NUMERO: 706 DATA: 07/10/2013

A P P U N T I

STUDENTE: Orefice

MATERIA: Optimisation Methods and Algoritms
Prof. Tadei

OPTIMIZATION METHODS AND ALGORITHMS

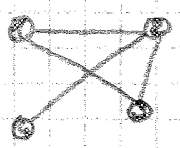
- LIBRI di TESTO. 1) "FONDAMENTI di OTTIMIZZAZIONE"
  R. Tadei - F. Della Croce
  2) "ELEMENTI di RICERCA OPERATIVA"
  R. Tadei - F. Della Croce
  3) "ESERCIZI SVOLTI di RICERCA OPERATIVA"

- ESAME. Esame Scritto che contiene
  * TEORIA (Descrivi un Metodo, Descrivi la Filosofia di un Algoritmo, ...)
  * ESERCIZI (Sapere come funziona un algoritmo ⇒ 1-2 Iterazioni basiche)
  * OPTIMIZATION SOFTWARE

Concerning the VARIABLES, we can have

* x_{ij} INTEGER VARIABLES, $x_{ij} \in \mathbb{Z}^+$
* $x_{ij} \in \{0,1\}$ BOOLEAN
* x_{ij} NON-NEGATIVE CONTINUOUS

SHORTEST PATH MODEL



$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} x_{ji} = \begin{cases} 1 & \text{if } i = \text{Origin Node} \\ -1 & \text{if } i = \text{Dest. Node} \\ 0 & \text{otherwise} \end{cases}$$

* $c_{34} = c_{43} = \infty$ ($c_{ii} = 0$) Nel caso discreto!

$x_{ij} \in \{1,0\}$ $\quad i = 1, \dots, n$ $\quad j = 1, \dots, n$

↳ Possiamo RILASSARE questa condizione e considerare $x_{ij} \geq 0$ così facendo è più semplice risolvere problema. Inoltre, a causa della peculiare struttura particolare delle condizioni imposte, non SICURO che, pur rilassando questa condizione di continuità, la soluzione ottimale sarà $x_{ij} \in \{0,1\}$!

• Proviamo a generalizzare e risolvere lo Shortest Path tra un nodo e TUTTI i restanti! Fino ad ora, infatti, lo abbiamo considerato per una singola coppia!

SHORTEST PATH From node i To any other Node in the Graph:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} x_{ji} = \begin{cases} n-1 & \text{, if } i \text{ is Origin Node} \\ -1 & \text{, all remaining nodes} \end{cases}$$

$$x_{ij} \geq 0$$

K is the FEASIBLE SET

⇒ Feasible Solution = $\begin{bmatrix} x_A \\ x_B \end{bmatrix}$ which satisfy the constraints ⇒ Each point in K

We can draw the PROFIT FUNCTION: $20x_A + 30x_B = €$ PARAMETRO

for example $\begin{cases} x_A = 0, \ x_B = 60 \\ x_B = 0, \ x_A = 90 \end{cases}$ ⇒ $20x_A + 30x_B = 1800$ ← esempio di t

Aumentando t, la Profit line sale (parallela?) finché non risulta tangente a K, in un VERTICE = ossia se K NON è Vuoto!

Theorem: If the Problem is solvable, at least 1 Basic Optimal Solution will be on a vertex (or a corner) of the Feasible Set

Partiamo da un insieme infinito, ~~possibile considerando ogni~~ ~~accettabile~~ ~~determinare~~ ~~punto~~ ~~che~~ ~~caratterizza~~ ~~la~~ ~~regione~~ ~~ammissibile~~ è limitato la nostra ricerca nell'insieme dei vertici ⇒ Importantissimo!! FINITO

- OPTIMAL SOLUTION: è a Feasible Solution (⇒ ∈ in K) che massimizza il mio problema, the Profit Function
- Per rispondere alla nostra richiesta (Max la Profit Function), possiamo andare a disegnare la Profit Function stessa
- Abbiamo disegnato la prima retta isoprofitto considerando, come esempio, t = 1800. Vediamo, possiamo fare meglio, e pertanto aumentiamo t così muoviamo verso l'alto la retta isoprofitto
- Otteniamo così Rette PARALLELE isoprofitto, fino a che troviamo l'ultimo punto accettabile = OPTIMAL SOLUTION, NON possiamo fare meglio.

⇒ ricaviamo $x_A^* = 60$, infatti quel punto si trova sul constraint $x_A = 60$

e, visto il Manpower Constraint: $x_A^* + 2x_B^* = 120$

⇒ $x_B^* = 30$

Tuttavia è possibile che anche questo insieme sia finito, pertanto è impensabile usare un Metodo "Brute" in cui ci mettiamo ad analizzare ogni singolo punto.

⇒ Dobbiamo trovare un Metodo più intelligente: SIMPLEX METHOD

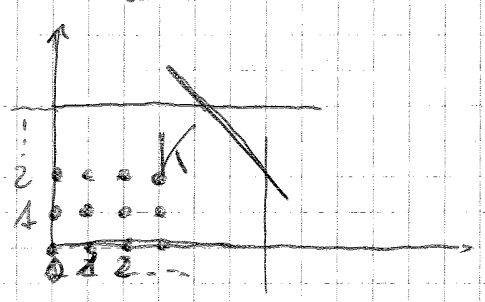
Metodo intelligente per risolvere Problemi Lineari Continui

Ricordiamo che CONT: $x_j \geq 0$ (Basta che sia Non-Negativa)

INT: $x_j \in \mathbb{Z}^+$

Inoltre un Problema CONTINUO è PIÙ SEMPLICE di uno INTERO, infatti:

* Product Problem:



c.: Caso Intero

Nel caso continuo, l'idea era quella di trovare retta ingrandita e muoverci verso l'alto

È possibile applicare la stessa idea al caso INTERO?

In questo caso non abbiamo più un problema convesso e, il fatto che i punti siano distanziati fra loro, rende poco chiaro il concetto di "ultimo punto di tangenza"

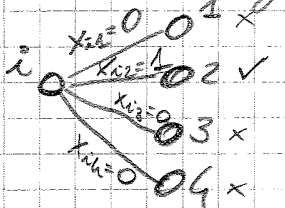
⇒ NO! ⇒ È più complicato del Problema Continuo.

Passiamo ora ad un nuovo Problema:

VINCOLI:

Abbiamo detto che ogni Acquirente può essere visitato 1 sola volta
⇒ 1 edge che entra e 1 che esce

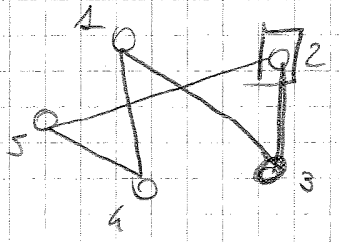
⇒ • $\sum_{j=1}^{n} x_{ij} = 1$ $\forall i$, cioè 1 solo edge uscente.



• $\sum_{j=1}^{n} x_{ji} = 1$ $\forall i$, cioè 1 solo edge entrante

NB. intersezioni pedici, perché sono nodi entranti

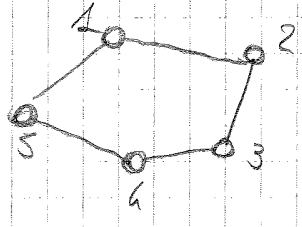
• A questo va aggiunto un VINCOLO HAMILTONIANO. Abbiamo garantito un Tour Completo, ogni nodo deve essere toccato una volta sola; il Venditore deve raggiungere Tutti i nodi e tornare a casa, non ne può Scalare nessuno:



Questo è un esempio di Hamiltonian Tour partendo da 2
NON è l'unico Possibile!

Infatti, se abbiamo n nodi, avremo n! Tour ~~Hamiltoniani~~
infatti, ogni Permutazione degli n acquirenti corrisponde ad un Tour. MA solo $(n-1)!$ sono Hamiltoniani, in quanto non interessa qual'è il punto di partenza.

*



Questo è un Ham-Tour, ~~MA~~ è uguale lo stesso sia che parta da 1 o da 2 o da 3 o da 4 o da 5

16/10/12

- SOMMARIO:

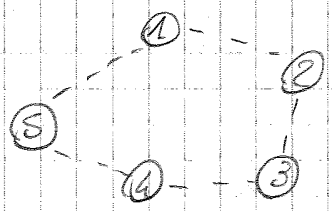
  → TSP ∈ Insieme dei Problemi NP-COMPLETI

Vediamo adesso qualche Metodo Euristico per risolvere TSP, esso può essere risolto sia in modo ESATTO che in modo EURISTICO in base alla dimensione del Problema: ovviamente, se troppo oneroso, conviene usare un Metodo Euristico

L'altra volta ne abbiamo visto uno: NEAREST NEIGHBOUR, il "Vicino Più Vicino"!:

Di i andiamo verso k perché $C_{ik} < C_{iK}, C_{ij}$ e iteriamo il procedimento, fino a completare il Tour. Questa, però, NON è una SOL. OTTIMALE!

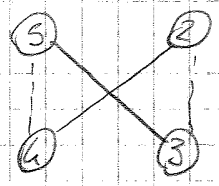
Si può MIGLIORARE.



Consideriamo questa semplice Soluzione ottenuta euristicamente. È un Hamilt. Tour

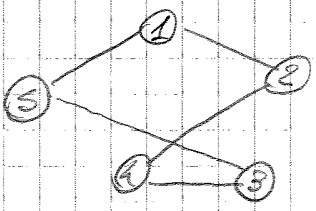
Consideriamo un Paio di Archi:

ad esempio



e proviamo a scambiarli

⇒



otteniamo questa Soluzione che è ancora un Ham. Tour!

