4 - LP

Linear Programming

@May 5, 2023

New technique for solving decision problems. Different approach to the solving involving variables and constraints. For these kind of problems you don’t need (lots of) data. You don’t need a powerful hardware. The solution you get (if you get it) is actually a solution: an assignment that satisfies all the constraints (maybe even optimal). So far, we’ve seen the “Computer-Science” way to solve these kind of problems (SAT, SMT…). Now we see something more mathematical-oriented: “Operation Research”. Not based on Logic. Central notion of inequalities and based on relaxation and cutting-planes rather than propagation. Still we have variables, constraints and objective functions (central role)

Operation Research is a well-established field (’40s) originated for military purposes and then applied in many other fields, based on mathematical techniques for enhancing complex decision making. One of the main influences is Linear Programming and its variants (programming is intended as planning, modeling, not coding). Linear Programming = OR framework based on solving linear equalities and inequalities (exponents are always one). Common problems → find optimal allocation for a limited number of resources. Several applications.

When we have two variables, we can represent the problem on a Cartesian plane → equalities are straight lines and inequalities are half planes (in n dimension we talk about hyperplanes and half spaces). A feasible solution is an assignment that satisfies all the constraints, one in which all the points are within the intersection of all half-spaces defined by inequalities. The feasible region is always a convex polyhedron → nice, because you don’t get stuck in local minima/maxima. Can be empty, bounded or unbounded. The goal is to find the optimal solution, meaning the point within the feasible region where the objective function has maximal value.

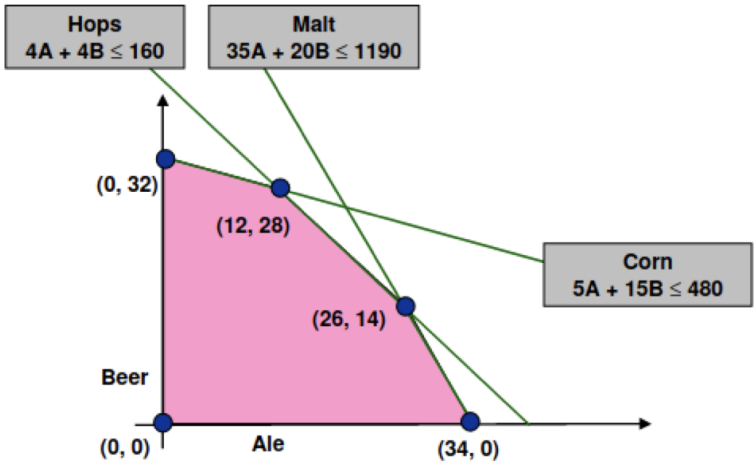
Geometric Interpretation for the Baking Problem. The dashed line represents the objective function, here set to the minimum bound (0). "Tuning the isolines" means setting the objective function to a z which will be slid up (in this case) to find the optimal solution

A graphical approach would be "tuning the isolines" → move the objective function to find the optimal solution. This solution, though, might be inconsistent with the problem formulation.

For sure, I know that the solution will be at an extreme point.

> 👩‍💻 **Extreme Point Property**
> If an optimal solution for a LP problem exists, then there is one at the extreme point of its feasible region

In this scenario, you can always define a procedure that goes from extreme point to extreme point to find eventually a solution. The number of points though can be exponential.



Feasible Region defined for the Brewery Problem

Canonical form of a LP problem:

$$\max \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} a_{i,j} x_j \leq b_i \quad 1 \leq i \leq m$$

$$x_j \geq 0 \quad 1 \leq j \leq n$$

We have a linear objective function in n non-negative variables to maximize, subject to a number m of linear inequalities. You can write it in a matrix form (vectorize). We are considering real values and coefficients. The only domain constraint is on the non-negativity of the variables.

Standard form (we need it for Simplex Algorithm):

$$\max \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} a_{i,j} x_j = b_i \quad 1 \leq i \leq m$$

$$x_j \geq 0 \quad 1 \leq j \leq n$$

The only difference is that we don’t have inequalities, but equalities. Not so surprising, because in the 2D dimension, we have just lines and the feasible region is made of a contour and to find the optimal solution you just need to move from vertex to vertex (as the optimal solution will be a vertex)

How to convert one to the other? For each inequality we just need to add a new set of variables (slack variables). The dimension of the problem, though, increases to an $(n+m)$-dimensional problem.

Simplex Algorithm

Idea that we don't need to know the feasible plane if the solution is at vertex. You start from a given extreme point and move to another with the aim to increase the objective value. If we can no more improve, we've found a solution.

Concept of basis (from Linear Algebra) is fundamental. We select m variables (the inequalities that we have). We call these variables our basis if the corresponding columns form an invertible matrix.

First step of the method is to select a basis. Then you can rebrand the problem by separating basic variables and non-basic variables (associativity of the sum holds). If you put the non-basics to null vector, the whole non-basic set of variable goes away and you have the invertibility property, which you can use to obtain the solution (basic solution for corresponding basis). How do we switch bases? Use neighborhood search: fix $m-1$ variables and change the remaining.

