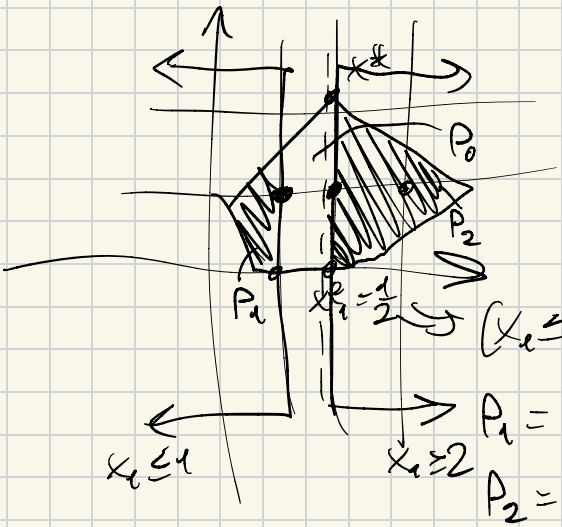
14/11

# BRANCH-AND-BOUND




$x_i^*$ frazionario (se + presenti, scelgo)

$(x_1 \leq 1)$ o $(x_1 \geq 2)$

$P_1 = \{x \geq 0 \mid Ax = b, x_1 \leq 1\}$

$P_2 = \{x \geq 0 \mid Ax = b, x_1 \geq 2\}$

confronto e prendo sol. ott. migliore

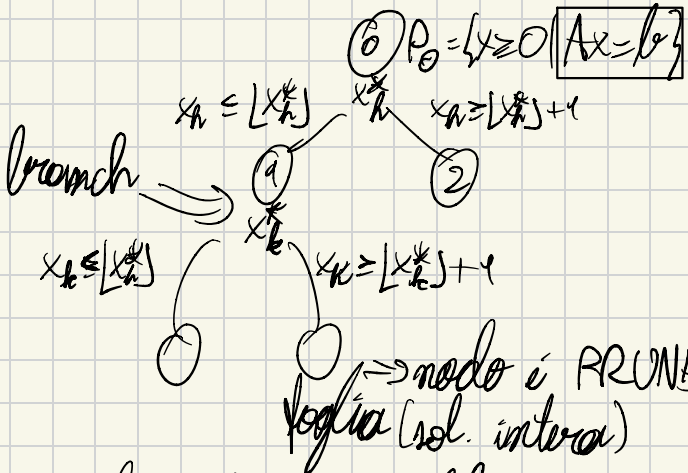
"ricorsivo" => se trovo sol. frazionaria, divido ancora

approccio DIVIDE AND CONQUER

anche detto IMPLICIT ENUMERATION

Rappresentabile con BRANCHING TREE

albero orientato




$\Rightarrow$ nodo è PRUNED/FATHOMED foglia (sol. intera)

Non serve branch se non abbiamo comp. frazionarie

condizioni per pruning:

- $x^*$ intero
- LP-relax. infeasible

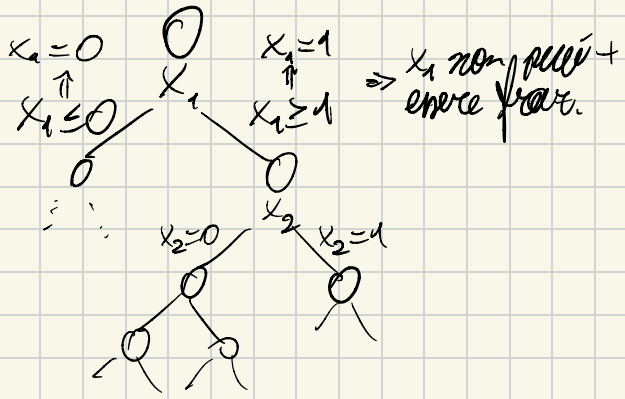
Bisogna risolvere tante LP

convergenza di metodo

supponiamo variabili $x_i \in \{0,1\}$ ($x_j \in [0,1]$ intere)

caso peggiore possibile

$x_1 \in (0,1) \Rightarrow$




livello $n$: tutte var. fissate

non avrò mai $> n$ livelli

branching tree con $2^n$ nodi

per prob. binari, sempre # finito di it.

Var. non binarie (e.g. $x_3 \in \{0, \dots, 7\}$)

rappresentare in forma binaria:

$x_3 = x_3^0 \cdot 1 + x_3^1 \cdot 2 + x_3^2 \cdot 4$ (come forma base 2)

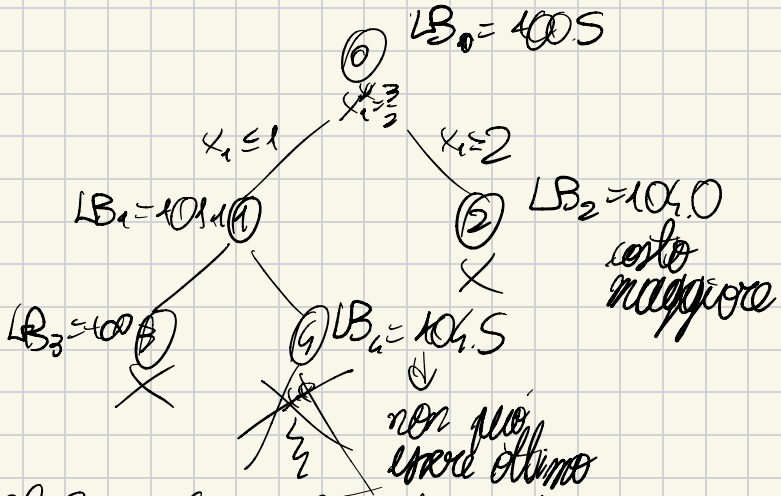
sempre # finito di var.

Stabilità numerica

non c'è problema: ad ogni it. problema originale con condizioni di bound non aggiungiamo numeri in tableau

Punto focale: BOUND

Es:




var. globale: miglior sol. intera finora (INCUMBENT) contiene miglior sol. $X^{OPT}$ e suo costo $Z^{OPT}$

Inizio: ZOPT = +∞, XOPT = vuoto

Quando trovo sol. intera:

se costo < ZOPT, aggiorno XOPT, ZOPT

se costo ($LB_k$) > ZOPT, posso tagliare ramo (prune node)

Nuove condizioni:

- $x^*$ intera
- no $x^*$
- $LB_k \geq ZOPT \Rightarrow$ riassume le altre

Problemi di implementazione

Scelta di branching var. $x^*_h$:

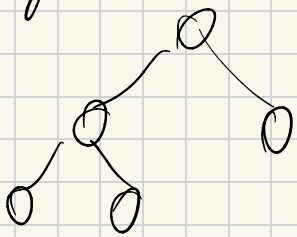
no teorema $\Rightarrow$ spesso si taglia prima su var. importanti per modello

oppure selezionare $\ell(x^*_h)$ il + vicino a 0,5 (molto popolare tempo fa)

albero equilibrato

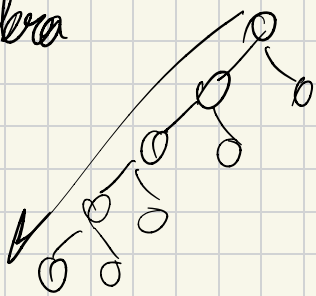
non molto migliore di scelta random

# Strategia di ricerca



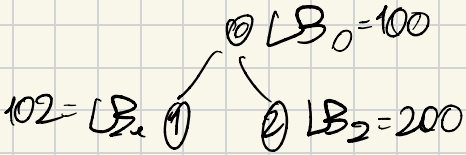
quale nodo processare dopo?

- DEPTH-FIRST: prima scopro q, x
  si va subito a fondo ⇒ ottenere sol. intera
  buona policy
  pericoloso (e.g. ; sol. ottima in altro albero)



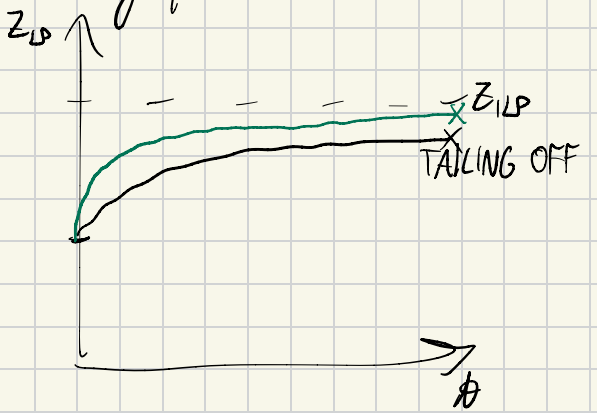
- BEST-BOUND FIRST

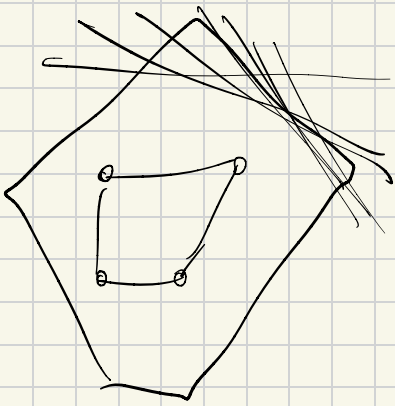
  si sceglie LB migliore, ma non si va molto a fondo