Adesso confrontiamo questa nuova soluzione con la precedente: se MIGLIORE la Preferiamo all'Altra!

Questo gioco di scambio tra archi può essere fatto A COPPIE di ARCHI e questo procedimento prende il nome di LOCAL SEARCH HEURISTICS

• DIET PROBLEM

Immaginiamo di fare la spesa e ciò che facciamo è provare a MINIMIZZARE il costo totale della spesa. Se non compriamo niente, spendiamo 0 €, ma poi moriremo, infatti esistono dei nutrimenti (Vitamine, Proteine, ...) che dobbiamo assumere ogni giorno:

$b = \{b_1, b_2, \dots, b_m\}$ Quantità minima di Nutrimento necessario al giorno

$c = \{c_1, c_2, \dots, c_n\}$ Costo (Unitario) dei Cibi

$A = \{a_{ij}\}$ dove ogni a_{ij} = Quantità del Nutrimento i contenuto in una unità del cibo j

Riassumo Modello:

> VARIABILI: Possono essere sia intere che continue, perché uno può comprare pure 1 Kg e 200 gr di carne o simili. Pertanto, visto che usando VARIABILI CONTINUE il PROBLEMA è PIÙ SEMPLICE, ogni volta che ne abbiamo la possibilità le Preferiamo $\Rightarrow$ $x_i \geq 0$

> OBJECTIVE FUNCTION: $\min \sum_{j=1}^{n} c_j x_j$ MINIMIZ. COSTO SPESA

> CONSTRAINTS: Vogliamo soddisfare la Quantità Minima necessaria di ogni singolo nutriente da assumere ogni giorno

$$\Rightarrow \begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \geq b_m \end{cases}$$

Ognuna di queste somme mi dà la Quantità totale del Nutrimento i nella nostra spesa

14/10/22

SUMMARY:

Nell'ultima lezione ci siamo lasciati con 2 questioni relative al TRANSPORT PROBLEM.

a) $x_{ij} \geq 0$ Condizione, NA se a_i e $b_j \in \mathbb{Z}^+ \Rightarrow x_{ij}^* \in \mathbb{Z}^+$

b) $\begin{cases} \sum_{j=1}^{m} x_{ij} \leq a_i \quad \forall i \\ \sum_{i=1}^{n} x_{ij} \geq b_j \quad \forall j \end{cases}$ Quand'è che posso sostituire le disuguaglianze ($\leq$) con uguaglianze?

Cerchiamo di fare un esempio numerico per capire: Consideriamo proprio il problema visto ad esercitazione:

c_{ij}	MI	TO	BO	a_i (Disponibilità)
BS	2	3	2	20
NO	4	2	1	10
VE	3	1	2	20
b_j (Richieste)	10	10	30	50 = Somma Dispon e Somma Richieste

(SHOPS / DEPOSITI)

NB: Quando $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$ il SISTEMA si dice BILANCIATO!

Consideriamo anche la Matrice dei Flussi:

x_{ij}	MI	TO	BO	
BS	10	10	-	20
NO	-	-	10	10
VE	-	-	20	20
	10	10	30	

Come possiamo ricavare questa matrice? In che modo possiamo trovare dei FEASIBLE FLOWS? Ricordiamo che una Soluzione Feasible soddisfa tutti i Vincoli, ma non è Ottimale!

Utilizziamo il concetto NORTH-WEST CORNER METHOD.

alla fine del processo risolutivo ci aspettiamo che la nostra soluzione ottenuta sia tale che ogni deposito sia completamente vuoto e ogni negozio sia esattamente soddisfatto, pertanto, partendo dalla

Viceversa, se avessimo avuto TOT Disp = 40 e TOT Rich = 20 avremmo introdotto un DEPOSITO FITTIZIO con Disp = 30 e nuovi costi C_{ij} per i quali faremmo le stesse considerazioni di prima.

- Abbiamo visto con XPRESS quanto sia importante avere un Modello Matematico associato ad un Real-Life Problem per 2 motivi:
  1) Ci aiuta a capire meglio il Real-Life Prob. in esame
  2) Possiamo Risolverlo con un Solutore

  Fino ad ora abbiamo visto 2 categorie di Problemi: MIN e MAX
- Introduciamo ora la categoria MINMAX. → BOTTLENECK PROBLEM

  In questo caso, NON abbiamo più una Combinazione Lineare delle nostre Variabili e quello che vogliamo è riuscire a trasformarla in una Combinazione Lineare. Come?

* $\underbrace{MIN\ MAX\{l_1, l_2, l_3\}}_{y}$ → Sostituiamo questa parte con una nuova variabile

$\Rightarrow MIN\ y$ → la più semplice Comb. Lineare possibile!

dove $y = MAX\{l_1, l_2, l_3\} \Leftrightarrow \begin{cases} y \geq l_1 \\ y \geq l_2 \\ y \geq l_3 \end{cases}$ che diventano i VINCOLI del mio nuovo Problema

$\Rightarrow$ $\boxed{MIN\ MAX\{l_1, l_2, l_3\}}$ DIVENTA $\boxed{\begin{array}{l} MIN\ y \\ \begin{cases} y \geq l_1 \\ y \geq l_2 \\ y \geq l_3 \end{cases} \end{array}}$ che è LINEARE!

Viceversa $\boxed{MAX\ MIN\{l_1, l_2, l_3\}}$ ⟶ $\boxed{\begin{array}{l} MAX\ y \\ \begin{cases} y \leq l_1 \\ y \leq l_2 \\ y \leq l_3 \end{cases} \end{array}}$

Pertanto introduciamo il concetto BIG M CONSTRAIN.

$$x_i \leq M_y$$

dove M avrà un limite superiore per x_i. Tale vincolo è corretto, in quanto l'unico modo per soddisfarlo è che sia $x_i > 0$, infatti in tal caso $y = 1$ e il vincolo ha senso. Altrimenti, se $x_i = 0$, in automatico $y = 0 \Rightarrow f_i$ sparisce, giustamente, dall'Obj. Func in quanto NON stiamo usando la macchina i e quindi questo vincolo NON ci serve.

$\Rightarrow$

$$\begin{aligned} &\text{MIN } c_i x_i + f_i y \\ &x_i \leq M_y \end{aligned} \qquad y \in \{0,1\}$$

• Proviamo a risolvere:

$$\boxed{\text{Se } x_i > 0 \Rightarrow y_j = 0}$$

I) Prima di tutto, dobbiamo introdurre 2 variabili booleane, una per ogni nostra variabile: $y_i, y_j \in \{0,1\}$

II) Dobbiamo rappresentare $x_i > 0$ attraverso y_i:

$$\text{Se } x_i > 0 \Rightarrow y_i = 1 \longmapsto x_i \leq M y_i \quad \text{visto prima}$$

$$\text{Se } x_j > 0 \Rightarrow y_j = 1 \longmapsto x_j \leq M y_j$$

Quindi: $x_i > 0 \equiv y_i = 1$

$x_j = 0 \equiv y_j = 0$

$$\Rightarrow \boxed{\text{Se } y_i = 1 \Rightarrow y_j = 0}$$ è equivalente a quella di partenza

che diventa una combinazione lineare del tipo

$$\boxed{y_i + y_j \leq 1}$$

infatti: Se $y_i = 1 \Rightarrow y_j = 0$ → UNICA AREA in cui funziona, OK

Se $y_i = 0 \Rightarrow y_j$ può essere 0 che 1

NB: NON possiamo usare uguaglianza perché implicherebbe che una tra y_i e y_j sia necessariamente posta a 1, richiesta eccessiva che il problema NON richiede

• COMPUTATIONAL COMPLEXITY

Prendiamo in considerazione il TSP. In particolare, consideriamo un INSTANCE of the TSP, dove per INSTANCE intendiamo una certa configurazione dei dati del TSP. Di quali dati abbiamo bisogno per rappresentare questo problema?

- M = Numero di clienti
- Matrice dei Costi, che è Simmetrica perché trattiamo Edges e NON Archi, e sulla matrice diagonale poniamo costi infiniti per evitare loops

Questi sono i dati necessari per avere una Istanza del Problema. Variandoli avremo diverse istanze (un problema una istanza è un esempio!)

- Un ALGORITMO è una Sequenza, una Lista di Different Steps
- L'EFFICACIA è una Misura della Qualità dei risultati di un Algoritmo mentre l'EFFICIENZA riguarda il Tempo richiesto da un Algoritmo per ottenere qualcosa
- In materia di Complessità Computazionale, abbiamo a che fare con ALGORITMI EFFICIENTI, quindi trascuriamo la Qualità. Infatti, se parliamo di ALGORITMI ESATTI, questi garantiscono una SOLUZIONE OTTIMALE, che è la migliore in assoluto, pertanto è inutile parlare di Efficacia visto che necessariamente otteniamo la migliore soluzione, e pertanto ci limitiamo ad analizzare l'EFFICIENZA!

Ovviamente, se invece avessimo a che fare con ALGORITMI EURISTICI allora dovremmo analizzare sia l'Efficienza che l'Efficacia.

26/10/12

SUMMARY

1) LOGICAL CONSTRAINTS

2) COMPUTATIONAL COMPLEXITY:

- Sia A il nostro Algoritmo. Abbiamo introdotto la funzione T_A, che è una misura del tempo di computazione di A (quanto tempo ci mette A)
- * $T_A = O(g(n)) \iff \exists c>0, \exists \hat{n} \,/\, T_A(n) \leq c\,g(n) \quad \forall n \geq \hat{n}$

Questo vuol dire che per enormi valori della dimensione del problema n (da $\hat{n}$ in poi), il tempo richiesto dall'algoritmo A per risolvere un problema di dimensione n non è peggiore della funzione $g(n)$, che può essere POLINOMIALE o NON POLINOMIALE: i primi sono EASY PROBLEM, i secondi NO!