Not all basic solutions are feasible. A solution is considered feasible iff $\left(\forall_{i=1}^{m}\right)\left(x_{\mathcal{B}}\right)_i \geq 0$. In particular, it is called non-degenerate if all the variables for the solution are strictly greater than 0.

Repeatedly consider BFS (basic feasible solution) until it works out. How do we choose the entering variable x^{in}? Choose the variable that potentially increases more the objective value. And the leaving variable x^{out}? Minimum Ratio Rule: choose the variable in the row whose ratio of known terms w.r.t. the coefficient of the entering value is minimum.

In the tableau representation, it means selecting a pivot column (the entering variable) and a pivot row (leaving variable). Then you use the new values to reformulate the problem.

- Selecting x^{in}, x^{out} resp. means choosing a pivot column and a pivot row from the tableau representation of $\max(Z)$ s.t.:

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

- From 2nd row we get $B = \frac{480-5A-S_C}{15} = 32 - \frac{1}{3}A - \frac{1}{15}S_C$ and we substitute it in all other equations:

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

Pivoting example from the Brewery Problem

When there's no way to increase the objective value, you stop.

The optimality check is sufficient, but not necessary (we might have an optimal solution even if one reduced cost - reduced costs are the coefficients of the objective function row - is greater than 0)

Feasible region for a LP problem in canonical form is an intersection of half-planes which denotes a convex polyhedron = set of all the points (tuples of values) that satisfy the constraints. This is not an objective function. The optimal region instead consists of all the points in the feasible region s.t. if I take another point in the feasible region, it cannot have value greater than it. By definition, the optimal region is a subset of the feasible region. If the optimal region is finite, its cardinality is 1: the optimal solution (an extreme point)

Unbounded problem → there are solutions but there is no optimum (no upper bound). The feasible region is an unbounded polyhedron. The Simplex Method performs an unboundedness check.

Given LP P in standard form, we can (roughly) summarize the simplex method as:

0. Let $k \leftarrow 0$, let $\mathcal{B}_0$ a feasible base for P and go to 1.
1. If BFS of $\mathcal{B}_k$ is optimal then STOP, else go to 2.
2. If P unbounded then STOP, else go to 3.
3. Select an entering variable $x^{in} \notin \mathcal{B}_k$ and go to 4.
4. Select a leaving variable $x^{out} \in \mathcal{B}_k$ and go to 5.
5. Let $\mathcal{B}_{k+1} = \mathcal{B}_k \cup \{x^{in}\} - \{x^{out}\}$ and reformulate P accordingly. Let $k \leftarrow k+1$ and go back to 1.

Summary of the steps for the Simplex Algorithm

If all the possible BFS are non-degenerate, the simplex method always terminates in a finite number of steps. The number of vertices is finite and at each step we move from one vertex to another, always strictly improving the objective value. Otherwise, stalling is possible (we repeatedly change basis without improving) → termination can be though guaranteed by setting rules preventing infinite loops.

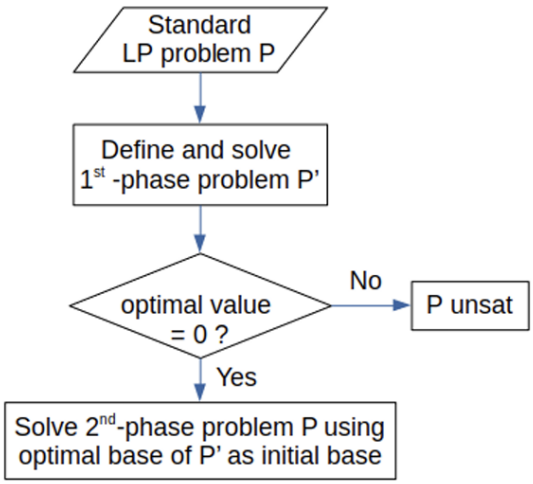
LP problems are not as hard as SAT → The worst case complexity for Simplex Algorithm is $O(2^n)$ but other algorithms can be found to solve this in polynomial time. It's not NP-Hard.

So far, we've never assumed anything about satisfiability (is there any feasible base?)

Two-phase Method

1st phase P' is obtained from the original problem (which is 2nd phase): you introduce as many variables as the number of constraints, maximize this new negative value $-\sum_{i=1}^{m} s_i$ (upper-bounded by 0) and for each equality having a term non negative add $+s_i$ else you add $-s_i$

The objective will have upper bound 0 and is always satisfiable (I have a feasible basis with the s values). The feasible region of the original problem is not empty if and only if the optimal solution of P' has value 0. Otherwise, P is unsatisfiable. If the problem is satisfiable, you can use the solution as a starting solution for the original problem



Duality Concept

@May 11, 2023

What is the dual of something? Symmetrical equivalent of the same property (e.g. the dual problem of the Brewery Problem is the Entrepreneur problem, in which you want to minimize the cost instead of maximizing the profit (opposed but in a way equivalent to the other)). It is a concept which can be extended also to other concepts, like e.g. $\mathcal{T}$-satisfiability and $\mathcal{T}$-validity of formulas (e.g. to prove that a formula is satisfiable in a certain theory, you need to prove that its negation is not valid).

