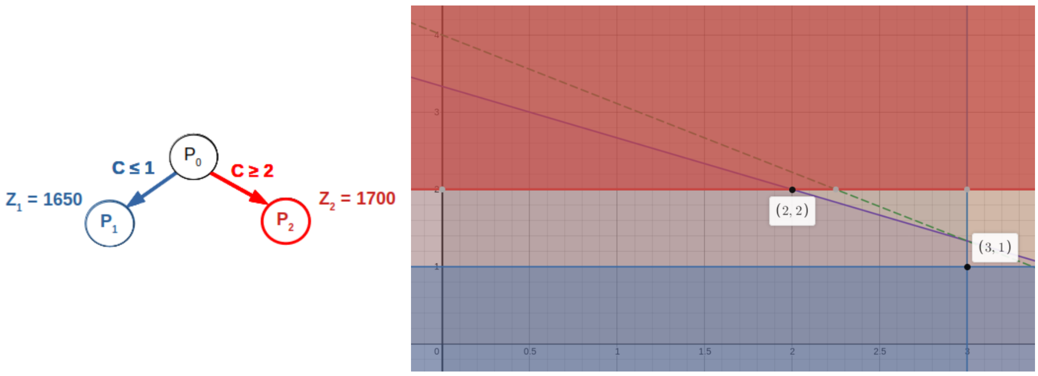


2. We have to branch on the variable C:

$$\begin{cases} \mathcal{P}_1 = \mathcal{P}_0 \cup \{C \leq \lfloor \frac{4}{3} \rfloor\} = \mathcal{P}_0 \cup \{C \leq 1\} \\ \mathcal{P}_2 = \mathcal{P}_0 \cup \{C \geq \lceil \frac{4}{3} \rceil\} = \mathcal{P}_0 \cup \{C \geq 2\} \end{cases}$$



3. The solution of $\mathcal{L}(\mathcal{P}_1)$ is $\{B = 3, C = 1\}$ and has objective value $z_1 = 1650$. As it is integral, the current best solution is updated to $z^* = 1650$ and no further branching is needed.
4. The solution of $\mathcal{L}(\mathcal{P}_2)$ is $\{B = 2, C = 2\}$ and has objective value $z_2 = 1700$. As it is integral, the current best solution is updated to $z^* = 1700$ and no further branching is needed.



5. The leaf containing $\mathcal{P}_2$ is optimal.

Possible techniques to improve branch-and-bound are:

Presolve Reformulate the problem $\mathcal{P}$ before solving it to reduce the size of $\mathcal{F}_{\mathcal{L}(\mathcal{P})}$ (without altering $\mathcal{F}_{\mathcal{P}}$). Presolve

Bounds tightening Infer stronger constraints. Bounds tightening

Example. $\{x_1 + x_2 \geq 20, x_1 \leq 10\} \models x_2 \geq 10$.

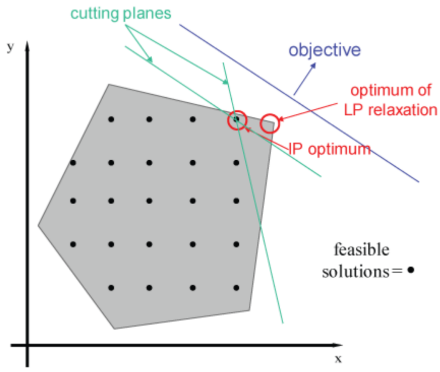
Problem reduction Infer the assignment to variables. Problem reduction

Example. $\{x_1 + x_2 \geq 0.8\} \models x_1 = x_2 = 0$.

Cutting planes Add constraints to reduce the space of non-integral solutions ($\mathcal{F}_{\mathcal{L}(\mathcal{P})} \setminus \mathcal{F}_{\mathcal{P}}$). Cutting planes

Given a MILP problem $\mathcal{P}$, a cut is an inequality:

$$\begin{aligned} \mathbf{px} \leq \mathbf{q} \text{ such that } \forall \mathbf{y} \in \mathcal{F}_{\mathcal{P}} : \mathbf{py} \leq \mathbf{p} \wedge & \quad \text{(feasible solutions inside the cut)} \\ \forall \mathbf{z} \in \mathcal{O}_{\mathcal{L}(\mathcal{P})} : \mathbf{pz} > \mathbf{q} & \quad \text{(non-integral solutions outside the cut)} \end{aligned}$$



Theorem 2.2.3. There always exists a (possibly non-unique) cut separating the optimal solution in $\mathcal{F}_{\mathcal{L}(\mathcal{P})} \smallsetminus \mathcal{F}_{\mathcal{P}}$ from $\mathcal{F}_{\mathcal{P}}$.