- Vediamo adesso l'importanza della COMPLESSITÀ POLINOMIALE e NON per i nostri problemi:

CPU TIME	DIFFERENT PROBLEM SIZE n			
$g(n)$	10	20	---	60
POLINOMIALI: n	10^{-5} s	$2\cdot10^{-5}$ s	- -	$6\cdot10^{-5}$ s
POLINOMIALI: n^2	10^{-4} s	$4\cdot10^{-4}$ s	- -	0.036 s
POLINOMIALI: n^3	10^{-3} s		- -	0.216 s
NON POLINOMIALI: 2^n	10^{-3} s	1 s	- -	366 secoli
NON POLINOMIALI: 3^n	-		- -	$1.3\cdot10^{12}$ secoli

Ci si rende subito conto che anche di fronte a problemi di piccolissime dimensioni (solo 60!), nel caso NON-POL il tempo CPU richiesto diventa inaccettabile e, purtroppo, la maggior parte dei problemi di interesse ingegneristico reale rientrano proprio in questa categoria, perciò abbiamo bisogno di un metodo che ci aiuti.

Ovviamente, noi dobbiamo cercare la MIGLIOR FORMULAZIONE RILASSATA possibile così da ottenere il MINIMO ERRORE!

- Cerchiamo di capire se la potenza di un processore può realmente influire sulla capacità di risolvere ESATTAMENTE Problemi di Dimensione Maggiore.

DIMENSIONE della PIÙ GRANDE ISTANZA risolubile in 1 hour CPU

$g(n)$		COMP ATTUALI	100 VOLTE PIÙ VELOCI	1000 volte PIÙ VELOCI
POLI	n	N_1	$100\,N_1$	$1000\,N_1$
	n^2	N_2	$10\,N_2$	$31.6\,N_2$
	n^3	N_3	—	$10\,N_3$
NON POLI	2^n	N_4	$N_4 + 6.64$	$N_4 + 9.97$
	3^n	N_5	$N_5 + 4.19$	$N_5 + 6.29$

Pertanto, si capisce che per Problemi NON-POLINOMIALI, la migliorare la Tecnologia NON È SUFFICIENTE per risolvere problemi di dimensioni maggiori ⟹ Abbiamo bisogno di Algoritmi "Furbi"!!

- Un PROBLEMA di OTTIMIZZAZIONE dà come risultato un VALORE (Min, Max, ...) che deve soddisfare determinati Vincoli. Un PROBLEMA di DECISIONE, invece, dà come risultato una RISPOSTA, SI o NO, ad una certa Domanda.

Quello che ci chiediamo è: È possibile trasformare un Problema di Ottimizzazione in uno di Decisione?

* TSP $\xrightarrow{?}$ DECISION PROBLEM Y/N

Supponiamo che B sia un limite per lo shortest Tour, ossia sia un valore soglia che misura la lunghezza di un tour.

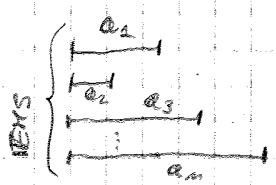
Ci chiediamo se esiste un Tour la cui lunghezza sia minore di B

RISPOSTA:
- NO ⟹ B è il nostro OTTIMO
- YES ⟹ B è troppo diventa nuova soglia e ricerchiamo, finché la risposta non sarà NO

Vediamo ora un esempio di Problema NP.

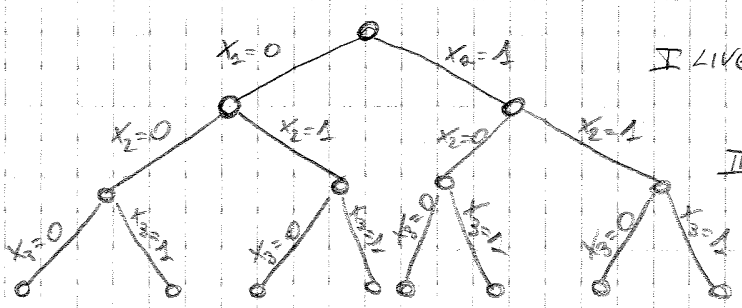
- SUBSET SUM PROBLEM:

Sia dato un segmento di lunghezza b e un insieme contenente n segmenti più piccoli



Vogliamo trovare un sottoinsieme di KEYS che copra esattamente Tutto il Segmento b!

* $n=3$: Per risolvere un tale problema dobbiamo costruire un SEARCH TREE



I LIVELLO $x_i = \begin{cases} 1, \text{ se } a_i \text{ è preso} \\ 0 \text{ altrimenti} \end{cases}$ $i = 1, \dots, n$

II LIVELLO

III LIVELLO

Ogni "percorso" (RAMO) dalla radice a foglia è una FEASIBLE SOLUTION!

Questo perché la Nostra Objective function è: Esiste Tale Sottoinsieme $\Rightarrow$ Risposta è SI o NO, non stiamo MIN o MAX mente!

Per quanto riguarda la COMPLESSITÀ, nel caso peggiore dobbiamo esaminare TUTTI i Rami, il che significa "contare" TUTTE le FOGLIE!

$\Rightarrow$ COMPLESSITÀ $= 2^n$ $\Rightarrow$ NP-Problem!

$\nexists$ Alcun Algoritmo Efficiente per risolverlo!!

30/10/12

SUMMARY

Nell'ultima lezione abbiamo parlato di:

- COMPLESSITÀ COMPUTAZIONALE
- COMPLESSITÀ di un ALGORITMO: è una misura del tempo computazionale necessario per risolvere il Problema associato

In realtà, però, noi siamo interessati alla COMPLESSITÀ di un PROBLEMA: è la Complessità dell'Algoritmo Migliore che abbiamo a disposizione per risolverlo

* SORTING PROBLEM

Sappiamo che la Complessità di Questo Problema è $O(n \log n)$, data dal Quicksort che è il Miglior Algoritmo che abbiamo a disposizione per risolverlo, MA non è l'unico, ce n'è un altro che lo risolve in $O(n^2)$, che, però, è Peggiore, Meno Efficiente e pertanto sosteniamo che la Complessità di Questo Problema è $O(n \log n)$

- L'ultima volta abbiamo visto anche il SEARCH TREE, che è un Problema Booleano. Qual è la sua Complessità? Un modo per misurarla è quello di "contare" le Foglie, e pertanto avremo $O(2^n)$, che è il numero di tutte le possibili Soluzioni. Un altro modo è legato al numero di LIVELLI presenti nel nostro albero, pertanto la SEARCH DEPTH è uguale a n, ossia è lineare rispetto alla Dimensione del Problema (para ad n, appunto). Questo è MOLTO Importante perché si può concludere, allora, che il Problema è NP!

Vale allora:

" Se abbiamo un Problema P, e da esso deriviamo un SEARCH TREE, la cui DEPTH sia uguale ad un Polinomio in n, allora si può affermare che P ∈ NP "

NB: NP ≡ NON-DETERMINISTIC POLYNOMIAL !

- PROBLEM SIMMETRY

Consideriamo TSP_Δ, dove la Δ sta ad indicare che ne stiamo considerando il Problema Decisionale associato. In tal caso, quello che ci chiediamo è:

"$\exists$ un Hamiltonian Tour la cui lunghezza sia minore di una data soglia B?"

Ciò che ci aspettiamo è una Risposta Positiva, ossia SI è la risposta "buona", in modo da poter essere certi di aver trovato una Soluzione Ottima, altrimenti la Soluzione Ottima è già data (H-Tour di lungh. B) e quindi non ha senso risolvere il problema.

Ora: E' possibile riformulare questa domanda in modo che la risposta "buona" sia NO? Ossia in modo tale che possiamo dire di aver trovato l'Ottimo se rispondiamo NO?

"Ogni Hamiltonian Tour ha lunghezza maggiore di B?"

Così formulato, se rispondiamo NO vuol dire che migliorano e quando troviamo l'Ottimo.

MA: cosa possiamo dire della Complessità delle 2 diverse formulazioni? Nel primo caso il Problema è NP, infatti dobbiamo trovare le 2 fasi per risolverlo. Nel secondo caso, invece, la fase di test viene effettuata in un Tempo NON-Polinomiale, in quanto devo prendere in considerazione Tutti gli Ham-Tour, il che comporta una complessità $O(n!)$

Questa osservazione è molto importante perché ci fa capire che dobbiamo essere bravi a formulare il Problema Decisionale nel modo giusto, altrimenti la Complessità Esplode!!

} Pertanto, NON Esiste SIMMETRIA per i PROBLEMI NP!

Per farlo abbiamo bisogno di un importante concetto:

Consideriamo 2 Problemi: Π e Π'

Prima di tutto diciamo che

Π' può essere RIDOTTO a Π $\rightarrow$ $\Pi' \propto \Pi$

$\Updownarrow$

Π' è un CASO PARTICOLARE di Π

* Subset Sum Problem (SS) $\propto$ Knapsack Problem (KP)

$\Updownarrow$

SS è un Caso Particolare di KP

Questo vuol dire che ogni istanza I di SS può essere trasformata IN TEMPO POLINOMIALE in un'istanza I' di KP tale che se risolviamo I' allora risolviamo anche I

Proviamo a farlo:

Consideriamo un'istanza $I \in SS$, il che vuol dire che abbiamo una certa lunghezza b e certi valori $\{a_i\}$

Vogliamo trovare da I un $I' \in KP$ in tempo polinomiale.