Let $P : \max(cx)$ s.t. $Ax = b, x \geq 0$, with $b \in \mathbb{R}^m, x \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}$ the **primal** problem. Its **dual** problem $\mathcal{D}(P)$ is $\min(by)$ s.t. $A^T y \geq c$ with $y \in \mathbb{R}^m$:

- a variable y_i for each constraint $\sum_{j=1}^{n} a_{i,j} x_j = b_i$ of $P, i = 1, ..., m$
- a constraint $\sum_{i=1}^{m} a_{j,i} y_i \leq c_j$ for each variable x_j of $P,\ j = 1, ..., n$

If I transpose it I get the original problem: $\mathcal{D}(\mathcal{D}(P)) = P$. The dual of the dual is the primal problem. Two important properties are:

- Weak Duality → the objective value of any feasible solution of the primal is $\leq$ of the objective value of any feasible solution of the dual

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

  ⇒ The dual acts as an upper bound for any feasible solution of the primal (and the primal is a lower bound for the objective value of the dual). by decreases until a minimum value is eventually reached; at the same time cx increases until a maximum value is eventually reached. If the original problem is unbounded, the dual is unfeasible; at the same time if the dual is unfeasible, the primal is unsatisfiable.

- Strong Duality → if both primal and dual are feasible, they have the same optimal values (e.g. the brewery/entrepreneur example).

$$\mathcal{F}_P, \mathcal{F}_{\mathcal{D}(P)} \neq \emptyset \implies (\forall x^* \in \mathcal{O}_P, \forall y^* \in \mathcal{O}_{\mathcal{D}(P)})\ cx^* = by^*$$

We have primal unbounded ⇒ dual unfeasible, but not necessarily the opposite (primal unfeasible $\not\iff$ dual unbounded).

	$\mathcal{D}(P)$ bounded	$\mathcal{D}(P)$ unbounded	$\mathcal{D}(P)$ unfeasible
P bounded	✓	✗	✗
P unbounded	✗	✗	✓
P unfeasible	✗	✓	✓

Summary of the possible combinations between the Primal and the Dual

We can now apply the dual simplex → we can avoid to compute the dual by directly running the dual simplex on the primal. While in the primal simplex we start from a feasible basis and we aim for the optimal, preserving feasibility, in the dual simplex, we start from an optimal basis, which might be not exactly feasible, and go into the feasible, while preserving optimality. Now our x^{out} is the variable with the minimum negative value and the x^{in} is the variable with the maximum negative ratio.

We can have also the Primal-Dual hybrid approach, in which from a dual solution y we look for a primal solution x satisfying certain optimality conditions (we solve a "reduced problem" P' if possible; else, we derive a new solution y' from $\mathcal{D}(P')$ and repeat).

Duality can be used for theoretical purposes. You can reduce the number of variables, if you have more variables than constraints. You can find both upper and lower bounds for the objective problem.

Sensitivity Analysis

Also called Post-Optimality Analysis. It is the problem of post-processing a solution: how and if the optimal solution changes if I change (perturb) the input parameters. The inputs are the coefficients of the variables, objective functions and the known terms.

The trivial answer would be to solve again the problem. Through post-optimality analysis one could avoid this step.

$$\bar{P} : \max(cx) \text{ s.t. } Ax = b, x \geq 0$$

Changing the Known Term b

Can imply unfeasibility or that the current solution is no longer optimal. Changing the known term means changing the right-hand side of (in)equalities.

E.g. in the brewery problem one would change the available quantities and could fall into:

- Decreasing → can impact on the feasibility and the optimality
- Increasing → improves the objective value

Changing Objective Value Coefficients c

Also called changing costs. This does not affect the feasibility, because constraints are still the same and the feasible region stays the same, but affects the optimality:

- I can have a loss of optimality
- The solution is still optimal, but associated to a different objective value

This happens if the coefficient refers to a basic variable. A non-basic variable has 0 value and it is not considered in the optimal value (no change in optimality)

Changing Constraints

Evaluating if changing a constraint will affect the optimality depends on if the coefficient refers to a variable in the optimal basis:

- If not, the solution is still feasible, but we may lose optimality as the objective function can change
- If it is a basic variable, to evaluate the impact we need to resolve the problem