Remark. The cut can be done when branching or as a standalone operation (branch-and-cut).

Gomory's cut Consider the optimal solution of $\mathcal{L}(\mathcal{P})$ with basis $\mathcal{B}^* = \{x_{i_1}, \dots, x_{i_m}\}$ and non-basic variables $\mathcal{N}^* = \{x_{i_{m+1}}, \dots, x_{i_n}\}$. The cut aims to separate a non-integral vertex of the polytope and all the other feasible integer points.

Gomory's cut

If there is a k such that $x_{i_k} = \beta_k \notin \mathbb{Z}$, then $\mathbf{x}^* \in \mathcal{F}_{\mathcal{L}(\mathcal{P})} \smallsetminus \mathcal{F}_{\mathcal{P}}$ and it can separated from the optimal solution in $\mathcal{F}_{\mathcal{P}}$. x_{i_k} can be written in basic form as:

$$x_{i_k} = \beta_k + \sum_{j=1}^{n-m} \alpha_{k,j} x_{i_{m+j}}$$

The cut has form:

$$\begin{aligned}
&\sum_{j=1}^{n-m} (-\alpha_{k,j} - \lfloor -\alpha_{k,j} \rfloor) x_{i_{m+j}} \geq (\beta_k - \lfloor \beta_k \rfloor) \\
\iff \quad & -(\beta_k - \lfloor \beta_k \rfloor) + \sum_{j=1}^{n-m} (-\alpha_{k,j} - \lfloor -\alpha_{k,j} \rfloor) x_{i_{m+j}} \geq 0
\end{aligned}$$

The problem is then extended with the cut as:

$$\mathcal{L}(\mathcal{P}) \cup \left\{ y_k = -(\beta_k - \lfloor \beta_k \rfloor) + \sum_{j=1}^{n-m} (-\alpha_{k,j} - \lfloor -\alpha_{k,j} \rfloor) x_{i_{m+j}} \land y_k \geq 0 \right\}$$

where y_k is a new slack variable. The old optimal solution for the updated problem is unfeasible but the dual is feasible.

Bender's decomposition Consider a problem with m inequalities and n variables. Variables can be partitioned into $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^{n-p}$ for $p = 1, \dots, n$. The problem can be rewritten as:

Bender's decomposition

$$\begin{array}{lcccccc}
\min_{\mathbf{x},\mathbf{y}} & \mathbf{c}^T\mathbf{x} & + & \mathbf{d}^T\mathbf{y} & & \\
\text{subj. to} & \boldsymbol{A}\mathbf{x} & + & \boldsymbol{B}\mathbf{y} & \geq & \mathbf{b} \\
& \mathbf{x} \geq \bar{\mathbf{0}} & , & \mathbf{y} \in \mathcal{Y} & &
\end{array}$$

where $\mathcal{Y} \subseteq \mathbb{R}^{n-p}$ is the feasible set of $\mathbf{y}$.

For any $\bar{\mathbf{y}} \in \mathcal{Y}$, the residual problem is defined as:

$$\begin{array}{llcll} \min_{\mathbf{x}} & \mathbf{c}^T\mathbf{x} & + & \mathbf{d}^T\bar{\mathbf{y}} & \\ \text{subj. to} & \boldsymbol{A}\mathbf{x} & \geq & \mathbf{b} - \boldsymbol{B}\bar{\mathbf{y}} & , \quad \mathbf{x} \geq \bar{\mathbf{0}} \end{array}$$

The dual of the residual problem is:

$$\begin{array}{llcl} \max_{\mathbf{u}} & (\mathbf{b} - \boldsymbol{B}\bar{\mathbf{y}})^T\mathbf{u} & + & \overbrace{\mathbf{d}^T\bar{\mathbf{y}}}^{\text{Constant}} \\ \text{subj. to} & \boldsymbol{A}^T\mathbf{u} \leq \mathbf{c} & , & \mathbf{u} \geq \bar{\mathbf{0}} \end{array}$$

Therefore, the original problem (master problem) becomes a min-max problem:

$$\min_{\mathbf{y} \in \mathcal{Y}} \left[\mathbf{d}^T\mathbf{y} + \max_{\mathbf{u} \geq \bar{\mathbf{0}}} \left\{ (\mathbf{b} - \boldsymbol{B}\bar{\mathbf{y}})^T\mathbf{u} \mid \boldsymbol{A}^T\mathbf{u} \leq \mathbf{c} \right\} \right]$$