Ricordiamo il Modello del KP:

$$\max \sum_{i=1}^{n} p_i x_i$$

$$\sum_{i=1}^{n} w_i x_i \leq W$$

$$x_i \in \{0,1\}, \forall i$$

Pertanto dobbiamo cercare di trovare una relazione tra i dati p_i, w_i, W del KP e i dati $b, \{a_i\}$ del SS

Quindi possiamo dire che $b \simeq W \rightarrow$ 2 condizioni, 2 Problemi:

$$a_i \simeq w_i$$
$$a_i \simeq p_i$$

Sono relazioni lineari $\Rightarrow$ il passaggio da I a I' viene fatto in TEMPO POLINOMIALE $\Rightarrow$ OK!

Pertanto avremo I', una formulazione del KP:

$$\max \sum_{i=1}^{n} a_i x_i$$

$$KP \ni I' \qquad \sum_{i=1}^{n} a_i x_i \leq b$$

$$x_i \in \{0,1\}, \forall i$$

In realtà, nel SS non vogliamo che la somma scelta copra esattamente b, quindi il vincolo = non stiamo formulando KP, pertanto dobbiamo essere fedeli al Modello KP, non lo possiamo alterare

21/10/16

- **SUMMARY**

1) SIMMETRIA dei Problemi = quelli POLINOMIALI sono SIMMETRICI, gli NP, invece, NO, e pertanto per questi è importante poter risolvere nel modo giusto il Decision Problem

2) Abbiamo introdotto il concetto di SATISFIABILITY PROBLEM:

* $y = \underbrace{x_2}_{\text{PROPOSIZIONE (CLAUSE)}} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee \bar{x}_3)}_{\text{PROPOSIZIONE}} \wedge \underbrace{x_3}_{\text{PROPOSIZIONE}}$ dove $x_1, x_2, x_3 \in \{0,1\}$

Ogni Proposizione C_j è data dalla relazione fra le variabili booleane x_i attraverso l'Operatore "OR":

$C_j = (C_{1j} \vee C_{2j} \vee \dots \vee C_{kj})$ dove $C_{ij} = \{x_1, x_2, \dots x_m, \bar{x}_1, \bar{x}_2, \dots, \bar{x}_m\}$

$\Rightarrow$ $\boxed{y = C_1 \wedge C_2 \wedge C_3}$

Vogliamo verificare che esista una certa combinazione delle variabili booleane x_i tale che y sia SODDISFATTA, ovvio $y = TRUE\ (=1)$

Questo è un Esempio di SATISFIABILITY PROBLEM!

- Ci chiediamo SAT $\in$ NP?

In generale, per verificare che un Problema sia NP, ci sono 2 modi:

1) Vedere se riusciamo a derivare un SEARCH TREE come quello che abbiamo visto qualche lezione fa — che abbia Profondità Polinomiale

2) Trovare un algoritmo che si comporta nelle 2 note Fasi (Hypothesis di Soluzione e Test della stessa in Tempo Polinomiale)

Proviamo ad usare questo secondo metodo per trovare una risposta:

a) Hp: $x_1 = 1,\ x_2 = 1,\ x_3 = 1$

b) TEST: $y = 1 \wedge (0 \vee 1 \vee 0) \wedge 1 = 1 \wedge 1 \wedge 1 = 1$ → OK

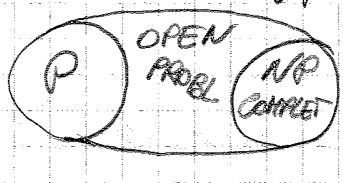
Il Test è stato effettuato in un Tempo Polinomiale! Perché possiamo, se

m = # Clauses
n = # Boolean Variables

$\Rightarrow$ Test: $O(m \cdot n)$ — Polinomiale nella dimensione n del Problema!

QUINDI Tutti i problemi NP-Complete sono EQUIVALENTI in termini di Complessità, HANNO la STESSA COMPLESSITÀ!!
Questo è molto importante perché, come abbiamo, se riuscissimo a trovare un Algoritmo Polinomiale che risolve UNO di questi problemi NP-Complete allora immediatamente li abbiamo risolti TUTTI!
E in tal caso affermeremmo che P=NP!!
MA la sensazione è che questo sia improbabile e che in realtà P≠NP, anche se NON c'è Nessuna prova matematica che lo dimostri!

- Esiste una 3ª Classe di Problemi che SONO in NP MA NON appartiene né a P né agli NP-Complete in quanto NON siamo in grado di dire nulla sulla loro complessità. Questi vengono detti OPEN PROBLEMS



Quindi, consideriamo di avere un Nuovo Problema π e di sapere che $\pi \in NP$. Dove collochiamo π?
Se riusciamo a trovare un $\pi' \in$ NP-Complete e a dimostrare che $\pi' \propto \pi$, allora π sarà difficile ALMENO quanto π'
$\Rightarrow$ $\pi \in$ NP-Complete
Se invece NON riusciamo a trovare un tale π', allora collocheremo π tra gli OPEN PROBL.

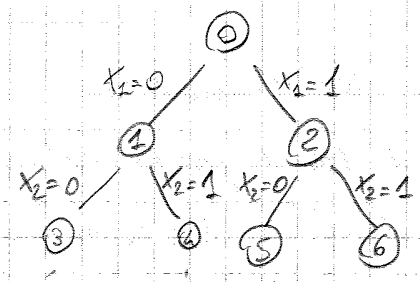
BRANCH & BOUND

- NON è un Algoritmo, in quanto NON possiamo definire Esattamente ogni passo MA è, piuttosto, un METODO GENERALE, una sorta di "FILOSOFIA" per risolvere i cosiddetti COMBINATORIAL PROBLEMS.

NB: Con COMBINATORIAL intendiamo INTEGER, ossia Problemi le cui Variabili sono INTERI!

- È un METODO ESATTO, ossia garantisce una SOLUZIONE OTTIMA! (Se il Problema è risolvibile, ovviamente...)
- Sono 2 le idee alla base di questo metodo:

1) BRANCHING, partendo dal nostro Problema iniziale P_0, lo SEPARIAMO in una serie di Sottoproblemi P_i e così via, assicurandoci, però, che in questa fase di Separazione NON ci si PERDA di FEASIBLE SOLUTIONS ossia l'unione delle Feasible Solutions di ogni singolo Sottoproblema P_i deve essere uguale all'insieme di tutte le Feasible Solutions del Problema Iniziale P_0. Abbiamo già incontrato un'applicazione di questa Branching idea quando abbiamo parlato del Boolean Search Tree.



Abbiamo già constatato, tuttavia, che una RICERCA COMPLETA all'interno di questo albero costa $O(2^n)$, pertanto vorremmo cercare un modo per "tagliare" alcuni rami così da ridurre la dimensione dell'albero stesso!

Questa è l'idea che è alla base del:

2) BOUNDING, esistono 3 FATHOMING CRITERION, ossia per "eliminare" un generico nodo i del mio albero e ridurre, così, la mia ricerca devo soddisfare almeno 1 dei seguenti criteri:

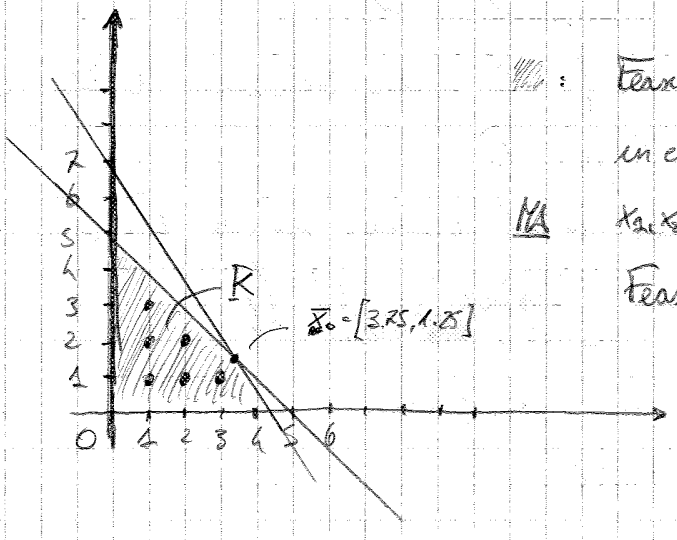
è vista che il TSP è un Problema di MIN, LB_i non è migliore di qualsiasi LB_j visto i, pertanto il nodo i può essere tagliato

c) NON IMPROVING SOLUTIONS, ossia Troviamo una Feasible Solution che non è MIGLIORE di un'altra che abbiamo già trovato

* Assumiamo che esista una Feasible Solution per TSP che abbia un valore $\bar{z}$, ossia $\bar{z}$ = lunghezza dell'Ham. Tour. Se in un nodo i per il problema P_i troviamo un $LB_i > \bar{z}$, posso tagliare i rami visto che soluzioni già peggiori di LB_i. Ovviamente se troviamo $LB_i = \bar{z}$, questo è equivalente al criterio b). Il Lower Bound è dato da una Feasible Solution, MA se $LB_i = \bar{z}$ che è un Feasible Optimum, allora LB_i è associato ad una Feasible Solution e pertanto ci ritroviamo nel caso b)!