(Mixed) Integer Linear Programming

@May 12, 2023

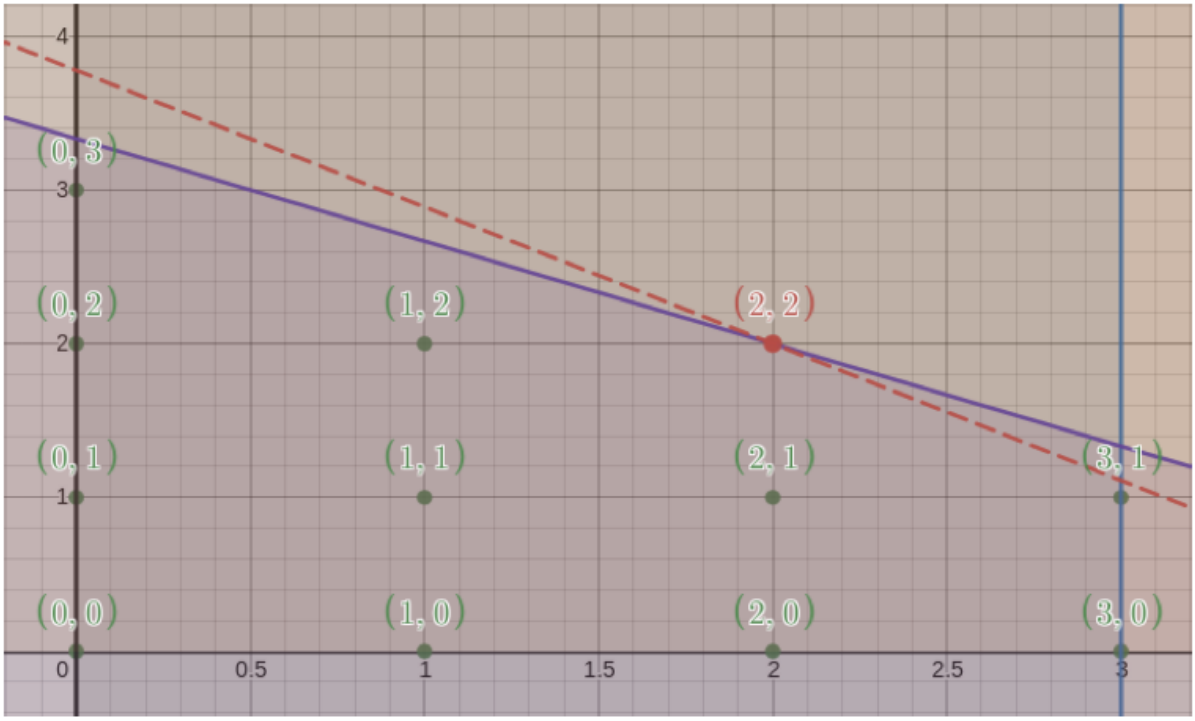
Until now, we have considered Linear Programming in the $\mathbb{R}$ domain, which we have shown that can be solved in "typically polynomial" time (Simplex sometimes, Ellipsoid Methods, Interior Point Method…). This holds for variables in domains $\mathbb{R}$ or $\mathbb{Q}$. If we introduce the constraint that at least some variables are in a domain subset of $\mathbb{Z}/\mathbb{N}$ the problem becomes harder → we don't have extreme points anymore to exploit. We don't have a polyhedron anymore, but a set of points as our feasible and optimal regions. Rounding won't cut it as heuristic for this problem.

An Integer Linear Programming problem has a standard form:

$$\max \sum_{j=1}^{n} c_j x_j$$
$$\text{s.t. } \sum_{j=1}^{n} a_{i,j} x_j = b_i \;\; 1 \leq i \leq m$$
$$x_j \geq 0,\; x_j \in \mathbb{Z} \;\; 1 \leq j \leq n$$

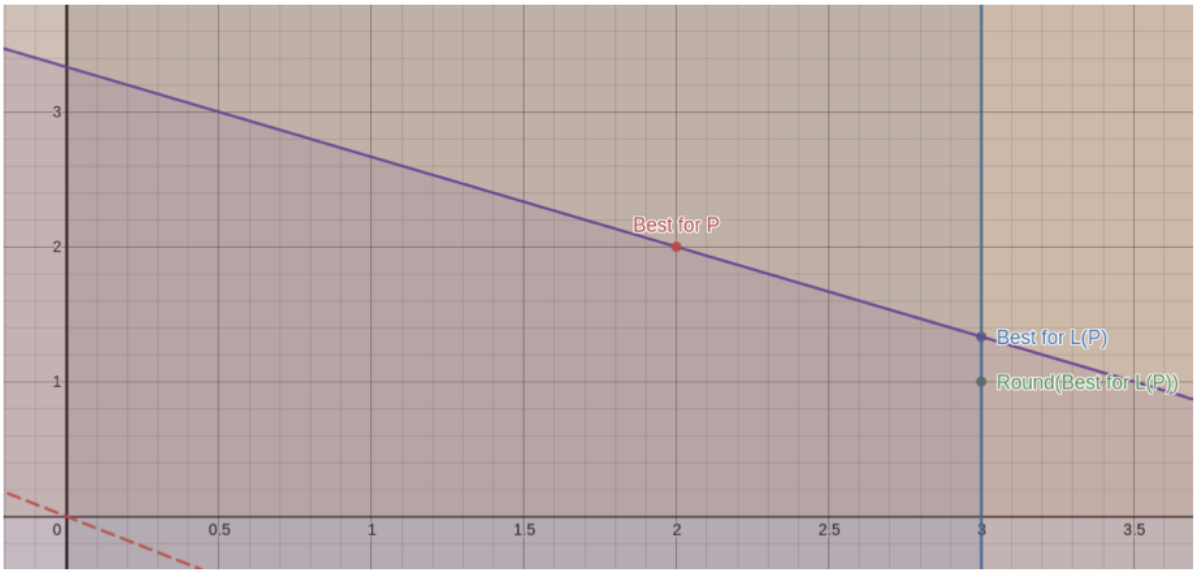
If we require only a subset of the variables to be integer, we talk about MIP problems (Mixed-Integer Programming).