15/11

Cutting plane

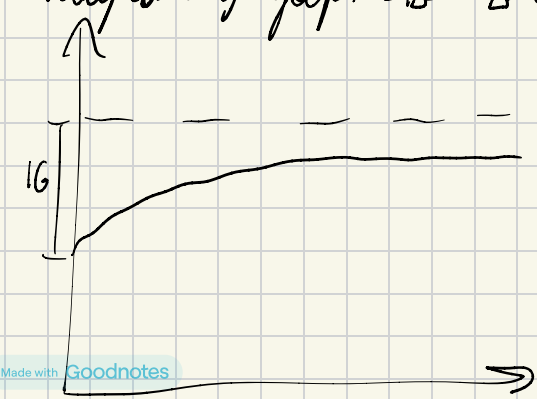


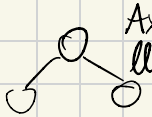



Linea verde: miglioramento metodo
primi cut potenti, poi di meno
idea: faccio Gomory cut per ogni comp. frazionarie
comunque c'è il tailing off

Branch-and-bound

integrality gap: $z_{IP} - z_{LP}$ (spesso in %



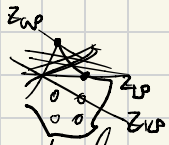

prune by bounding
$LB_K \ge z_{OPT}$

+ piccola $z_{OPT}$, + facilmente rispettato
+ grande $LB_K$

Idea: genero migliori bound con cutting plane ⇒

⇒ BRANCH-AND-CUT:

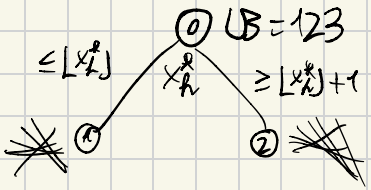
usare cut per avere migliori bound



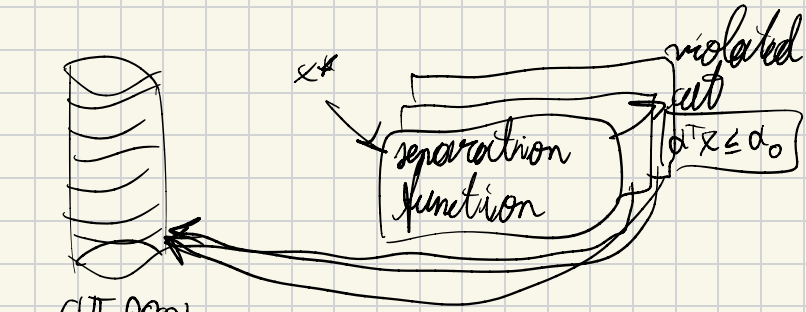

$z_{LB}$ trovato dopo aggiunta tot cut

$\min\{c^T x \mid Ax = b,\ x \geq 0\}$

$LB = \dots 123$




Implementazione (Padberg - Rinaldi)




poi, programma branch controlla se $x^*$ viola cut in nodi non ci si chiede dove sono stati fatti cut ⇒

⇒ si controllano cut in pool

In cut pool si segna anche se cut sono GLOBAL o LOCAL (generati in certo nodo e validi per sottoalbero) ⇒ B-P: vogliamo solo global cuts

1. Identification of the structural properties of the ILP model under study.
2. Translation of the properties identified in terms of *classes of valid inequalities* (*polyhedral analysis*).
3. For each inequality class $C$, definition of effective procedures for the exact/heuristic solution of the following

   **Separation Problem for the $C$ class**: *Given* $\mathbf{x}^*$*, identify (if there exists) an inequality* $\alpha^T\mathbf{x} \leq \alpha_0$ *belonging to class* $C$ *and such that* $\alpha^T\mathbf{x}^* > \alpha_0$.

In practice, for any class $C$ we are interested in identifying several violated inequalities, chosen among those that maximize the degree of violation $\alpha^T\mathbf{x}^* - \alpha_0$. This typically accelerates the overall convergence of the *branch-and-cut* algorithm.

### 6.5.1 An Example: the Index Selection Problem

The main features of the *branch-and-cut* algorithm will be illustrated by means of a problem occurring in database design.

A relational database can be thought as a set of tables (data), plus *query* and update procedures. Typically, the data structure must be queried in real time, and therefore the response time to answer a query is a determining factor in the choice of the method with which to organize the information. The answer to a query involves scanning data, an operation that can be accelerated if the records are kept organized (using some keys) by means of one or more indices.

The response time to a given query is therefore a function of the index used. On the other hand, each index has a fixed cost (in terms of computing time) related to the periodic updating of data, and it occupies memory on its own.

As an example, consider the following numerical case. There are $m = 6$ queries and $n = 5$ potential indices. There is a dummy index, index 0, the use of which actually corresponds to the sequential scanning of data. The following table quantifies the response costs (computing times) to each query, using one of the indices provided; it is assumed that it is not possible to use more than one index for the same query.

ex: min di esecuzione / mese

| query | index 0 | index 1 | index 2 | index 3 | index 4 | index 5 |
|---|---|---|---|---|---|---|
| 1 | 6200 | 1300 | 6200 | 6200 | 6200 | 6200 |
| 2 | 2000 | 900 | 700 | 2000 | 2000 | 2000 |
| 3 | 800 | 800 | 800 | 800 | 800 | 800 |
| 4 | 6700 | 6700 | 6700 | 1700 | 6700 | 2700 |
| 5 | 5000 | 5000 | 5000 | 2200 | 1200 | 4200 |
| 6 | 2000 | 2000 | 2000 | 2000 | 2000 | 750 |

ex: dati di ospedale
↓
ho indici con puntatori a dati per ordinarli secondo vari attributi (nome, età, etc.)
↓
non posso crearne troppi (vanno mantenuti)
↓
bisogna capire quali sono query e indici migliori (euristiche)
↓
si arriva a modello (semplificazione)

The fixed cost and the size in terabyte (TB) of the indices are the following:

| | index 1 | index 2 | index 3 | index 4 | index 5 |
|---|---|---|---|---|---|
| fixed cost | 200 | 1200 | 400 | 2400 | 250 |
| size | 10 | 5 | 10 | 8 | 6 |

The total index storage space available is $D = 19$ TB.

A feasible solution corresponds to any subset of indices having size not greater than $D$ (they are the indices that we actually want to create). The overall cost of the solution is obtained by summing the fixed costs and the costs related to the $m$ queries, according to the following considerations.

Consider, for instance, solution $S = \{1, 5\}$, which occupies only 16 TB, and is thus feasible. Solution $S$ does not create indices 2, 3 and 4, which must then be removed from the previous tables. For each query $i \in \{1, \ldots, m\}$ we have to decide which index $j \in S \cup \{0\}$ to select. For query 1 it will be convenient to select index 1 (1300 being the minimum cost on the row). Index 1 will also be used for query 2 (cost 900), while index 5 will be used for queries 4 (cost 2700), 5 (cost 4200) and 6 (cost 750). Instead, index 0 is used for query 3 (cost 800). The overall cost to answer to the $m$ queries is 10.650. We need to add to this cost the fixed costs (200 and 250) related to the creation/updating of indices 1 and 5. The total cost of solution $S = \{1, 5\}$ is $10.650 + 450 = 11.100$.

The *Index Selection Problem, ISP* consists in identifying a feasible solution with minimum cost. In the numerical example, it is easy to verify that the optimal solution is precisely $S = \{1, 5\}$, with cost 11.100.

---

Suppose we want to design a *branch-and-cut* algorithm for this problem. The first step consists in defining a valid ILP model. To that end, let:

- $m$ = total number of queries;
- $n$ = total number of potential indices;
- $D$ = available memory size for the indices selected;
- $c_j$ = positive fixed cost of index $j \in \{1, \ldots, n\}$;
- $d_j$ = positive size of index $j \in \{1, \ldots, n\}$;
- $\gamma_{ij}$ = positive cost to answer query $i \in \{1, \ldots, m\}$ by means of index $j \in \{1, \ldots, n\}$.

The "natural" variables of the problem are, for all $j \in \{1, \ldots, n\}$:

$$y_j = \begin{cases} 1 & \text{if index } j \text{ is selected} \\ 0 & \text{otherwise.} \end{cases}$$

To express the fact that every query is associated with a selected index, we introduce the following auxiliary variables, defined for all $i \in \{1, \dots, m\}$ and for all $j \in \{0, 1, .., n\}$:

$$x_{ij} = \begin{cases} 1 & \text{if query } i \text{ uses index } j \\ 0 & \text{otherwise.} \end{cases}$$

Note that variables $x_{ij}$ are defined also for $j = 0$ (dummy index).