* Proviamo a vedere con un esempio numerico l'applicazione di queste 2 idee.

$$P_0: \quad \min -5x_1 - 4x_2$$
$$x_1 + x_2 \leq 5$$
$$10x_1 + 6x_2 \leq 45$$
$$x_1, x_2 \in \mathbb{Z}^+$$



: Feasible Space del problema P_{0R} RILASSATO in cui $x_1, x_2 \geq 0$

MA $x_1, x_2 \in \mathbb{Z}^+ \Rightarrow$ pallini rossi sono il vero Feasible Space di P_0

Vogliamo risolvere P_0 con il Branch and Bound Method!

Visto che i coefficienti dell'Objective Function sono interi e stiamo cercando soluzioni intere, vuol dire che l'Ottimo che troveremo sarà necessariamente intero; pertanto possiamo approssimare i Lower Bound con i valori interi maggiori più vicini, quindi possiamo considerare

$$LB_2 = \lceil -23.3 \rceil = -23$$

e, per il Criterio c), visto che abbiamo già trovato $LB_1 = -23$, possiamo chiudere anche il Nodo P_2

$\Rightarrow$ La' OPTIMAL SOLUTION per P_0 è quella che abbiamo già trovato prima: $\boxed{\bar{x}^* = [x_1^*, x_2^*] = [3,2]}$

e l'OPTIMUM sarà $\boxed{-23}$

QUINDI: Si parte dal Bound, che è sempre una procedura di RELAXATION, e poi si fa BRANCH...

1) ELIMINATING CONSTRAINTS

È il più semplice in quanto consiste nel rimuovere uno o più vincoli del nostro Problema

* TSP:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ij}$$

CONSTRAINTS
$$\begin{cases} \sum_{j=1}^{n} x_{ij} = 1 \quad \forall i \\ \sum_{j=1}^{n} x_{ji} = 1 \quad \forall i \\ \text{HAMILTON CONST} \end{cases}$$

$$x_{ij} \in \{0,1\} \quad \forall i, \forall j$$

→ Per calcolare il LB eliminiamo il vincolo più difficile e CERCHIAMO L'OTTIMO del "nuovo" problema, che in questo caso riconosciamo essere l'ASSIGNMENT PROBL (ricorda: vogliamo assegnare un set di macchine ad un set di lavorazioni)

NB. L'Assignment Probl è POLINOMIALE pertanto è un EASY PROBLEM!

Quest'ultima osservazione è molto importante perché dobbiamo muovere in questa direzione ogni volta che dobbiamo risolvere, dobbiamo fare in modo tale che il "nuovo" probl sia Easy!

E se volessi calcolare l'UPPER BOUND per il TSP?

Un UB per un TSP è l'Objective function di una qualsiasi Feasible Sol. Ad esempio Costruiamo una Feasible Sol usando un Greedy Criterio come il Nearest Neighbourhood e ne calcoliamo il valore dell'Obj Fun associata ad essa. Solitamente si indica un UB con $\bar{z}$ e abbiamo visto che, sempre per Fathoming Criteria possiamo dire che, se in un nodo i troviamo $LB_i \geq \bar{z}$, allora si può Fathom il nodo i, in quanto al più potrò fare uno sforzo computazionale trovare una soluzione migliore di $\bar{z}$.

Passiamo a P_2: $x_{21} = 0 \iff c_{21} = \infty$

Troviamo $LB_2 = 13 \longrightarrow \dots$

MA, $LB_2 > \bar{z}$ $\Rightarrow$ inutile scrivere soluzione relativa a LB_2, tanto ho già una soluzione migliore! $\Rightarrow$ fathom P_2 $\Rightarrow$ Fine Algoritmo!

$\Rightarrow$ OPTIMAL SOLUTION: $\begin{cases} x_{13} = x_{21} = x_{34} = x_{45} = x_{52} = 1 \\ \text{others } x_{ij} = 0 \end{cases}$

MINIMUM COST = 11

2) LAGRANGEAN RELAXATION

Consideriamo il problema

DIFFICULT PROBLEM $\begin{cases} \min f(x) \\ g_i(x) \geq 0, \ i = 1, \dots, m \\ x \in Y \end{cases}$

Se avessimo $\begin{cases} \min f(x) \\ x \in Y \end{cases}$ questo è un EASY PROBLEM $\Rightarrow$ l'aggiunta di quei vincoli lo ha reso DIFFICULT

L'idea è, quindi, TRASFORMARE i VINCOLI in OBIETTIVI, ossia vogliamo prendere quei vincoli e farli entrare nell'Obj. Func con lo scopo di avere l'Easy Probl. Così facendo stiamo "chiedendo meno": trasformando vincoli in obiettivi stiamo chiedendo meno.

Come facciamo? Facciamo una COMBINAZIONE LINEARE dei "DIFFICULT" CONSTRAINTS e la combiniamo con l'Obj. Func:

$$L(\bar{x}, \bar{\mu}) = f(\bar{x}) - \sum_{i=1}^{m} \mu_i g_i(\bar{x})$$

questa nuova funzione prende il nome di LAGRANGEAN FUNCTION!

Otteniamo così il RELAXED PROBLEM

$$\min L(x, \bar{\mu})$$
$$x \in Y$$
$$\bar{\mu} \geq \bar{0}$$

Pertanto affinché $LR_0(\mu)$ si avvicini il più possibile a $LR_0(\mu)^*$ ne dobbiamo prendere il MAX rispetto alla sua unica variabile μ.
Si ~~è~~ altresì così un processo iterativo (risolvo Primal → Risolvo Dual → Risolvo Rem-z...) che termina o dopo un tot di tempo fissato o quando si raggiunge una distanza soglia fra $LR_0(\mu)$ e $LR_0(\mu)^*$ Fissata o ...

* ESERCIZIO per CASA
  - FACILITY LOCATION PROBLEM (pag 45 Libro Verde)
    - > Scrivi il Modello Matematico
    - > Rilassa il Primo Vincolo attraverso Lagrangean Relaxation
    - > ~~Risolvi~~ Risolvi il Problema

MAX Total Profit = Demand Satisfaction − Location Costs

Vogliamo allocare Servers: ogni Server j ha un Costo fisso f_j
Definiamo h_{ij} = Profitto nel servire customer i con server j
Pertanto Obj. Fun:

MATH. MODEL

$$\max \underbrace{\sum_{i=1}^{n}\sum_{j=1}^{m} h_{ij} x_{ij}}_{\text{Demand Sat}} - \underbrace{\sum_{j=1}^{m} f_j y_j}_{\text{Location Cost}}$$

$$x_{ij} \in \{0,1\} \quad \forall i, \forall j \qquad x_{ij} = \begin{cases} 1 & \text{se customer } i \text{ servito con server } j \\ 0 & \text{altrimenti} \end{cases}$$

$$y_j \in \{0,1\} \quad \forall j \qquad y_j = \begin{cases} 1 & \text{se server } j \text{ è allocato} \\ 0 & \text{altrimenti} \end{cases}$$

Questo è il vincolo che vogliamo rilassare in modo Lagrangiano

CONSTR
- $\sum_{j=1}^{m} x_{ij} = 1$ → Ogni customer è servito da 1 solo server
- $x_{ij} \le y_j \quad \forall i, \forall j$ → $\sum y_j = 0 \Rightarrow x_{ij} = 0 \ \forall j$ → LOGICAL CONSTR

Ci dice che se un server è chiuso, allora nessun customer può essere servito da esso

Conformiamo con i Vincoli: • $\sum_{j=1}^{m} x_{ij} = 1 \quad \forall i$ → Ogni customer servito da 1 solo server

• $x_{ij} \le y_j \quad \forall i, \forall j$ → Logical Constraint:

$x_{ij} \in \{0,1\} \quad \forall i, \forall j$ se $y_j = 0 \Rightarrow x_{ij} = 0 \quad \forall i$

$y_j \in \{0,1\} \quad \forall j$ ossia se il server è chiuso, nessun customer può essere servito da quel server!

• Vogliamo RILASSARE il primo Vincolo, notando che esso può essere riscritto come

$$1 - \sum_{j=1}^{m} x_{ij} = 0$$

pertanto introduciamo i Coll. Lagrange $\mu_i \ge 0 \quad \forall i$

e definiamo la quantità $\sum_{i=1}^{n} \mu_i \left(1 - \sum_{j=1}^{m} x_{ij}\right)$

Visto che è un Prob di MAX, cerchiamo un UB ⇒ Sommiamo questa quantità alla nostra Obj. Func

⇒ Avremo

RISOLVENDO QUESTO PROBLEMA dell'Ottimalità troveremo un UPPER BOUND!

$$\begin{cases} \max L(\bar{x}, \bar{y}, \bar{\mu}) = \sum_{i=1}^{n}\sum_{j=1}^{m} h_{ij} x_{ij} - \sum_{j=1}^{m} f_j y_j + \sum_{i=1}^{n} \mu_i \left(1 - \sum_{j=1}^{m} x_{ij}\right) \\ x_{ij} \le y_j \quad \forall i, \forall j \\ x_{ij} \in \{0,1\} \quad \forall i, \forall j \\ y_j \in \{0,1\} \quad \forall j \end{cases}$$

L'Obj. Fun può essere aggregata un po':

$$\max L(\bar{x}, \bar{y}, \bar{\mu}) = \sum_{j=1}^{m} \left(\sum_{i=1}^{n} (h_{ij} - \mu_i) x_{ij} - f_j \right) y_j + \sum_{i=1}^{n} \mu_i$$

Vediamo di trovare una Soluzione "a mente", dato $\bar{\mu}$:

> È meglio avere $y_j = 1$ solo se il suo coeff è Positivo (dato vogliamo MAX), altrimenti è meglio avere $y_j = 0$

⇒ $y_j = 1$ SE $\sum_{i=1}^{n} (h_{ij} - \mu_i) - f_j > 0$

> Per lo stesso motivo

$x_{ij} = 1$ SE $h_{ij} - \mu_i > 0$ e $\boxed{y_j = 1}$

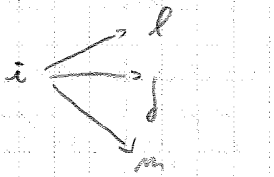
↳ Per VINCOLO LOGICO !!