Important concept of Linear Relaxation: we remove the integrality ($\in \mathbb{Z}$) constraints $\rightarrow$ we go back to the standard LP problem as the set of solutions of the original problem is always contained in its relaxation (e.g. the Baking problem).



We can see the points are all contained in the trapezoid representing the feasible region

Solving the Linear Relaxation does not mean finding a solution, as some points in the relaxation do not belong to the original problem.



Difference between the Best Solution for the Linear Relaxation (in blue) and the Best Solution for the MIP (Red)

Rounding the optimal solution of the linear relaxation is not sound in general. We might get a solution which is not even feasible. If the Linear Relaxation is unsatisfiable, for sure we won't find a solution for the original problem (we can use the Linear Relaxation to detect unfeasibility for the original problem). If the Linear Relaxation is unbounded, we can have that the problem could be bounded, unbounded or unsatisfiable.

Adding integrality constraint means that the problem becomes NP-Complete, meaning that it is in NP (solvable by a Non-Deterministic Turing Machine in polynomial time $\equiv$ certifying a solution takes polynomial time), but is among the hardest problems in NP (NP-Hard).

We can get an equisatisfiable P_{MIP} problem from any generic SAT problem in CNF P_{SAT}. We have that MIP is at least as hard as SAT and from this, we have that the problem is NP hard and NP-Complete as it is solvable with a NDTM.

From any generic SAT problem in CNF P_{SAT} with clauses $C_1, \dots, C_m$ and literals $\ell_1, \dots, \ell_n$ we get an equisatisfiable MIP problem P_{MIP}:

$$\begin{array}{lll} \max & 0 & \\ \text{s.t.} & \sum_{j=1}^{n} a_{i,j} x_j \leq |C_i^-| - 1 & 1 \leq i \leq m \\ & x_j \in \{0,1\} & 1 \leq j \leq n \end{array}$$

where $C_i = C_i^- \cup C_i^+$ and C_i^- / C_i^+ are the negative/positive literals of C_i and $a_{i,j} = \begin{cases} -1 & \text{if } \ell_j \in C_i^+ \\ +1 & \text{if } \neg\ell_j \in C_i^- \\ 0 & \text{otherwise} \end{cases}$ for $i = 1, \dots, m$, $j = 1, \dots, n$

E.g. $\ell_1 \vee \neg\ell_2 \vee \ell_4 \vee \neg\ell_5 \implies x_1 + (1 - x2) + 0 \cdot x_3 + x_4 + (1 - x_5) \geq 1 \implies x_1 - x_2 + x_4 - x_5 \geq -1 \implies -x_1 + x_2 - x_4 + x_5 \leq 1$

The reduction from P_{SAT} to P_{MIP} takes polynomial time ($O(mn)$). Plus, we have that P_{SAT} is feasible $\iff$ P_{MIP} is feasible, so we have also that any solution of P_{MIP} can be mapped back into a solution of P_{SAT} in polynomial time ($O(n)$).

In general, finding a solution is difficult. We have these classes of strategies:

- Exact Algorithms → branch-and-bound. These always guarantee that sooner or later they will find an optimal solution or a decision problem, but may take exponential time.
- Approximation Algorithms → guarantee in polynomial time an optimal or suboptimal solution at most ρ times worse than the optimal one, where ρ is an approximation factor. We have a formal proof of this statement.
- Heuristic Algorithms → no guarantee of (sub)optimality or polynomial runtime. However, they may work well in practice: they can find good solutions in reasonable times, according to empirical evidence.

Branch-and-Bound Algorithm

Based on a “Divide and Conquer” approach in which you split a big problem into sub-problems until success or failure occurs, with the overall solution derived from the solution of the sub-problems

- Branch phase → you explore the search tree (open new branches related to the subproblems)
- Bound phase → compute the bounds of subproblems and possibly prune the tree if current solution is not improvable.

Is more of a general paradigm that can also be applied to other NP-Hard problem

We start from $\mathcal{L}(P)$, the linear relaxation, and we solve it, getting some solutions (we might be absurdly lucky to find a solution we can stop in). Usually we have some $\beta_k \notin \mathbb{Z}$.

$$\begin{cases} P_1 = P_0 \cup \{x_k \leq \llcorner \beta_k \lrcorner\} \\ P_2 = P_0 \cup \{x_k \geq \ulcorner \beta_k \urcorner\} \end{cases}$$

Then we pick a x_k so that $\beta_k \notin \mathbb{Z}$ and we branch and consider two sub-problems:

- The original problem $x_k \leq$ the floor
- The alternative problem in which we consider $x_k \geq$ the ceil

This is a partition of the space (the two subsets united give me the original space and the intersection is empty - even if it is not, it would still be valid, but we would explore some solutions twice).

Now we solve these new problems and obtain new solutions. When we find a solution, if it is integral, it will be optimal also for the original (as we know that $\mathcal{F}_P = \mathcal{F}_{P_1} \cup \mathcal{F}_{P_2}$), otherwise we branch again.