A possible formulation is then the following.

$$\min \underbrace{\sum_{j=1}^{n} c_j y_j}_{\text{fixed cost}} + \underbrace{\sum_{i=1}^{m} \sum_{j=0}^{n} \gamma_{ij} x_{ij}}_{\text{query cost}} \tag{6.6a}$$

$$\underbrace{\sum_{j=1}^{n} d_j y_j \leq D}_{\text{storage usage}} \tag{6.6b}$$

$$\underbrace{\sum_{j=0}^{n} x_{ij} = 1}_{\text{only one index for each query}} \quad , \; i \in \{1, \dots, m\} \tag{6.6c}$$

$$\underbrace{\sum_{i=1}^{m} x_{ij} \leq m y_j}_{y_j = 0 \;\Rightarrow\; x_{ij} = 0} \; , \; j \in \{1, \dots, n\} \tag{6.6d}$$

$$0 \leq x_{ij} \leq 1 \text{ integer} \; , \; i \in \{1, \dots, m\} \; ; j \in \{0, \dots, n\} \tag{6.6e}$$

$$0 \leq y_j \leq 1 \text{ integer} \; , \; j \in \{0, \dots, n\}. \tag{6.6f}$$

Constraint (6.6d) expresses the logic consistency of variables $x_{ij}$ and $y_j$. In particular, when $y_j = 0$ the constraint imposes that $\sum_{i=1}^{m} x_{ij} = 0$ and hence $x_{ij} = 0$ for all $i \in \{1, \dots, m\}$ (if index $j$ is not selected, then it cannot be used by any query). Note that, due to the effect of factor $m$ in the right-hand side, the constraint becomes inactive when $y_j = 1$ given that, in this case, index $j$ (if convenient) can be used by all the $m$ queries.

We now have an ILP formulation for the problem. If this formulation proves to be sufficiently tight, we can likely solve real-size problems (a few hundred indices and queries). Otherwise, it could be difficult to solve even very small instances, with only 15-20 indices and queries.

The ideal would be that solution $(\mathbf{x}^*, \mathbf{y}^*)$ of the continuous relaxation of the model had very few fractional components, and in any case a cost (6.6a) close to the optimal integer value (i.e., a tight lower bound). However, this hardly ever happens, mainly because of the *modeling mistake* we made when writing constraints (6.6d). Indeed, while condition "$y_j$ integer" is maintained, these constraints act as requested. Relaxing the integrality condition we have, however, that the model, in the attempt of minimizing the costs, will assign to $y_j^*$ the smallest value compatible with constraint (6.6d), i.e., it will

systematically define $y_j^* = \frac{1}{m}\sum_{i=1}^{m} x_{ij}^*$. Therefore, even assuming that the values $x_{ij}^*$ are all integer, we will have $y_j^* = 1$ only if $\sum_{i=1}^{m} x_{ij}^* = m$, that is, only in the (unlikely) case that a selected index $j$ is then used by *all* $m$ queries. In other words, coefficient $m$ of variable $y_j$ in constraint (6.6d) causes values $y_j^*$ to be typically far from 1. On the other hand, the integer solutions *must* have a certain number of variables $y_j = 1$, and hence they will be identified by the *branch-and-bound* algorithm only in the deep nodes of the branching tree, after a considerable number of branching operations. Note that there exist $2^k$ nodes with depth $k$: if the integer solutions can be found on average at depth $k = 15$ or 20, this means having approximately $2^{15} - 2^{20}$ nodes in the branching tree!

A first strengthening of constraints (6.6d) may be easily obtained as follows. Consider, e.g., the column associated with index $j = 1$ in the query cost table, and note that some costs $\gamma_{ij}$ are equal to the corresponding cost $\gamma_{i0}$ on column 0 (this happens because index $j$ is related to a key not related to query $i$). In this case, we can fix $x_{ij} = 0$, given that any optimal solution has no interest in using index $j$ for query $i$, since it can use at the same cost index 0. It is thus valid to set to zero, i.e., to eliminate from the model, all variables $x_{ij}$ such that $\gamma_{ij} \geq \gamma_{i0}$. Therefore, for every index $j \in \{1, \ldots, n\}$ only variables $x_{ij}$ with $i \in I_j$ are active, where

$$I_j := \{i \in \{1, \ldots, m\} : \gamma_{ij} < \gamma_{i0}\}$$

is the set of queries for which it is convenient to use index $j$ rather than index 0.

In the numerical example, the active variables (listed column-wise) are $x_{10}, \ldots, x_{60}$, $x_{11}$, $x_{21}$, $x_{22}$, $x_{43}$, $x_{53}$, $x_{54}$, $x_{45}$, $x_{55}$ and $x_{67}$, as well as variables $y_1, \ldots, y_5$.

In addition to reducing the number of variables of the problem, this reduction allow us to write constraints (6.6d) as

$$\sum_{i \in I_j} x_{ij} \leq |I_j| y_j \quad , \quad j \in \{1, \ldots, n\}. \tag{6.7}$$

These constraints are stronger than constraints (6.6d), since the coefficient of $y_j$ has been decreased from $m$ to $|I_j|$.

Consider now the continuous relaxation of the new model. In our numerical example, the optimal solution $(\mathbf{x}^*, \mathbf{y}^*)$ has the following non-zero components:

problema applicato a piccola istanza, per studiarlo

$$\begin{array}{ll} & x_{20}^* = \frac{6}{10},\ x_{30}^* = 1 \\ y_1^* = \frac{7}{10}, & x_{11}^* = 1,\ x_{21}^* = \frac{4}{10} \\ y_3^* = 1, & x_{43}^* = x_{53}^* = 1 \\ y_5^* = \frac{1}{3}, & x_{65}^* = 1 \end{array}$$

and a (rounded) cost equal to 8.940. As can be seen, there is a significant difference (gap = 2.160) between the lower bound 8.940 and the optimal integer value 11.100. By directly applying the *branch-and-bound* algorithm, a large number of subproblems are generated, and the necessary computational time becomes prohibitive.

Let us then study the specific properties of the ISP problem, trying to identify new classes of valid inequalities (*polyhedral analysis*). To this end, it is useful to carefully analyze the fractional solution of the example. A "fault" of this solution is that index 1 is selected at 70% ($y_1^* = \frac{7}{10}$), but is used at 100% for query 1 ($x_{11}^* = 1$). It seems natural, instead, to impose that a partially selected index is only partially used by the queries. This structural condition, specific to the problem under study, can be expressed by the linear constraint $x_{11} \leq y_1$. We have thus identified a first class $C_1$ of valid inequalities for our problem:

$$\textbf{Class } C_1: \quad x_{ij} \leq y_j \ , \ j \in \{1, \ldots, n\}, \ i \in I_j. \quad \begin{array}{l}\text{oppure:} \\ \forall i \in [1,m], \\ \forall j \in [1,m]\end{array} \qquad (6.8)$$

violati da $x^*, y^*$

Constraints (6.8) are similar to constraints (6.7), since they impose the congruence relation "$y_j = 0 \Rightarrow x_{ij} = 0$". However, in the absence of the integrality constraint, the new constraints may considerably strengthen the initial formulation, as is the case in the numerical example considered.

Note that constraints (6.7) can be obtained from (6.8) by simply summing up the conditions $x_{ij} \leq y_j$ for all $i \in I_j$. Constraints (6.7) are then linearly implied ( la Farkas) by constraints (6.8). We could think of substituting constraints (6.7) with (6.8) in the model. This, however, would create a model with a very large number of constraints, since each constraint (6.7) is replaced by $|I_j|$ different constraints. On the other hand, not all these new constraints are necessary for solving a *specific* numerical instance: in the example, constraint $x_{21} \leq y_1$, even if not imposed, is satisfied by the current solution. So it can be convenient to generate "on the fly" (i.e., during the execution of the algorithm) the constraints from class $C_1$, limiting ourselves to those violated by the current fractional solution. To that end, we have to solve the following:

**Separation Problem for class** $C_1$: *Given* $(\mathbf{x}^*, \mathbf{y}^*)$*, identify a pair (i,j) such that* $x_{ij}^* > y_j^*$ *(if any).*

Given that family $C_1$ contains only $\sum_{j=1}^{n} |I_j| \leq n \cdot m$ inequalities, this problem can be solved with little computational effort, by enumeration.

**Separation Procedure for class** $C_1$;
**begin**
  **for** $j := 1$ **to** $n$ **do**
    **for each** $i \in I_j$ **do** (for each $i:=1$ to $m$ do)
      **if** $x_{ij}^* > y_j^*$ **then**
        "constraint $x_{ij} \leq y_j$ is violated by $(\mathbf{x}^*, \mathbf{y}^*)$"
**end** .

ritornare vincolo

In the numerical example, the procedure identifies two violated constraints, namely $x_{11} \leq y_1$ and $x_{65} \leq y_5$. We add those two cuts to the current model, and re-optimize using the dual simplex algorithm obtaining an optimal solution with cost 9.900 (gap = 1.200) and

implementare ricerca in modello

$x_{11} \in y_1 \begin{cases} y_1 = 0 \\ y_1 = 1 \end{cases}$

no somma, ma vincolo per ogni elemento

poniamo mettere questi vincoli in modello o usarli come funzione di separazione per B&C

with the following non-zero components:

$$\begin{array}{ll} & x^*_{30} = 1,\ x^*_{60} = \frac{3}{4} \\ y^*_1 = 1, & x^*_{11} = x^*_{21} = 1 \\ y^*_3 = \frac{3}{4}, & x^*_{43} = x^*_{53} = \frac{3}{4} \\ y^*_5 = \frac{1}{4}, & x^*_{45} = x^*_{55} = x^*_{65} = \frac{1}{4}. \end{array}$$

By applying the separation algorithm for class $C_1$, no violated cuts are generated, meaning that the current solution satisfies *all* constraints (6.8), even if we explicitly imposed only two of them. At this stage, we can perform the branching operation, or we can study the fractional point in order to search for a new class of valid inequalities.