HEURISTICS

1) POLYNOMIAL CONSTRUCTIVE HEURISTICS

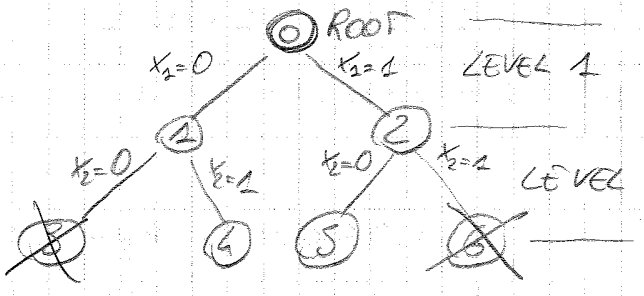
a) MYOPIC RULE (GREEDY)

Un esempio di Metodo appartenente a questa famiglia è il Nearest Neighbor Method visto per il TSP, in cui da un generico nodo i mi muovo verso il nodo "più vicino" ancora visto. È un Metodo "Greedy" (= "Ghiottone") in quanto è come se il nodo i "mangiasse" quello più vicino ancora visto. Teniamo a precisare che un Metodo Euristico Qualsiasi dà come risultato Sempre Feasible Solution!
Un modo per migliorare un Metodo Greedy è quello di introdurre una certa CASUALITÀ, nel caso dell'NN, ad esempio:



l è il più vicino
l è il secondo più vicino

se vogliamo migliorare il nostro metodo, dobbiamo fare sì che da i si vada qualche volta in j, qualche volta in l, cioè fare in modo che la scelta sia CASUALE e non determinata in maniera esatta!
Ovviamente la scelta può essere fatta anche tra più di 2 nodi, l'importante è che sia CASUALE! In questo modo, il nostro Metodo Greedy sarà sicuramente MIGLIORE!!



Arrivati al 2° livello, abbiamo 4 Nodi, ma $K=2 \Rightarrow$ Dobbiamo chiudere 2.
Come facciamo? Scegliamo i più "promettenti", andando, ad esempio, a guardare i Lower Bound:

LB_3: $x_1=0, x_2=0 \Rightarrow 15+11+5+3 = 34$ (prendo x_3, x_4, x_5, x_6)

LB_4: $x_1=0, x_2=1 \Rightarrow 36+11 = 47$ (prendo x_2, x_4, gli altri posso vendo capacità)

LB_5: $x_1=1, x_2=0 \Rightarrow 30+15+3 = 48$ (x_2, x_3, x_6)

LB_6: $x_1=1, x_2=1 \Rightarrow$ Capacità $x_1+x_2 = 21 > 12 \Rightarrow$ NO!

$\Rightarrow$ Nodo 6 INFEASIBLE!!

Dobbiamo chiudere un altro nodo: per selezionarlo, scegliamo il "migliore", ad esempio prendendo quello che hanno LB maggiore $\Rightarrow$ chiudo 3!

OSS: Per caricare gli items nello zaino abbiamo considerato gli items in ordine così come stanno, ossia una volta soddisfatti i vincoli, abbiamo guardato prima x_3, visto se rispettava capacità o meno, poi o meno, poi x_4, poi x_5, poi x_6. E' giusto fare così? Qual'è l'ordine "giusto" da seguire per caricare lo zaino?

L'ORDINE GIUSTO è: mettere in ordine NON-INCREASING i rapporti $\frac{p_i}{w_i}$ $\frac{\text{PROFITTO}}{\text{PESO}}$

Nel caso numerico di sopra, gli items sono già ordinati in questo modo: siamo stati fortunati!

20/11/22

• SUMMARY

HEURISTICS

1) CONSTRUCTIVE POLYNOMIAL HEURISTICS

È un'enorme famiglia. Ci sono 2 parole chiave:

- POLYNOMIAL: significa che richiede un tempo polinomiale per arrivare a convergenza
- CONSTRUCTIVE: la soluzione viene "costruita" in vari steps, ad ogni passo iterativo aggiungiamo un "pezzo" di soluzione e la soluzione finale viene costruita in tempo polinomiale!

2) LOCAL SEARCH HEURISTICS. Argomento di oggi!

NB. Alla 1) famiglia appartengono: GREEDY e SIMPLIFYING AN EXACT METHOD.

* LOCAL SEARCH HEURISTICS

1) BASED ON CLASSICAL LOCAL SEARCH

Con il termine "classical" intendiamo che ci stiamo riferendo a

- STEEPEST DESCENT
- FIRST IMPROVEMENT

2) METAHEURISTICS

- TABU SEARCH (TS)
- SIMULATED ANNEALING (SA)
- GENETIC ALGORITHMS (GA/EA)
- ...

⇒ Avremo $\frac{n^2}{2}$ elementi (considerando solo la parte triangolare superiore) a cui sottrarre gli n elementi della diagonale

⇒ $\frac{n^2}{2} - n = \frac{n(n-1)}{2}$ ⇒ $\boxed{O(n^2)}$ MISURA della DIMENS. del NEIGHBORHOOD $V_{\bar{x}}$

E' MOLTO IMPORTANTE che la dimensione del $V_{\bar{x}}$ sia POLINOMIALE perché esaminiamo TUTTE le soluzioni contenute in esso!

Quindi, cerchiamo di schematizzare la nostra procedura:

a) INIZIALIZZAZIONE: Cerchiamo una feasible solution iniziale $\bar{x}_0$

b) GENERIAMO NEIGHBORHOOD:

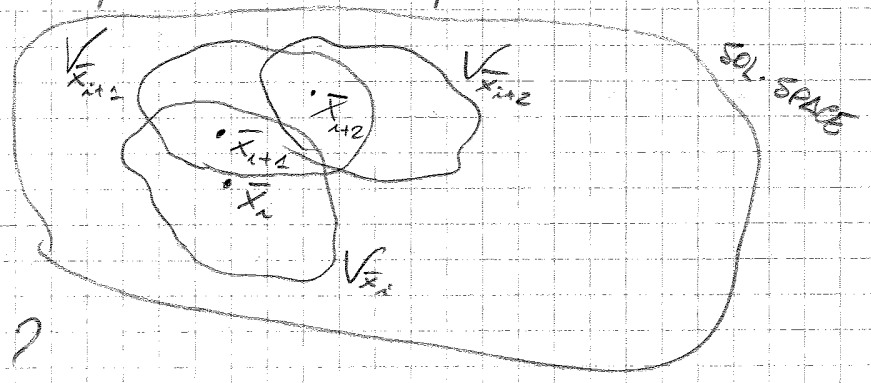
Sia $\bar{x}_i$ la feasible sol. attuale

→ Generiamo $V_{\bar{x}_i}$

→ Selezioniamo un "vicino" particolare in $V_{\bar{x}_i}$

→ lo identifichiamo con $\bar{x}_{i+1}$

→ Iteriamo, ripetendo il tutto a partire da $\bar{x}_{i+1}$ ...



Quando ci Fermiamo?

Suggerimenti:

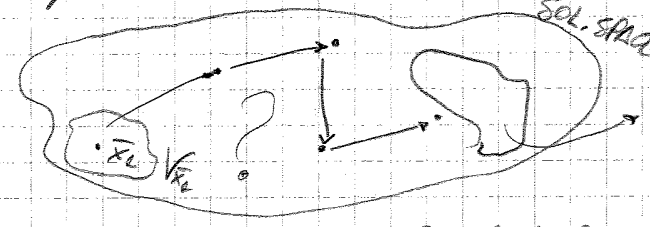
> "Ci Fermiamo quando raggiungiamo un certo gap dall'Optimal Solution"

NO, perché non conosciamo l'Optimal Solution, non siamo in grado di risolverla (non è un Metodo Esatto) pertanto non sappiamo dire quando ci siamo vicini

> "Ci fermiamo quando esaminiamo una certa percentuale di tutto il Solution Space".

NON percorribile, in quanto l'intero Sol. Space è dell'ordine di $O(n!)$, mentre noi esploriamo solo sottospazi dell'ordine di $O(n^2)$

Sono 2 Modi Differenti. Ciò che è certo è che il Secondo è PIÙ VELOCE MA PEGGIORE! → In termini di qualità!
Ciò che viene usualmente è usare un MIX dei due approcci:



che è Molto Vicino a quello del Sol Space!!

Noi non conosciamo niente di $V_{\bar{x}_i}$, pertanto inizialmente potrebbe essere utile usare il First Approach così che in pochissimo tempo raggiungiamo regioni molto lontane da $V_{\bar{x}_i}$ e, poi, una volta individuata una "regione Buona", allora usare lo Steepest in modo da trovare la Migliore Soluzione. Se pensiamo di poter fare meglio, usiamo il First per andare in un'altra regione e così via.

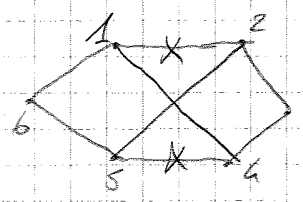
- Come si stabilisce che una Regione è "Buona"?