We build essentially search trees rooted in the P_0 with my edges sub-problems of the original problems (branches). Where's the bound? We remove unfeasible solutions as we find them (we don't branch from unsatisfiable solutions). When we close a node, we need to consider (when we can say stop?) the bound: if we get to a node with not necessarily an integral solution and we realize that we cannot improve ($\text{obj.val} \leq z^*$), we can close the branch. We update at each branch the objective value with the better one we can find (initially $z^* \leftarrow -\infty$, then it is improved with what we find)

Leaves are called fathomed nodes, current optimal solutions are the incumbent solutions.

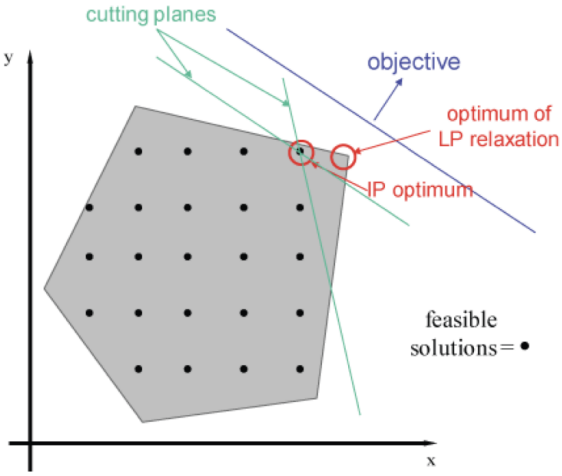
Typically, BB works well in combination with other techniques (e.g. Presolve, Cutting Planes…)

Presolve means reformulating a problem before the solving process to possibly reduce its size (e.g. by improving bounds and/or getting rid of variables for which you can surely derive the value domains). You reduce the size without altering the set of feasible solutions (unlike Symmetry Breaking)

Cutting Planes Technique

In addition to or instead of using the Branch-and-Bound approach on sub problems, we cut and cut (we add further constraints entailed by the original problem) until we get to a small feasible region that only contains the optimal solution. You can always find a cut that separates the optimal solution from the feasible region, though there could be more than one (we have guaranteed existence but not uniqueness).

Formally, a cut for a MIP problem P in standard form is an inequality $px \leq q$ such that $py \leq q$ and $pz \geq q$ for each $y \in \mathcal{F}_P$ and $z \in \mathcal{O}_{\mathcal{L}(P)}$.



Cutting Planes

Gomory's Cut

Suppose we have an optimal basis $\mathcal{B}^*$ for the linear relaxation. The variables can be rewritten in a new form in which you isolate them and write them in terms of the non-basic variables $x_{i_{m+1}}, ..., x_{i_n}$.

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

The β_k can be 0 and can also be not an integer. If this is the case, we generate a cut that separates this from the feasible region of the original problem.

The mathematical form of the cut is an inequality in which we take the mantissa of β_k (If you subtract to the number its floor, you get $0 < f_k < 1$ - we have the guarantee that the number is not an integer)

$$-f_k + \sum_{j=1}^{n-m} f_{k,j} x_{i_{m+j}} \geq 0$$

where for $j = 1, ..., n-m$ and $k = 1, ..., m$:

- $f_k = \beta_k - \llcorner \beta_k \lrcorner$ (mantissa of β_k)
- $f_{k,j} = -\alpha_{k,j} - \llcorner \alpha_{k,j} \lrcorner$ (mantissa of α_{kj})

We add a new slack variable y_k that excludes the solution using the cut.

Now we solve the resulting problem using the dual simplex. The basis we have is not feasible anymore, so this new restart can't be used for the primal problem. However, it was optimal for it, so it is dual feasible → [goes from optimal solution to optimal solution, preserving optimality and looking for feasibility]. In the first round we will have y_k leaving variable and x_i as entering variable. If the dual simplex finds an optimal solution which is integral, or is unbounded (meaning the primal is unfeasible), we stop, otherwise we keep going.

Originally considered not effective. Proved to be very effective in combination with branch-and-bound. Runs BB and if an optimal solution is not integral we add cuts to refine the region. Some cuts can be global (valid for all solutions of P) and some local (valid for sub-problems).

Bender's Decomposition

The approach of cutting planes generates new constraints. We can see, when we use the matrix notation of a problem, that this is affine to say that it is a row generation method = when you add a constraint, you add a new row to the matrix of the coefficients.

Bender's Decomposition is a row generation method in which you have a master problem P (minimization) and some sub-problems to maximize. The idea is that you iteratively fix some variables and solve the dual of the residual subproblems, with the objective to try to reduce the span of the bounds to some epsilon. We can integrate with CP or SMT through Logic-Based Decomposition.

We also have column-generation methods: we start with a subset of variables and repeatedly add variables until the objective value cannot be improved. We start from the assumption that only a small subset of variables is actually useful.