Note that indices 1 and 3 cannot be chosen at the same time, as they exceed the available memory $D$. We then have condition $y_1 + y_3 < 2$, which can be strengthen into $y_1 + y_3 \leq 1$ given that $y_1$ and $y_3$ must be integer. This latter inequality is violated by the current fractional point, given that $y^*_1 = 1$ and $y^*_3 = \frac{3}{4}$. More generally, we have the following new class of valid constraints:

$$\textbf{Class } C_2: \quad \sum_{j \in S} y_j \leq |S| - 1\ ,\ \forall\, S \subseteq \{1, \ldots, n\}\ ,\ S \neq \emptyset\ :\ \sum_{j \in S} d_j > D. \tag{6.9}$$

Since there exist $2^n - 1$ non-empty subsets $S \subseteq \{1, \ldots, n\}$, class (6.9) can contain a huge number of inequalities. This is certainly a positive feature: the more inequalities, the tighter the resulting formulation (as there is a better chance of identifying a violated cut). Obviously, the separation algorithm for class $C_2$ cannot just enumerate all subsets $S$, verifying that $\sum_{j \in S} d_j > D$ and $\sum_{i \in S} y^*_j > |S| - 1$: the computational time requested would be huge.

A more sophisticated approach consists in formulating the separation problem as an ILP problem of its own, that can be solved, for instance, by means of the *branch-and-bound* algorithm. To that end, for all $j \in \{1, \ldots, n\}$ we can introduce the auxiliary variables

$$z_j = \begin{cases} 1 & \text{if } j \text{ belongs to } S \\ 0 & \text{otherwise,} \end{cases}$$

and reformulate the separation problem as the problem of identifying, if there exists, a vector $\mathbf{z} \in \{0,1\}^n$ such that:

$$w := \underbrace{\sum_{j=1}^{n} z_j}_{|S|} - \underbrace{\sum_{j=1}^{n} y^*_j z_j}_{\sum_{j \in S} y^*_j} < 1 \quad (1 - w \text{ being the constraint violation}) \tag{6.10a}$$

$$\underbrace{\sum_{j=1}^{n} d_j z_j}_{\sum_{j \in S} d_j} \geq D + \varepsilon, \qquad \text{(validity of the constraint)} \tag{6.10b}$$

where $\varepsilon > 0$ is a sufficiently small value (for example, $\varepsilon = 1$ if values $d_1, \ldots, d_n$ and $D$ are all integer). If we want to identify the most violated constraint (6.9), we can interpret (6.10b) as an objective function, and write the following ILP model for the separation problem:

$$\begin{aligned} w^* := \quad & \min \sum_{j=1}^{n} (1 - y_j^*) z_j \\ & \sum_{j=1}^{n} d_j z_j \geq D + \varepsilon \\ & 0 \leq z_j \leq 1 \text{ integer }, \; j \in \{1, \ldots, n\}. \end{aligned}$$

problemi ausiliari devono essere molto + veloci di problema principale

For this problem, known in the literature as the minimization form of the *knapsack problem*, particularly effective solution codes exist. Note that values $y_j^*$ are *known* values between 0 and 1, hence coefficients $1 - y_j^*$ of the objective function are all nonnegative. In addition, simple considerations show that it is possible to reduce the number of variables $z_j$ of the problem by setting $z_j = 1$ if $y_j^* = 1$ (variable $z_j$ has null cost in the objective function), and $z_j = 0$ if $y_j^* = 0$ (every solution with $z_j = 1$ would have a value of the objective function not lower than $1 - y_j^* = 1$, hence it would correspond to a non-violated constraint (6.9)).

By solving this easy (with respect to the original problem) ILP model, we can identify subsets $S$ associated with maximally violated constraints (6.9): if $w^* < 1$, this constraint is indeed violated, and can be added to the current continuous relaxation. Otherwise the current solution $(\mathbf{x}^*, \mathbf{y}^*)$ satisfies *all* potential $2^n - 1$ constraints (6.9).

Another possibility is to heuristically limit the choice of subsets $S$ imposing $|S| \leq k$ for any value $k$ set (for instance, $k = 3$ or 4), according to the following scheme:

**Heuristic separation procedure for class** $C_2$;
  **begin**
    let $J^* := \{j \in \{1, \ldots, n\} \; : \; y_j^* > 0\}$;
    **for each** $S \subseteq J^* \; : \; 1 \leq |S| \leq k$ **do**
      **if** $\sum_{j \in S} d_j > D$ **then** /* valid constraint */
        **if** $\sum_{i \in S} y_j^* > |S| - 1$ **then**
          "constraint $\sum_{j \in S} y_j \leq |S| - 1$ is violated by $(\mathbf{x}^*, \mathbf{y}^*)$"
  **end** .

Note that the algorithm only considers subsets $S$ made by indices $j$ with $y_j^* > 0$ given that, as already observed, constraint (6.9) cannot be violated when $j \in S$ and $y_j^* = 0$.

In the numerical example, by applying the separation algorithm for $k = 2$, we obtain the violated constraint $y_1 + y_3 \leq 1$. By adding this constraint to the current formulation and

re-optimizing, we obtain an optima lfractional solution with cost 10.880 (gap = 220), with the following non-zero components:

$$\begin{aligned} & x^*_{30} = 1 \\ y^*_1 = 1, \quad & x^*_{11} = x^*_{21} = 1 \\ y^*_4 = \tfrac{3}{8}, \quad & x^*_{54} = \tfrac{3}{8} \\ y^*_5 = 1, \quad & x^*_{45} = x^*_{65} = 1, x^*_{55} = \tfrac{5}{8}. \end{aligned}$$

The separation procedure for class $C_1$ does not produce violated cuts, as that for class $C_2$ with $k = 2$. Trying with $k = 3$ we identify, instead, subset $S = \{1, 4, 5\}$, which corresponds to the violated constraint $y_1 + y_4 + y_5 \leq 2$. By adding this constraint and re-optimizing we obtain a solution with cost 11.100 (gap = 0) and non-zero components

$$\begin{aligned} & x^*_{30} = 1 \\ y^*_1 = 1, \quad & x^*_{11} = x^*_{21} = 1 \\ y^*_5 = 1, \quad & x^*_{45} = x^*_{55} = x^*_{65} = 1. \end{aligned}$$

Since no fractional component exists, this solution is optimal and the *branch-and-cut* algorithm stops (in this simple example) without the need to perform any branching operation.

Obviously, it is necessary to verify that the procedure described is effective for bigger problems, and on real data. As a curiosity, we report a computational comparison between the *branch-and-bound* technique (without constraints of the classes $C_1$ and $C_2$) and the *branch-and-cut* technique. The results are average values on 10 real life problems; the times are expressed in CPU seconds and refers to a (very old!) computer SUN Sparc-2.

| | | Branch-and-Bound | | Branch-and-Cut | |
|---|---|---|---|---|---|
| $m$ | $n$ | node | time | nodes | time |
| 250 | 150 | $> 10.000$ | $> 2310.3$ | 1 | 1.3 |
| 375 | 225 | $> 10.000$ | $> 3030.2$ | 17 | 31.2 |
| 500 | 300 | $> 10.000$ | $> 4397.0$ | 6 | 32.3 |

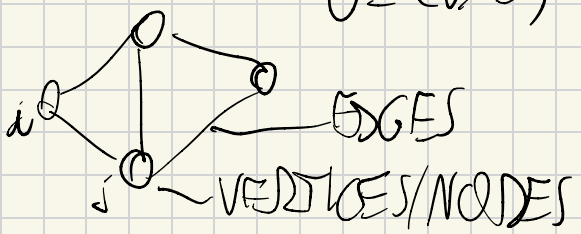
The *branch-and-bound* algorithm could not solve to optimality any of the 30 instances considered, having exceeded the maximum size of the queue of nodes (10.000 nodes). The same instances were easily solved, in few seconds, using the *branch-and-cut* technique. This proves the usefulness, for these problems, of considering the generation of cuts during the algorithm: the computational time needed to process each node increases, but the overall number of nodes is drastically reduced (from more than 10.000 nodes to less than 10, on average).