È una questione di "esperienza" ed è estremamente legato al tipo di problema che stiamo affrontando: Cerchiamo di analizzare in dettaglio quelle regioni che ci sembrano possano dare buone soluzioni.

- Continuando le Fasi del Local Search:
  c) SOLUTION ACCEPTANCE TEST → MOLTO IMPORTANTE
  d) STOPPING CONDITION → già vista

Analizziamo c):
Scelto $\bar{x}_i$, generiamo $V_{\bar{x}_i}$ e con Steepest troviamo $\bar{x}_{i+1}$ = Best in $V_{\bar{x}_i}$
Se $\bar{x}_{i+1}$ è Migliore di $\bar{x}_i$ ⇒ ci Muoviamo in $\bar{x}_{i+1}$ ⇒ lo ACCETTIAMO ⇒ Accettare vuol dire che ci Muoviamo da $\bar{x}_i$ a $\bar{x}_{i+1}$

* TSP_2



$\bar{x}_1 = 1\ 2\ 3\ 4\ 5\ 6$ Fea. Sol.

$\bar{x}_2 = 1\ 4\ 3\ 2\ 5\ 6 \rightarrow$ Fea Sol OK

* KP. $\bar{x}_1 = 0\ 1\ 1\ 0\ 1$ Fea. Sol, ora la matrice è sbbuffata

Come Fcso Neighbor? Ad esempio posso sostituire un 1 con uno 0 o viceversa:

$\bar{x}_2 = 1\ 1\ 1\ 0\ 1 \rightarrow$ Feas? $\rightarrow$ Boh!

$\bar{x}_3 = 0\ 1\ 0\ 0\ 1 \rightarrow$ OK

Oppure, come prima, scambiamo una coppia

$\bar{x}_4 = 0\ 0\ 1\ 1\ 1 \rightarrow$ Feas? $\rightarrow$ Boh!

$\Rightarrow$ FEASIBILITY PROBLEM FOR THE NEIGHBORHOOD

Come facciamo a costruire un Neighborhood che sia composto da elementi che sono SICURAMENTE FEA. SOL? $\rightarrow$ Quanto che facciamo vale per KP.

Generalmente, si fa un ordinamento degli elementi, ad esempio in maniera lineare, scegliamo $x_1, x_2, x_3, \dots, x_n$

Poi si procede così: $x_1 = 1$ SE $w_1 \le W$ altrimenti $x_1 = 0$

$x_2 = 1$ SE $w_1 x_1 + w_2 \le W$ " $x_2 = 0$

$x_3 = 1$ SE $w_1 x_1 + w_2 x_2 + w_3 \le W$ " $x_3 = 0$

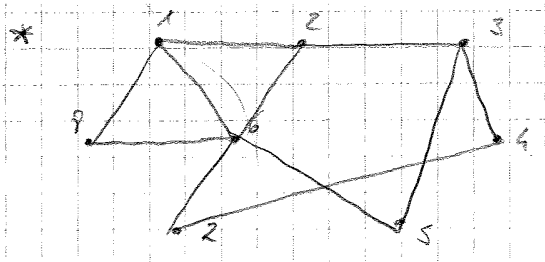
⋮

• Osa: Quanto costa calcolare l'Obj. Func. per i Vicini?

$\geq TSP_2$ Se abbiamo calcolato $OF_{\bar{x}_1}$ (Obj. Func per $\bar{x}_1$), è possibile derivare tutte le altre da questa solo sommando o sottraendo 2 numeri. $\bar{x}_2$ differisce con scopo di 2 e 4 $\Rightarrow$ Perdo archi $\overline{12}$ e $\overline{45}$ ma aggiungo archi $\overline{14}$ e $\overline{25}$

$\Rightarrow$ $OF_{\bar{x}_2} = OF_{\bar{x}_1} - c_{12} - c_{45} + c_{14} + c_{25}$

Sono tutti valori COSTANTI



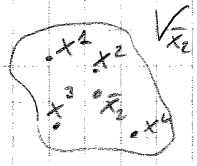
$\bar{x}_1 = \begin{cases} 1,3,7 = W \\ 5 = Y \\ 6 = R \\ 2,4,8 = G \end{cases}$

$\bar{x}_1$ è una FEA SOL ottenuta con 4 colori

Proviamo a usarne 3. Eliminiamo Y:

$\bar{x}_2 = \begin{cases} 1,3,7 = W \\ 5,6 = R \\ 2,4,8 = G \end{cases}$ → INFEASIBLE perché ora 56 è in conflitto!

Proviamo a costruire $V_{\bar{x}_2}$: un vicino di $\bar{x}_2$ potrebbe essere ottenuto cambiando il colore di 5 o di 6, che possono diventare, avendo ora 3 colori a disposizione, o G o W, pertanto ho solo 4 vicini:



x^1: nodo 5 diventa W
x^2: nodo 5 diventa G
x^3: nodo 6 diventa W
x^4: nodo 6 diventa G

⇒ Dobbiamo ora esaminare $V_{\bar{x}_2}$, ossia calcolare l'Obj_Fun relativa

⇒ OSS. Il nostro obiettivo ora è eliminare l'INFEASIBILITY, pertanto l'Obj Func = Numero di archi in conflitto ⇒ vogliamo Obj_Fun = 0!

Quindi:

	Obj Func	
x^1	1	5: R → W
x^2	1	5: R → G
x^3	2	6: R → W
x^4	2	6: R → G

⇒ Migliore soluzione di $V_{\bar{x}_2}$ è x^1 ($o\ x^2$)

⇒ Ci muoviamo in x^1 e otteniamo

$\bar{x}_2 = \begin{cases} 1,3,5,7 & W \\ 6 & R \\ 2,4,8 & G \end{cases}$ con 1 CONFLITTO (arco 35)

⇒ Ripetiamo e cerchiamo nuovo Neighborhood, gestendo nodi 3 e 5 che sono in conflitto

21/11/12

- SUMMARY
  - CLASSICAL LOCAL SEARCH METHODS
    - > STEEPEST DESCENT
    - > FIRST IMPROVEMENT
  - METAHEURISTICS
    - > TABU SEARCH
    - > SIMULATED ANNEALING

- TABU SEARCH

TABU vuol dire Proibito e l'idea di questo metodo è di PROIBIRE dei MOVIMENTI (TABU MOVES)

* TSP $\bar{x}_1 = 1\ \overset{\curvearrowright}{2}\ \overset{\curvearrowleft}{3}\ 4$ $\Rightarrow$ MOVE = SWAP 2-3

$\bar{x}_2 = 1\ 3\ 2\ 4$

Per evitare che nel corso della procedura si esplori una soluzione già esaminata (quindi per evitare di passare da $\bar{x}_2$ a $\bar{x}_1$) PROIBIAMO dei MOVIMENTI, ossia, in questo caso, definiamo come TABU MOVE lo swap 2-3.

Tuttavia, questo modo di procedere è un po' troppo restrittivo in quanto, dopo un elevato numero di movimenti, ripetere un movimento già fatto potrebbe essere del tutto insufficiente, non ci fa evitare soluzioni già viste. Ad esempio:

$\bar{x}_i = 1\ 2\ 3\ \overset{\curvearrowright}{4}\ \overset{\curvearrowleft}{5}\ 6\ 7\ 8 \longrightarrow \bar{x}_{i+1} = 1\ 2\ 3\ 5\ 4\ 6\ 7\ 8$

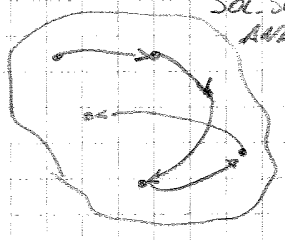
TABU LIST = {4-5, ... }

Dopo un elevato numero di movimenti, sicuramente qualche altro elemento al di fuori di 4 e 5 si è spostato, pertanto se ripetessi lo swap 4-5, di sicuro visiterò una nuova soluzione e non $\bar{x}_i$.

Pertanto ciò che si fa è PROIBIRE i MOVIMENTI SOLO per un certo periodo di tempo!

OSS: ASPETTI IMPORTANTI del TABU SEARCH:

1) Definire un BUON Neighborhood, BUONO in termini di:
   - DIMENSIONE: NON deve essere troppo grande perché non dobbiamo esaminarlo tutto, tutti gli elementi
   - COMPLESSITÀ della VALUTAZIONE dell'Obj. Func: è molto importante avere una complessità bassa quando si va a valutare l'Obj. Func per tutte le soluzioni presenti in esso
2) Lunghezza della TL: è MOLTO importante che NON sia troppo lunga, altrimenti avremo troppi move bloccati e quindi il calcolatore è troppo limitato, ci ritroveremo in una fase di stallo perché non possiamo più muoverci



Con una TL molto piccola, abbiamo la possibilità di spaziare tutto il Sol Space, altrimenti ci areneremo su una piccola regione!

L'ideale sarebbe avere una TL Dinamica, piccola all'inizio per esplorare e più grande, quando andiamo a ridurre la regione, per analizzarla in profondità (Analogia Fuori Improvement - Sleeper!)