Original problem with m inequalities and n variables: we separate into $\mathbf{x} \in \mathbb{R}^p$ and $\mathbf{y} \in \mathbb{R}^{n-p}$:

$$\begin{gathered}\min c^T \mathbf{x} + d^T \mathbf{y} \\ \text{s.t. } A\mathbf{x} + B\text{y} \geq b \\ \mathbf{x} \geq 0,\ \mathbf{y} \in Y\end{gathered}$$

We have the non negativity of $\mathbf{x}$ and $\mathbf{y} \in Y$ (the feasible set of $\mathbf{y}$- e.g. the Integrality constraint). So the first is a standard LP problem, the other sets the domain. Suppose that we fix the $\mathbf{y}$ variables, we get a residual problem.

$$\begin{gathered}\min c^T \mathbf{x} + d^t \overline{y} \\ \text{s.t. } A\mathbf{x} \geq b - B\overline{y},\ \mathbf{x} \geq 0\end{gathered}$$

At this point we can exploit duality.

$$\begin{gathered}\max (b - B\overline{\mathbf{y}})^T \mathbf{u} + d^T \overline{\mathbf{y}} \\ \text{s.t. } A^T \mathbf{u} \leq c, \mathbf{u} \geq 0 \\ A^T \in \mathbb{R}^{p \times m}, u \in \mathbb{R}^m\end{gathered}$$

Can see the original problem as a minmax:

$$\min_{y \in Y} [d^T y + \max_{u \geq 0} \{(b - By)^T u | A^T u \leq c\}]$$

We have initially an empty set of constraints and then we repeatedly generate cuts from the max sub-problem:

- unbounded → the residual problem is unfeasible, so we generate a cut to exclude $\overline{y}$
- unfeasible → the residual problem is unbounded or unfeasible, we stop
- optimal → we update the upper bound of the min problem, because a solution for the linear problem is a solution for the master problem, and we generate a new constraint because of duality. We repeat until the bounds gap is less than a given constant, then we STOP.

Heuristics Methods

You can have different ones. The goal is to find a solution in reasonable time: they are inherently experimental, with weak theoretical guarantees. They are constructive methods (often based on evolutionary algorithms). Also local search methods (start with a complete solution and iteratively improve) and meta-heuristics (heuristics for selecting, combining, tuning or generating other heuristics).

MIP can use many at the same time, e.g. rounding, diving (rounding and bounding/fixing some variables, then re-solve the linear relaxation at each node of the search tree) or Sub-MIPing (restrict to sub-problem by fixing/bounding some variables and solve it)

Interesting: warm start. We begin with a (partial) solution which has been already computed to kick-start the solver. Not necessarily a new incumbent, but might be efficient to tighten the bounds of the variables.

MIP Extensions and Technology

@May 18, 2023

Non-Linear Programming

All the problems we have seen so far have the particular form of Standard LP. But of course we can have something different (e.g. inequalities/logical constraints, which are non linear e.g. allDifferent, which is quadratic)

If it is known that the nonlinear constraints are in a given form, one could use a specific Non-Linear Programming paradigm; else, one could encode the problem into an equisatisfiable linear problem.

A Non-Linear Program is of the form:

$$\min f(x)$$
$$\text{s.t. } g_i(x) \leq 0 \;\; i = 1, ..., m$$
$$h_j(x) = 0 \;\; j = 1, ..., p$$

We have a function to minimize, a number of inequalities and a number of equalities, where $x \in \mathbb{R}^n$ and at least one among f, g, h_j is non-linear.

Newton's Method

If f is constant, $m = 0$ and we have 1 equality, one can apply the Newton-Rhapsod Method, which uses derivatives (slope of the function in a given point). f function f' derivative, $x_0 \in X$, we can compute the tangent as a linear approximation of f.

$$t_0(x) = f'(x_0)(x - x_0) + f(x_0)$$

The intersection with the x-axis is at a point x_1 such that $t_0(x_1) = 0$ and we obtain that $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$. So we can iterate to whenever. If x_0 is close enough to a zero of f and we don't get stuck, we converge to a zero of the function.

Pros: if it converges, it is fast

Cons:

- we need to analytically compute $f'(x)$
- We need to choose a good x_0
- We don't know how close we are to a solution
- We can end up stuck in a stationary point.

It can also be generalized to n-ary functions.

E.g. if the function is twice differentiable, the method can be applied to f', we look for points where the derivative is equal to zero. Instead of the tangent we consider a parabola

$$p_k(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2} f''(x_k)(x - x_k)^2$$
$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

If $f(x) = ax^2 + bc + c$ we need only one step to converge. We can use the generalization of Newton's to find local optimums.

If we are in a n-dimensional space, we can use the gradient ∇f(a vector) instead of the prime derivative. And instead of the second derivative one could use the Hessian $\nabla^2 f$ (a

matrix of $\mathbb{R}^{n \times n}$).

Used for interior point approaches (seen to solve LP problems). It is a second order approach as you use 2nd order derivatives to compute trajectories. The ones based on the gradient are first-order methods. You can avoid derivatives altogether (e.g. dichotomic search)

Gradient Descent

Gradient Descent is a well known first-order approach to find a local minimum of a differentiable function $f(x)$. It is commonly used to train ML models and NN to minimize a cost function. You start from an initial point x_0 and move using the gradient $x_{k+1} = x_k - \lambda \nabla f(x_k)$. The $k+1$-th step is proportional to:

- direction $-\nabla f(x_k)$ (where to go to minimize the error)
- learning rate λ (size of the step towards the minimum error)

Used for linear regression, in which we estimate the relation between n independent variables $x_i \in \mathbb{R}^d$ (d is the number of features) and a number of dependent variables y with linear predictors $\theta_0 + \theta_1 x_{i,1} + ... + \theta_d x_{i,d}$ for $i = 1, ..., n$. We want to minimize the error of the estimate (MSE). One independent variable ($d = 1$) → simple linear regression

For simple regression we have:

$$\theta_0 = \bar{y} - \theta_1 \bar{x}$$
$$\theta_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$

Also this method can be generalized for n-dimensional spaces.