21/11

# GRAPH THEORY

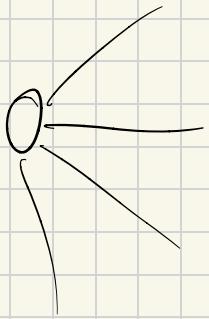
si possono definire problemi su grafi
Possono essere UNDIRECTED e DIRECTED

## GRAPH




$G = (V, E) \Rightarrow V = \{v_1, \dots, v_n\}$, $n := |V|$
$E \subseteq V \times V$, edge $\{i,j\}$, $(i,j)$, $ij$
$\{i,i\}$: LOOP
(di solito no loop e archi multipli: SIMPLE GRAPH)

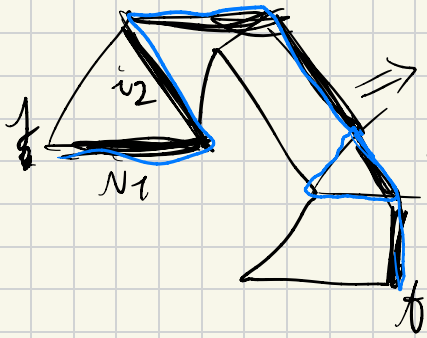
EXTREME/TERMINAL nodes: $i, j$ in $\{i,j\}$
$\{i,j\}$ TOUCHES $i, j$
$i$ ADJACENT to $j$
$\{i,j\}$ INCIDENT in $j$



$\Rightarrow$ insieme di archi incidenti in $j$
STAR di $j$: $\delta(j) = \{(a,b) : j = a \text{ o } j = b\}$
DEGREE di $j$: $d_j = |\delta(j)|$
se $\delta(j) = 0$, $j$ ISOLATED

dato $G=(V,E) \Rightarrow G'=(V',E')$ SUBGRAPH di $G$:
$V' \subseteq V$, $E' \subseteq E$

$V=V'$: PARTIAL GRAPH (solo togliere archi)



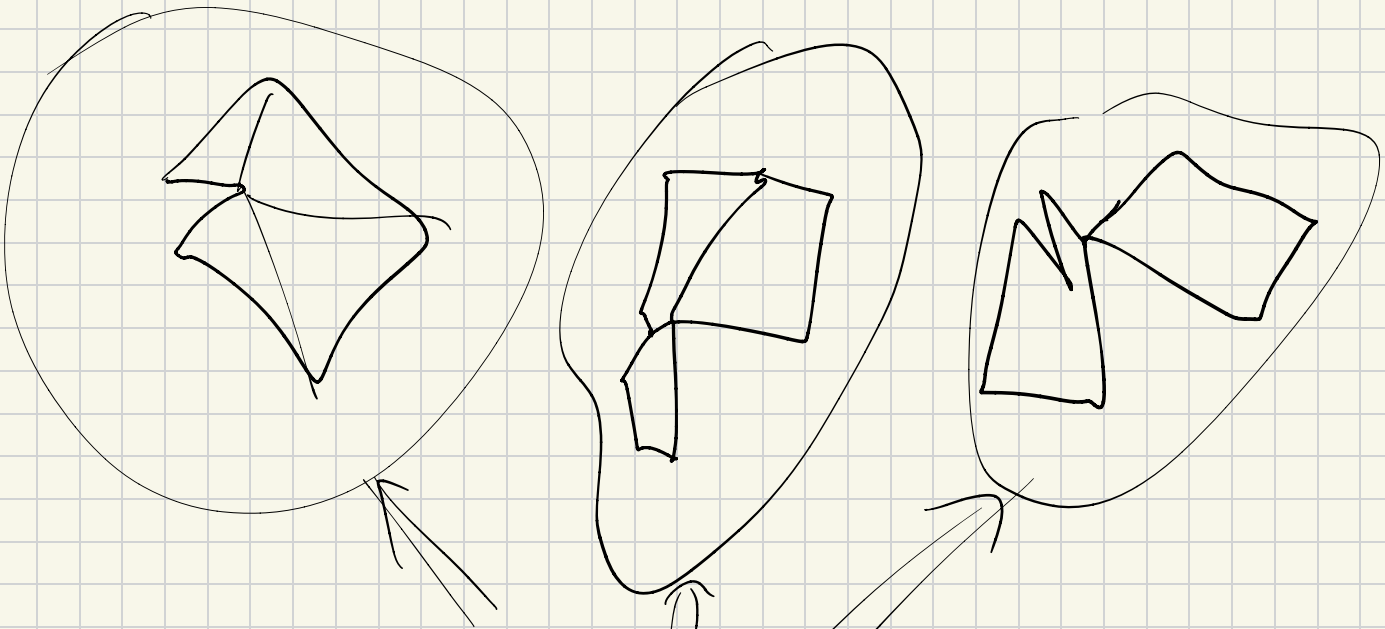

$\Rightarrow$ WALK / PATH
percorso senza loop: SIMPLE PATH

CONNECTED GRAPH:
si può trovare percorso per ogni coppia di vertici

dato insieme $S$, CUT indotto da $S$:
$\delta(S) = \{[i,j] \mid i \in S, j \notin S\}$
TEO: grafo connesso se non ha cut vuoti

$G=(V,E)$ connesso $\Rightarrow$ cerchiamo grafo parziale connesso con meno archi possibili
SPANNING TREE $G'$:

1- $G'$ connesso
2- $G'$ senza cicli (percorsi chiusi)
3- $|E'| = |V| - 1$



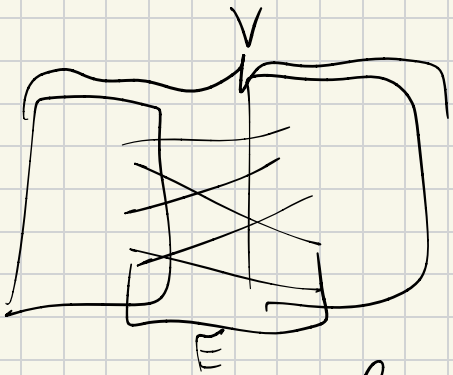
CONNECTED COMPONENT

posso trovare spanning tree in ogni componente

insieme di alberi: FOREST

BIPARTITE GRAPH:

grafo tale che



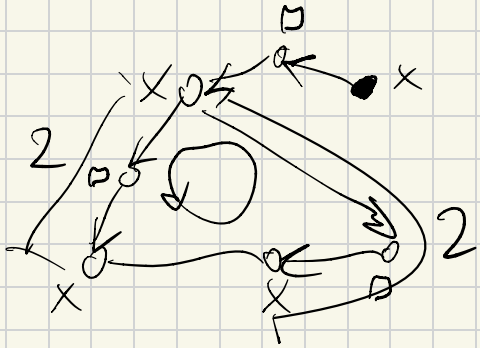

COLOR CLASSES

$G = (V_1 \cup V_2, E)$

$E = \delta(V_1) = \delta(V_2)$

non può avere cicli con numero dispari di nodi

Algoritmo di backtracking per dimostrare proprietà di bipartite graphs



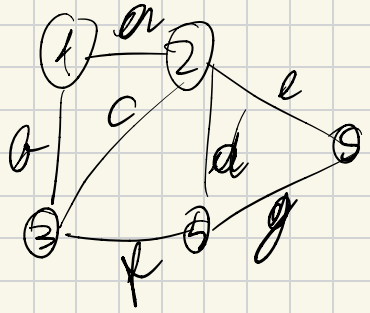

PLANARITY:

grafo PLANARE se posso disegnarlo senza incrociare archi

22/11

Per rappresentare grafo, usare matrice

Dato und. grafo $G=(V,E)$

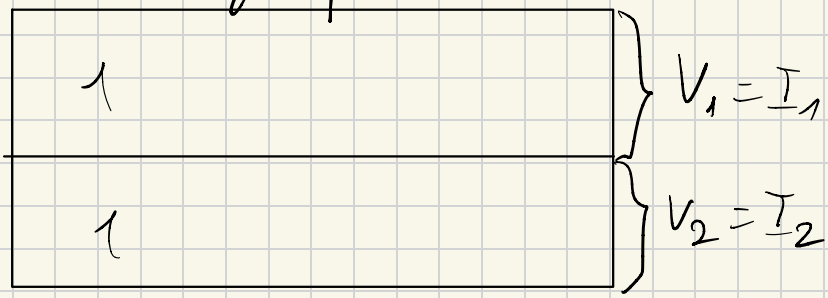



$\Rightarrow$

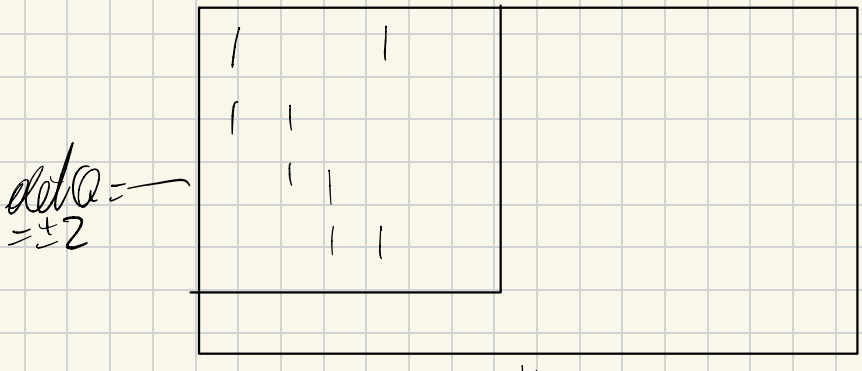
| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 5 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