Fixed an initial $\mathbf{y}$, the method does the following:

1. Initialize an empty set of cuts $\mathcal{C}$.
2. Solve the linear relaxation of the max sub-problem (dual of residual):
   - If it is unbounded, the residual problem is unfeasible. Add a cut in $\mathcal{C}$ to exclude $\mathbf{y}$.
   - If it is unfeasible, the residual problem is unbounded or unfeasible. Terminate.
   - If $\bar{\mathbf{u}}$ is optimal, it is optimal for the residual problem too. Add the cut $\mathbf{c}^T\mathbf{x} + \cancel{\mathbf{d}^T\mathbf{y}} \geq (\mathbf{b} - \boldsymbol{B}\mathbf{y})^T\bar{\mathbf{u}} + \cancel{\mathbf{d}^T\mathbf{y}}$ (by weak duality) to $\mathcal{C}$.
3. Solve the updated master problem to get a new solution $\bar{\mathbf{y}}$.
4. If the new bounds gap is lower than a threshold, stop and solve the residual problem for $\bar{\mathbf{x}}$. Otherwise, go to Point 2 with the new $\bar{\mathbf{y}}$.

Heuristics Empirical methods to guide the search. Heuristics

- Local search
- Meta-heuristics
- MILP heuristics:
  - Rounding.
  - Diving: rounding and re-solving by fixing some variables.
  - Sub-MIPing: solving by fixing some variables.

Warm start Search from a given initial total or partial assignment of the variables. Warm start

2.3 Non-linear programming

Problem of form: Non-linear programming

$$\begin{array}{lll} \min f(\mathbf{x}) \;\; \text{subj. to} & g_i(\mathbf{x}) \leq \bar{\mathbf{0}} & \text{for } j = 1, \dots, m \\ & h_j(\mathbf{x}) = \bar{\mathbf{0}} & \text{for } j = 1, \dots, p \end{array}$$

where $\mathbf{x} \in \mathbb{R}^n$ and f, g_i, h_j are non-linear functions.

Remark. Non-linear problems are solved using optimization methods (e.g. gradient descent, Newton's method, ...).

2.4 Linearization

Methods to linearize constraints. They usually work if the domains of the variables are bounded.

Reification Linearize logical combinations of linear constraints. Integer reification

Given a constraint $C(x_1, \dots, x_k)$, a new boolean variable $b \in \{0, 1\}$ is introduced to reify it. Depending on the type of reification, b behaves as follows:

Integer full-reification $(b = 1) \iff C(x_1, \dots, x_k)$ Full-reification

Integer half-reification $(b = 1) \Rightarrow C(x_1, \dots, x_k) \wedge (b = 0) \Rightarrow \neg C(x_1, \dots, x_k)$ Half-reification

Given the reifications b_i of some constraints C_i, the logical combination is modeled by adding new constraints on b_i.

Example. $\bigvee_i C_i$ is modeled by imposing $\sum_i b_i \geq 1$.

Big-M trick Half-reification of bounded linear inequalities. Big-M trick

Given a conjunction of constraints $C_1 \vee \dots \vee C_m$, it is modeled as follows:

1. Introduce m new boolean variables $b_1, \dots, b_m$ and impose $\sum_{i=1}^{m} b_i \geq 1$.
2. For each $C_i \equiv \sum_j \alpha_{i,j} x_j \leq \beta_i$, add a new constraint:

$$\sum_j \alpha_{i,j} x_j - \beta_i \leq M_i \cdot (1 - b_i)$$

where M_i is a "big enough" constant. In this way:

- $(b_i = 0) \Rightarrow \sum_j \alpha_{i,j} x_j - \beta_i \leq M_i$ is always satisfied (as M_i is big enough for any assignment of x_j).
- $(b_i = 1) \Rightarrow \sum_j \alpha_{i,j} x_j - \beta_i \leq 0$ is the original constraint.