If n and d are big, computing $\Theta = (\theta_0, ..., \theta_d)$ analytically can be computationally expensive. We need to minimize $||Y - X\Theta||^2$ with $Y \in \mathbb{R}^n, X \in \mathbb{R}^{n \times (d+1)}$. We can work iteratively by minimizing MSE with gradient descent:

$$\Theta^{(k+1)} \leftarrow \Theta^{(k)} - \lambda \nabla f(\Theta^{(k)}),\ k = 0, 1, ...$$

where $\Theta^{(0)}$ and λ are arbitrary.

GD can be used in different modalities:

- Batch → model updated after evaluating all training set (epoch)
- Stochastic → subset of N samples randomly selected from training set - if $N > 1$, it is also known as Mini-Batch.

Apart from Neural Networks, GD can be used to train linear classifiers by minimizing errors on misclassified examples → it is less robust than SVMs and Non-Linear Logistic Classifiers are better for binary classification. Newton Method has better convergence, but slower because it has to compute the Hessian Matrix.

Linearization

Viable alternative approach. As we already know how to solve linear problems, we linearize the nonlinear constraints of a problem to get an equisatisfiable MIP problem. Works if the variables are bounded.

Reification

When we have logical combinations of constraints.

Reification = when you have a general constraint in k variable, a full reification means to have a boolean variable s.t. the variable is true implies the constraint is satisfied.

$$b \text{ is a boolean variable s.t. } b = \text{True} \iff C(x_1, ..., x_k)$$

It is a channeling between the constraint and the boolean variable (it is the way if-else are handled in MiniZinc). We talk about half-reification if we only have a logical implication.

When we have a logical disjunction of constraints, for each of them we introduce a boolean variable b_i that (half-)reifies the constraints and impose that at least one of them is true ($\sum_i b_i \geq 1$). We need the connection. One thing we could use is the Big-M trick

Suppose we need to encode a disjunction of constraints. We introduce the binary variables and the imposition and as the constraints are linear, we can express them in a specific form.

Say we have $C_1 \vee ... \vee C_m$ and $b_1, ..., b_m \in \{0, 1\}$, we impose $\sum_{i=1}^{m} b_i \geq 1$ and for each constraint $C_i \equiv \sum_{i,j} \alpha_{i,j} x_j \leq \beta_i$ we add a constraint:

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

So for each of these constraints, we add: on the lefthand side we have the inequality and M is (the Big-M number) a big enough constant. Each of the b_i is either 0 or 1. If $b_i = 0$, we're multiplying for 1. So if M is big enough, the inequality is always satisfied. If $b_i = 1$ the constraint must hold, as we have $M \cdot 0$ (what we have is actually the constraint). If M is too big, the solving process gets slower (bigger feasible region).

This approach can be used to linearize a $\min$ / $\max$ constraint

Alternative approach is a Unary Encoding. If $D(x)$ is the domain of x we introduce $|D(x)|$ binary variables $b_k^x \in \{0,1\}$ and impose:

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

This means that if $b_k^x = 1$ we have $x = k$ (and viceversa). We have thus a tighter linear relaxation, with a better encoding of global constraints. The con is that we have lots of binary variables if the domain is big.

MiniZinc Encodings and Solvers

If you use MiniZinc to compile a CP model with finite domains you can have an equisatisfiable FlatZinc instance with linear constraints only → useful, because you can use any linear solver to solve, but not necessarily efficient because of automatic translation. We can access the file with definitions for MIP, thus we can redefine.

Function `eq_encode(var int:x)` enforces 1 for integer variable 1. Common subexpression elimination ensures that binary variables are reused if x is encoded again (papers of CP, 2016)

MIP Solvers for the project: need at least some free solvers (need docker files and need a license for reproducing):

- COIN-BC (CBC) → Based on Branch-and-Cut, included in MiniZinc and found on GitHub (actively maintained). Default solver for PuLP (Python Module for LP)
- HiGHS (advised) → Based on high performance dual revised simplex implementation and its parallel variant. Freely available.

Other not included in the MiniZinc bundle:

- IBM ILOG CPLEX
- Gurobi
- SCIP
- FICO Xpress

(NB → to use commercial solvers one needs to be able to write a script to install the license)

For the MIP part one might use different languages, like something more “mathematical-oriented” (e.g. AMPL) or LP libraries embedded in other languages (one can use Simplex method directly on Excel 😮) Or simply just use the API of one particular solver.

AMPL → A Mathematical Programming Language. Algebraic Modeling Language. Supports separation between model and data. Problems are in format `.nl` and it is supported by many

solvers.