NODE-EDGE INCIDENCE MATRIX

Questa matrice è TUM? $\Leftrightarrow$ grafo bipartito



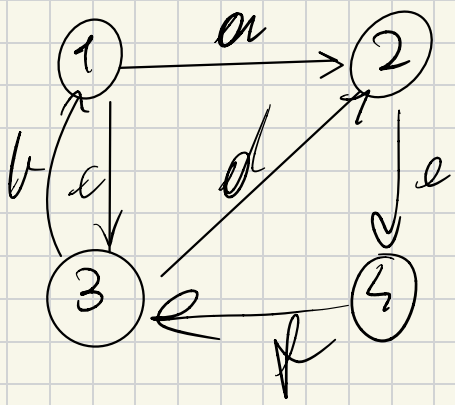

G non bipartite $\Rightarrow \exists$ odd ciclo




$\Downarrow$

non TUM

Diretto $G=(V,E)$



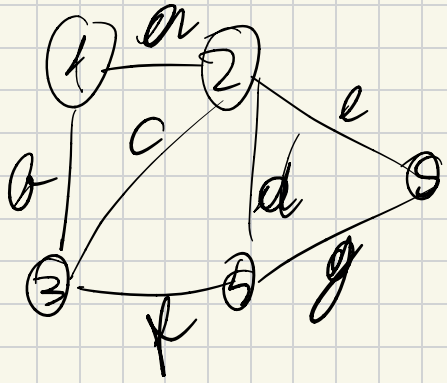

NODE-ARC INCIDENCE MATRIX

| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| 1 | +1 | -1 | +1 | 0 | 0 | 0 |
| 2 | -1 | 0 | 0 | -1 | +1 | 0 |
| 3 | 0 | +1 | -1 | +1 | 0 | -1 |
| 4 | 0 | 0 | 0 | 0 | -1 | +1 |

sempre TUM (proprietà $I_1=V$, $I_2=\emptyset$)

Problemi con salvataggio grafi $\Rightarrow$

$\Rightarrow$ matrici non sono efficienti $\Rightarrow$ piuttosto, array




$n=5$ $(m=7)$

$\forall$ lato: salvo:

| | 1 | 2 | 3 | 4 | | | | |
|---|---|---|---|---|---|---|---|---|
| FROM | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 |
| TO | 2 | 3 | 1 | 3 | 4 | 1 | 2 | 4 |

(FROM è ordinato)

ogni lato salvato 2 volte

2 blocchi per ogni nodo

| | 1 | 2 | 3 | 4 | | | | |
|---|---|---|---|---|---|---|---|---|
| FROM | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 |
| TO | 2 | 3 | 1 | 3 | 4 | 1 | 2 | 4 |

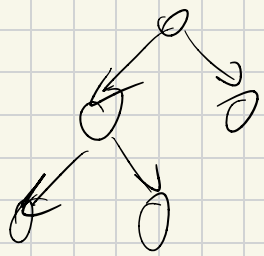
puntatori $\Rightarrow$ FIRST[1] FIRST[2] FIRST[3] ... metto anche puntatore dummy FIRST[6]

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| FIRST | 1 | 3 | 7 | 10 | 13 | 15 |

così, array FROM non serve

Quando visitiamo nodo di grado vogliamo sapere $\delta$(nodo) ⇒ così lo troviamo + efficientemente

Implementazioni particolari:



ogni nodo ha 1 predecessore

array pred 

include valore dummy per radice

(oppure pred[root] = root)

(NO comp. complexity per ora)

(oppure d(SHORTEST))

MINIMUM SPANNING TREE problem

dato $G=(V,E)$ undirected e connesso, $c: E \to \mathbb{R}$,

$c(e)$ = costo di dato e

vogliamo spanning tree con costo minimo

dato albero $G_T=(V,T)$, $T \subseteq E$

$cost(T) = \sum_{e \in T} c(e)$

definiamolo con ILP (anche se non serve)

$$x_e = \begin{cases} 1 & \text{se } e \in T \\ 0 & \text{altrimenti} \end{cases} \quad \forall e \in E$$

$$\min \underbrace{\sum_{e \in E} c_e x_e}_{\Rightarrow e \in T} \quad \sum c_e$$

$$\sum_{e \in E} x_e = |V| - 1 \Rightarrow |T| = |V| - 1$$

$$0 \le x_e \le 1 \text{ integer } \forall e \in T$$

$$\sum_{e \in E(S)} x_e \le |S| - 1 \quad \forall S \subset V, |S| \ge 3 \Rightarrow$$

SUBTOUR ELIMINATION CONSTRAINT

ne abbiamo $\simeq 2^n$

non ci serviranno perché non usiamo questo modello :-)

# lati in S

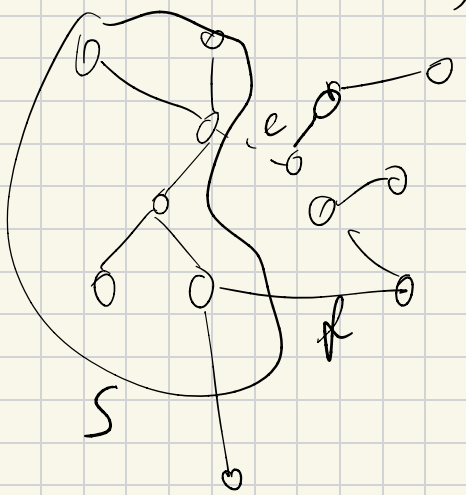
$E(S) = \{[i,j) \in E \mid i,j \in S\}$

Hp: grafo connesso; $c_e \neq c_f \; \forall e \neq f \in E$ (costi tutti diversi)

TEO: sia $G_T = (V, T^*)$ min spanning tree;

$\forall e \in E, e \in T^* \Leftrightarrow \exists S \subset V \mid e = \text{argmin}_{e \in E} \{c_f \mid f \in \delta(S)\}$

Dim: (dimostriamo $\Leftarrow$)




controesempio:

e minimo non in tree $\Rightarrow$

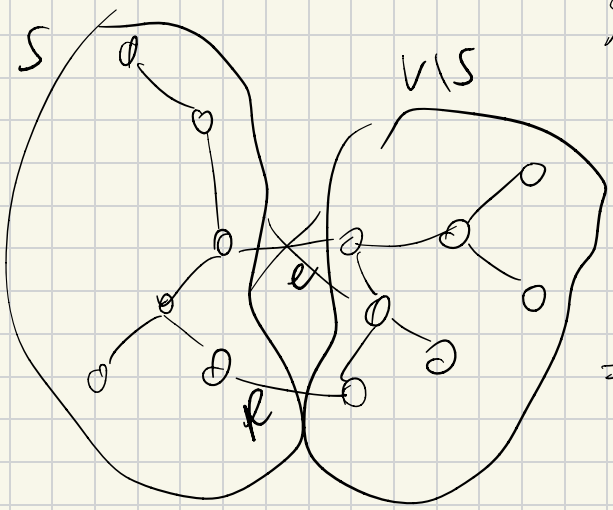
$\Rightarrow T^* \setminus \{f\} \cup \{e\} \Rightarrow$

$\Rightarrow$ albero albero, con

$\text{cost}(T^*) - c_f + c_e < \text{cost}(T^*) \Rightarrow$

$\Rightarrow$ contraddizione

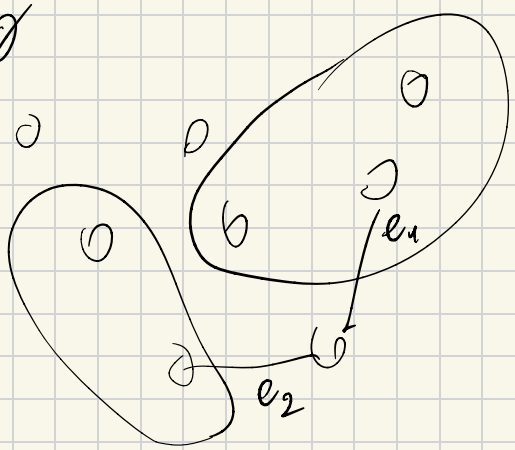
(dimostriamo $\Rightarrow$)




togliamo e da albero
supponiamo e non sia
minimo $\Rightarrow$ piuttosto $f$
$c_f < c_e \Rightarrow$
$\Rightarrow cost(T^*) - c_e + c_f < cost(T^*)$

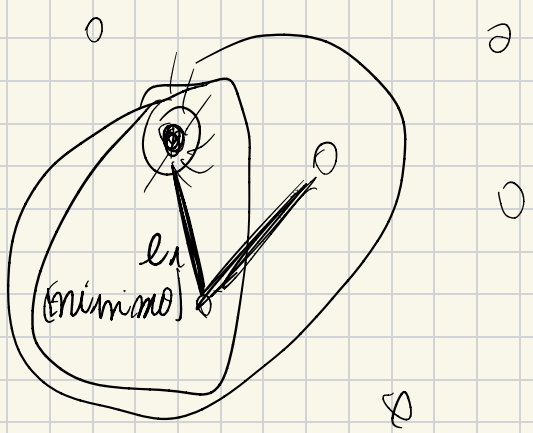
albero migliore
impossibile

Algoritmo base:
$T^* = \emptyset$




$\Rightarrow T^* = \emptyset \cup \{e_1\}$
poi $\cup \{e_2\}$
prendo man mano
S non toccati da $T^*$

NESTED FAMILY $\Rightarrow$ PRIM-DIJKSTRA'S ALG.