Big-M number Constant M for the constraints. Assuming that each variable is bounded ($x_j \in \{l_j, \dots, u_j\}$), the constant big-M for the constraint C_i can be defined as:

$$M_i = -\beta_i + \sum_j \max\{(\alpha_{i,j} l_j), (\alpha_{i,j} u_j)\}$$

Example. Given the variables $x \in \{0, \dots, 30\}$, $y \in \{-5, \dots, -2\}$, $z \in \{-6, \dots, 7\}$ and the constraint:

$$(5x \leq 18) \vee (-y + 2z \leq 3)$$

Its linearization is done by adding two boolean variables b_1, b_2 and the constraints:

- $b_1 + b_2 \geq 1$
- $5x - 18 \leq (\max\{5 \cdot 0, 5 \cdot 30\} - 18)(1 - b_1)$
- $-y + 2z - 3 \leq (\max\{-1 \cdot -5, -1 \cdot -2\} + \max\{2 \cdot -6, 2 \cdot 7\} - 3)(1 - b_2)$

Min/max constraints Given the variables $x_1, \dots, x_k$ such that $x_i \in \{l_i, \dots, u_i\}$, min/max constraints of form: Min/max constraints

$$y = [\cdot]\{x_1, \dots, x_k\}$$

are modeled as follows:

min Let $l_{\text{min}} = \min\{l_1, \dots, l_k\}$. Add k new boolean variables $b_1, \dots, b_k$ and impose:

$$\sum_{i=1}^{k} \big(b_i = 1\big) \land \big(y \leq x_i\big) \land \big(y \geq x_i - (u_i - l_{\text{min}})(1 - b_i)\big)$$

In this way, $(b_i = 1) \Rightarrow (x_i = \min\{x_1, \dots, x_k\})$.

max Let $u_{\text{max}} = \max\{u_1, \dots, u_k\}$. Add k new boolean variables $b_1, \dots, b_k$ and impose:

$$\sum_{i=1}^{k} \big(b_i = 1\big) \land \big(y \geq x_i\big) \land \big(y \leq x_i + (u_{\text{max}} - l_i)(1 - b_i)\big)$$

In this way, $(b_i = 1) \Rightarrow (x_i = \max\{x_1, \dots, x_k\})$.

Remark. This approach can also be applied to $y = |x|$, $y \neq x$, $y = kx$.

Unary encoding Encoding of the domain of a variable. Unary encoding

Given a variable x with domain $\mathcal{D}(x)$, its unary encoding introduces $|\mathcal{D}(x)|$ new binary variables b^x_k and imposes:

$$\sum_{k \in \mathcal{D}(x)} b^x_k = 1 \land \sum_{k \in \mathcal{D}(x)} k \cdot b^x_k = x$$

In this way: $b^x_k = 1 \iff x = k$.

Remark. This encoding provides a tighter search space of the linear relaxation of the problem and better encodes global constraints. On the other hand, it might introduce lots of new binary variables.

Example (`all_different`)**.** The encoding of `all_different`$(x_1, \dots, x_n)$ is done as follows:

- Encode each variable through unary encoding.
- For $j \in \bigcup_{1 \leq h < k \leq n} (\mathcal{D}(x_h) \cap \mathcal{D}(x_k))$ add the constraint:

$$\sum_{i=1}^{n} \alpha_{i,j} b^{x_i}_j \leq 1$$

where $\alpha_{i,j} = \begin{cases} 1 & \text{if } j \in \mathcal{D}(x_i) \\ 0 & \text{otherwise} \end{cases}$

For instance, consider the variables $x \in \{2, \dots, 11\}$, $y \in \{-5, \dots, 4\}$, $z \in \{3, \dots, 5\}$ constrained with `all_different`(x, y, z). We encode them using unary encoding and constrain b_j for $j \in (\{2, \dots, 11\} \cap \{-5, \dots, 4\}) \cup (\{2, \dots, 11\} \cap \{3, \dots, 5\}) \cup (\{-5, \dots, 4\} \cap \{3, \dots, 5\}) = \{2, \dots, 5\}$:

$$
\begin{array}{ccccccc}
1 \cdot b_2^x & + & 1 \cdot b_2^y & + & 0 \cdot b_2^z & \leq & 1 \\
1 \cdot b_3^x & + & 1 \cdot b_3^y & + & 1 \cdot b_3^z & \leq & 1 \\
1 \cdot b_4^x & + & 1 \cdot b_4^y & + & 1 \cdot b_4^z & \leq & 1 \\
1 \cdot b_5^x & + & 0 \cdot b_5^y & + & 1 \cdot b_5^z & \leq & 1
\end{array}
$$

Example (Array). $z = [x_1, \ldots, x_n][y]$ can be encoded as $z = \sum_{i=1}^{n} b_i^y x_i$.

Example (Bounded non-linearity). $z = xy$ with $y \in \{l_y, \ldots, u_y\}$ can be encoded as $z = [xl_y, \ldots, xu_y][y - l_y + 1]$

`<end of course>`