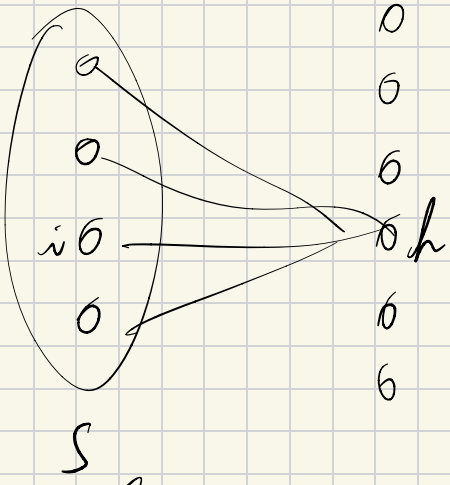

$T^* = \emptyset$
inizio con un nodo
$T^* = \{e_1\}$
scelgo insieme di nodi locali
da $T^*$, poi scelgo lato
minimo che esce

Worst case complessità temp. per questo alg. dipende da $m$ e $n$

Implementazione base: $O(nm)$

Grafo (quasi) completo: $m \approx \frac{n(n-1)}{2} \approx n^2 \Rightarrow O(n^3)$

Altra impl.




calcolo prima:

$L_j = \min\{c_{ih} \mid i \in S\} \quad \forall h \notin S$

prendo $\min\{L_j \mid j \in V \setminus S\}$

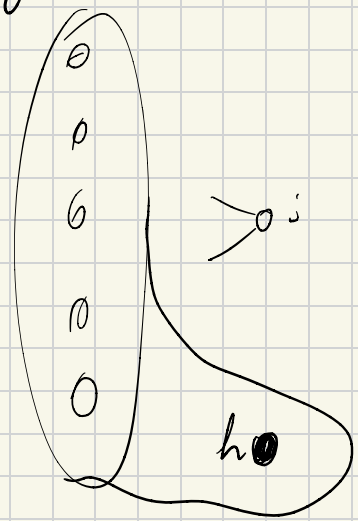
$\Downarrow$

$O(n)$: guardo solo $L_j$

inizializzo $L_h$ all'inizio:

$S = \{1\} \Rightarrow L_j = c_{1j} \quad \forall j \neq 1 \quad \Rightarrow O(n)$

aggiorno:




$L_j = \min\{L_j, \underline{c_{jh}}\}$

$\Downarrow$

unica nuova possibilità

così, $O(n)$

$O(n)$ iniziale

trovare min costo $e \in \delta(S)$: $O(n)$

estendere $S$: $O(1)$

aggiorno: $O(n)$

$\Big\}$ $n-1$ volte $\Rightarrow O(n^2)$

27/11

Prim-Dijkstra: complessità dipende da grafo => se completo: C: matrice con costi => dim. $O(n^2)$ per leggere dati

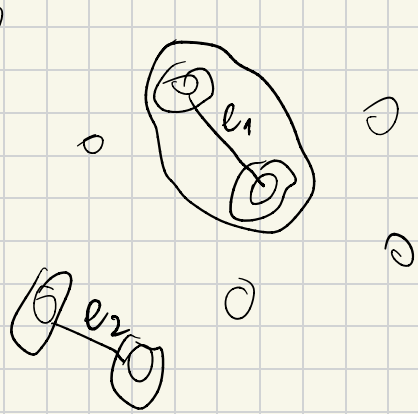
Grafo planare: pochi lati, $|E| \approx 3|V|$ => per lati che non ci sono, $c[i,j] = +\infty$ => per GRAFO SPARSE, P-D non appropriato => disponibile altra implementazione $O(|E| \log |V|)$ oppure altro algoritmo: KRUSKAL'S ALGORITHM

KRUSKAL

Si ordinano lati per costo crescente

Supponiamo $c_{e_1} < \dots < c_{e_m}$ ($m := |E|$, $n := |V|$)

$T^* = \emptyset$




n componenti connesse

$C_1 = \{1\}$, $C_2 = \{2\}$, ...

prendo $c_{e_1}$ (minimo) => sarà in albero per teorema

fondo comp.

prendo $e_2$ => esiste cut dove $e_2$ è ottimo => entra in $T^*$

Ogni it.: prendo sottoinsieme di lati e corrispondente comp.:

- se $e_k$ ha vertice in comp. e altro in altra comp. posso prenderlo e fondere comp.
- se $e_k$ è tutto in comp., creerei ciclo => non posso prenderlo

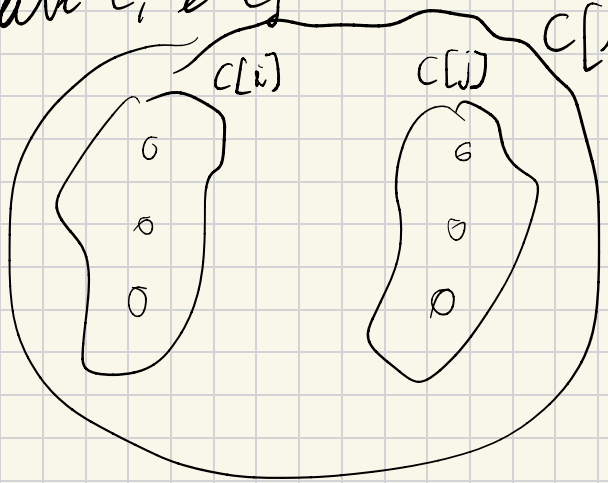
Fatica all'inizio per ordinare

complessità:

- init: $O(n)$
- sort: $O(m \log m) \Rightarrow O(m \log n^2) \Rightarrow O(m \log n)$
- ciclo: $O(n)$ it., merge al massimo $O(n) \Rightarrow O(n^2) \Rightarrow$
  $\Rightarrow$ bisogna trovare buon alg. per fare merge $\Rightarrow$

Alg. per merge in $O(n)$ volte: UNION-FIND methods

dato $C_i$ e $C_j$




```
NEW = C[i], OLD = C[j]
for k = 1 to n do
  if C[k] = OLD then
    C[k] = NEW
```

Kruskal prototipo per alg. GREEDY:

dato prob. di ott. generico

si fa sempre scelta che appare ottima al momento

funziona in certi casi, come MST [Kruskal]

MATROID: se si ha prob. ott. con, posso usare greedy

Input:

$G = (V, A)$ diretto, $c: A \to \mathbb{R}$

source node $s$, terminal node $t$

vogliamo trovare percorso di costo minimo:

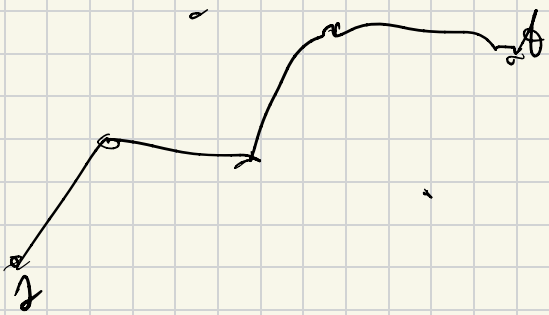
SHORTEST PATH

Tante applicazioni

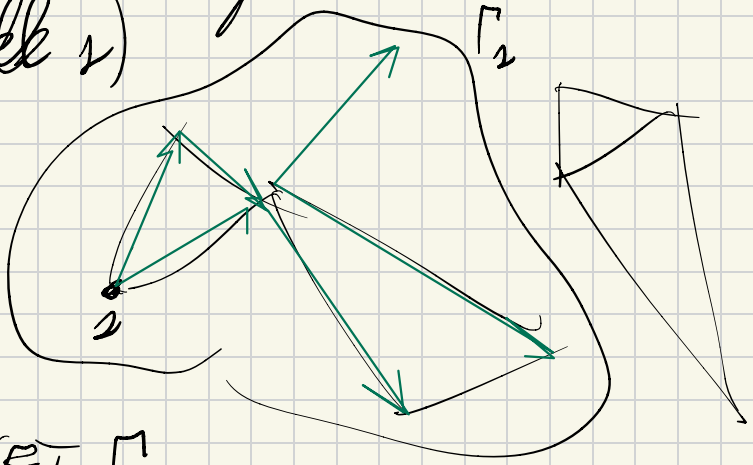
28/11

$G=(V,A)$, $s,t \in V$. vogliamo percorso da $s$ a $t$ $\Rightarrow$
$\Rightarrow$ SHORTEST PATH

Prima determiniamo se possiamo raggiungere $t$ da $s$ $\Rightarrow$ trovare nodi reachable da $s$

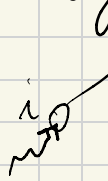



Alg. per reachability: LABELING algorithm
label su $i$ è reachable($i$)
se $r$ ha label, tutti nodi adiacenti sono reachable
NO relabeling
$\Rightarrow$ trova REACHABILITY SET $\Gamma_s$




Algoritmo errato: se mi fermo $i$ perché $\delta^+(S)=\emptyset$
$(\delta^+(S) := \{(i,j) \in A \mid i \in S, j \notin S\}$: archi che ESCONO da $S$)

Implementazione $O(n+m)$ (migliore possibile)
Data structures:
- $Q$ = coda di "nodi attivi": nodi con label ma non espansi
- $pred(j) = \begin{cases} -1 & \text{se nodo } j \text{ non è raggiunto ancora} \\ i \geq 1 & \text{altrimenti} \end{cases}$



Algoritmo:

```
pred[i] = -1 ∀ i ∈ [1, n];        O(n)
pred[s] = s;
Q = {s};
while (Q ≠ ∅) do:
    sia h ∈ Q a piacere;          } max. 1 per nodo
    Q = Q \ {h};                  }
    foreach (h, i) ∈ δ+(h) do:    } max. 1 per arco
        if pred[i] = -1 then      }
            pred[i] = h;          }
            Q = Q ∪ {i};          }
```

L'esecuzione cambia con coda FIFO o LIFO

- FIFO: prima percorsi lunghi 1, poi 2, etc. ⇒
  ⇒ cerca meno archi
- LIFO: depth-first

Entrambi corretti, pred diverso alla fine

SHORTEST PATHS: ILP modello

dati $s \neq t \in V$, vogliamo percorso

non vogliamo percorso con cicli ⇒ percorso semplice

$$x_{ij} = \begin{cases} 1 & \text{se prendo } (i,j) \\ 0 & \text{else} \end{cases} \quad \forall (i,j) \in A$$

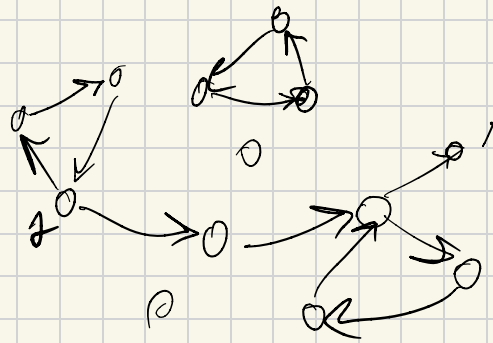
$$\min \sum_{(i,j)\in A} c_{ij} x_{ij}$$

$$\sum_{(i,j)\in\delta^+(h)} x_{ij} - \sum_{(j,i)\in\delta^-(h)} x_{ij} = \begin{cases} 0 & \forall h \in V \setminus \{1, t\} \\ +1 & h = 1 \\ -1 & h = t \end{cases}$$

(# archi uscenti − # archi entranti)

$0 \le x_{ij} \le 1$ intero $\forall [i,j] \in A$

Potrei avere



⇒ dobbiamo escludere cicli

cycle breaking constraints: SEC

$$\sum_{(i,j)\in A(S)} x_{ij} \le |S| - 1 \quad \forall S \subseteq V \mid |S| \ge 2$$

$A(S) = \{(i,j) \in A \mid i,j \in S\}$

SEC: # exp di vincoli

Hp: no negative cost cycles ⇒ in sol. ottima, non avrò mai cicli ⇒ in questo caso posso escludere SEC ⇒

⇒ d.p. $c_{ij} \ge 0 \ \forall i,j$

Senza SEC, non serve integralità

$$A = \left[\begin{array}{c|c} & \\ & \\ & \\ & \\ & \\ & \end{array}\right] \begin{matrix} \leftarrow a \\ \\ \\ \\ \\ \leftarrow b \end{matrix} \qquad b = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \leftarrow 1 \\ \\ \\ \leftarrow t \end{matrix}$$

$x_{ab}$

A: node-arc incidence di grado ⇒

⇒ sempre TUM

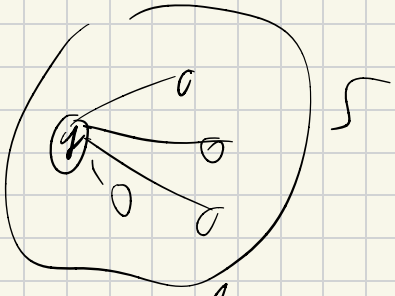
matrice I per vincoli $x_{ij} \le 1$ ⇒ per added TUM non ci serve

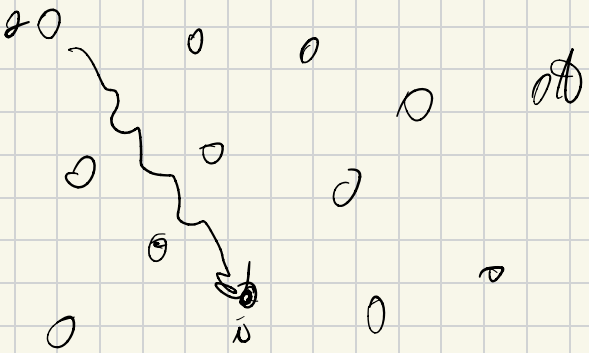
# DIJKSTRA'S ALGORITHM

idea simile a P-D

Hp: $c_{ij} \geq 0 \; \forall [i,j] \geq 0 \; \forall (i,j) \in A$

$\Rightarrow$ in ogni st. calcolo



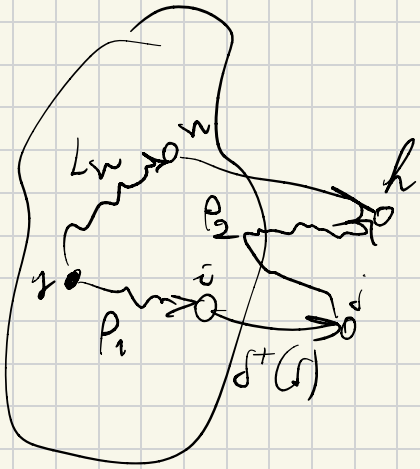




Teo: poniamo di sapere $L_i \; \forall i \in S$; $L_i$ = percorso + corto $s \to i$

calcoliamo $(v,h) := \operatorname{argmin}\{L_i + c_{ij} \mid (i,j) \in \delta^+(S)\}$

Se $c_{ij} \geq 0 \; \forall (i,j)$, allora $L_v + c_{vh}$ è costo di percorso + corto $s \to h$

Dim: prendiamo qualunque altro percorso $s \to h \Rightarrow$ ad una certa esce da $S \Rightarrow$ può essere scomposto: $P = P_1 \cup \{[i,j]\} \cup P_2$

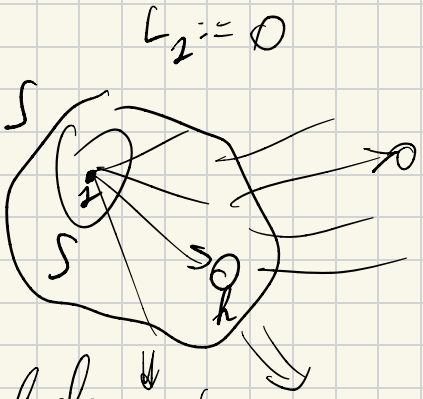



$\text{costo}(P) = \underbrace{\text{costo}(P_1)}_{\geq L_i} + c_{ij} + \underbrace{\text{costo}(P_2)}_{\geq 0}$

$\geq L_i + c_{ij} \geq L_v + c_{vh}$ ■

$\Downarrow$

$(i,j)$ è stato considerato in calcolo $(v,h) \Rightarrow$ non è minimo




fino a raggiungere $\emptyset$

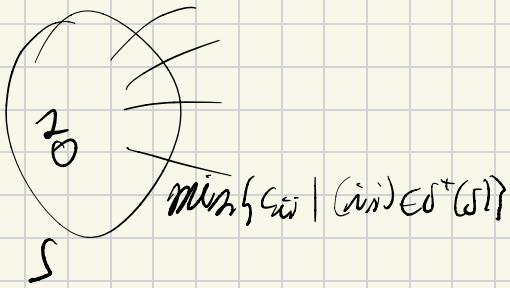
calcolo $c_{ij} + L_i$ $L_h + c_{vh} = L_v$

le uso 1-to-all shortest path: $O(n^3)$

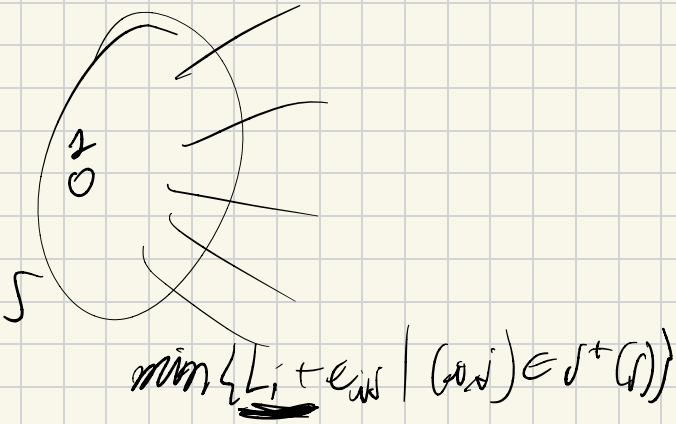
Posso adattarlo e ottenere $O(n^2)$:

confronto MST alg. con shortest path alg.:

MST




SP




prendo P-D e cambio $c_{ij}$ con $L_i + c_{ij} \Rightarrow$ ottengo $O(n^2)$