3) TERMINATION TEST. ← MAX Computing TIME ↘ MAX Numero di ITERAZIONI

* Riesaminiamo esempio GRAPH COLOURING:

Ad un certo punto abbiamo cambiato colore al nodo S_2

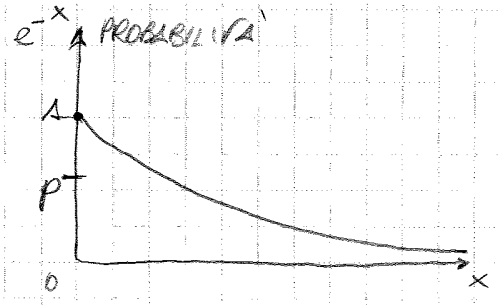
$S: R \to W \Rightarrow$ Mettiamo nella TL il movimento contrario, per non tornare indietro

$\Rightarrow TL = \{S: W \to R,$

In seguito, abbiamo ottenuto:

x^1	$S: W \to G$	2	
x^2	$S: W \to R$	0	→ OK, ha anche fatto la TABU MOVE, la prendiamo per ASPIRATION CRITERION!
x^3	$S: W \to G$	1	
x^4	$S: W \to R$	1	→ TABU MOVE, ma tanto non è stata migliore di quella trovata prima

Obj Funct = #conflitti



È una funzione di questo tipo, dove

$$x_2 \quad \frac{F(\hat{x}) - F(\tilde{x})}{T(I)}$$

dove: $F(\tilde{x}), F(\hat{x})$ sono valori della fg. func

$T(I)$ è la TEMPERATURA all'iterazione

Abbiamo detto che $\hat{x}$ PEGGIORE di $\tilde{x}$ ⇒ essendo un MIN Probl

vuol dire $F(\hat{x}) > F(\tilde{x})$ ⇒ $F(\hat{x}) - F(\tilde{x}) > 0$

Inoltre T è funzione del Numero di iterazioni all'inizio (I piccolo), è molto elevata, poi dopo tante iterazioni, I grande e T si abbassa

Questo vuol dire che:

I piccolo → $T(I)$ Elevata → x piccolo ⇒ Probabilità Molto Alta

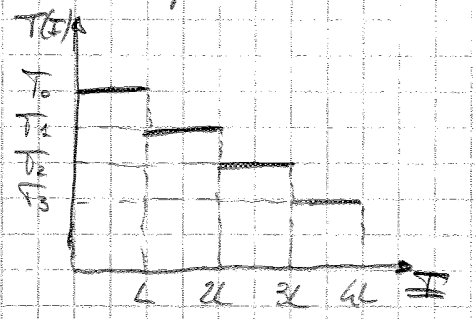
I grande → $T(I)$ Bassa → x elevato ⇒ Probabilità Molto Bassa

Pertanto definiamo una SOGLIA p:

Se $\hat{p} > p$ ⇒ ACCETTIAMO Sol. PEGGIORE $\hat{x}$

Altrimenti ⇒ NON ACCETTIAMO

▷OSS: Tipicamente, $T(I)$ è una funzione a scalini:



Se stiamo considerando PLATEAU di lunghezza L:

$$T(kL) = T_k = \alpha^k T_0$$

dove α = TASSO di RIDUZIONE della TEMP.

$$0 < \alpha < 1$$

Tipicamente $\alpha \geq 1$, questo perché se la distanza tra 2 temperature è molto piccola, il che significa che il Raffreddamento è Molto Lento e quindi si esplora di più il Sol-Space. Ovviamente dipende da Problema e da quali informazioni abbiamo sul Sol-Space: se sappiamo come è fatto, un buon esperto potrebbe essere capace l'α migliore!

Se, ad esempio, sappiamo che il Sol Space è CONVESSO, abbiamo un ott. Minimo Globale ⇒ Scegliamo α molto piccolo così che si raffredda velocemente, tanto sappiamo di trovare un unico minimo!

DIPENDE! Se ad ogni iterazione preferiamo tenere sempre le Best Solutions, il set INIT. POP si DIVERSIFICA, in quanto combinare soluzioni simili fra loro ci vale a soluzioni altrettanto simili. Questo può essere un vantaggio se, all'interno del nostro SOL. SPACE, ci sentiamo sono in area individuata come buona regione e vogliamo analizzarla a fondo. MA, nel caso in cui dobbiamo bruscamente DIVERSIFICARE le nostre soluzioni, perché dobbiamo esplorare il SOL. SPACE e individuare Regione Buona, allora sarebbe preferibile FARE SOSTITUZIONE e NON tenere a prescindere le Best Sol.
Ciò che si fa è questo: si va a calcolare il coefficiente FITNESS dell'INIT. POP, ossia una funzione che rappresenta la Misura della Qualità delle Soluzioni dell'INIT. POP. A questo punto, in base ad esso, si ~~cerca~~ elimina la best e worse sol e si sostituisce una parte dell'INIT. POP contenente un "egual" numero di best e worse, si cerca un compromesso!
A questo punto, dopo sostituzione, si ha una NEW POP, si ritorna a 2) e si itera.
3) STOP CRITERIA, come al solito Computing time.
Vediamo meglio in dettaglio tutto quanto detto!

→ MAIN FEATURES OF GA

1) SOLUTION CODING.
Generiamo i CROMOSOMI, i cui elementi vengono definiti GENI

2) STOPPING CRITERIA.
Semplicemente, MAX AVAILABLE COMPUTING TIME, che è comunque equivalente ad un Max # Iterazioni, in quanto ogni iterazione ha un costo preso in termini di Computing time

7) POPULATION SUBSTITUTION (UPDATING)

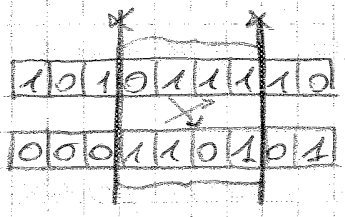
Vediamo GENERATIONAL REPLACEMENT, STEADY STATE, ...

8) IF STOPPING CRITERIA ne SATISFIED → STOP
OTHERWISE GO to 3)

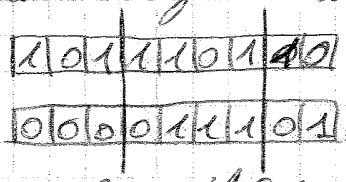
→ REPRODUCTION PHASE

* STANDARD CROSSOVER:

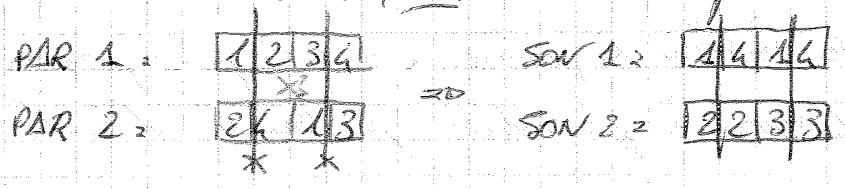
Esempio: KP



Scegliamo 2 punti di taglio e generiamo 2 figli "scambiando" il pezzo tagliato:



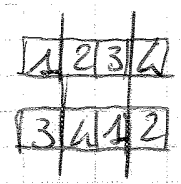
Questo metodo, però, ha un LIMITE, c'è la possibilità di GENERARE INFEAS. SOL !!
Questo è evidente nel KP, ma lo è ancor di più nel TSP:



INFEASIBLE !!

Per ottenere Feasible Solution possiamo usare il modello ORDER CROSSOVER:
consideriamo i cromosomi come se fossero delle catene chiuse ("anelli"); invece di scambiare i pezzi tagliati, lasciamo questi al loro posto e riempiamo partendo in ordine a partire da destra con i geni dell'altro cromosoma, tenendo conto di una volta "punto" il cromosoma genitore, lo continuo a leggere ripartendo da sinistra

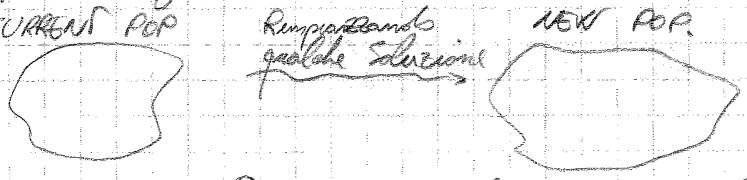
⇒



OK, ora FEASIBLE!

Cerchiamo, così, di fare un'idea dell'ambito di HYBRIDIZING, ossia combinare 2 diversi Kernels Evolutivi!

Ad esempio, se vogliamo HYBRID. GA:



HA, prima di Kellere Nuove Soluzioni nella Nuova Popolazione (Prima del rimpiazzo), non OTTIMIZZIAMO le NUOVE SOLUZIONI, ossia attraverso un qualsiasi LOCAL SEARCH APPROACH, "Migliorianno" i Figli, nati dalla combinazione di Geni della Current Pop, prima di inserirli nella New Pop!!

Proviamo a sviluppare il Sistema dei Vincoli del Flow Balance:

$$\begin{cases} \text{Nodo 1.} & x_{12} + x_{13} = 20 \\ \text{Nodo 2.} & x_{24} + x_{26} + x_{23} - x_{12} = 5 \\ \text{Nodo 3.} & x_{34} + x_{35} - x_{13} - x_{23} = 0 \\ \text{Nodo 4.} & x_{46} + x_{45} - x_{24} - x_{34} = 0 \\ \text{Nodo 5.} & -x_{35} - x_{45} - x_{65} = -15 \\ \text{Nodo 6} & [\text{illegible}] + x_{65} - x_{26} - x_{46} = -10 \end{cases}$$

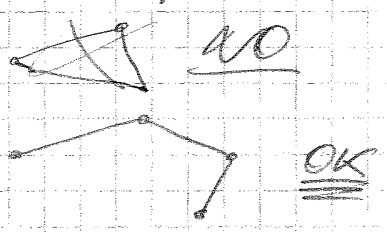
$n = \#$ Nodi
$m = \#$ Archi

Pertanto, la Matrice dei Coefficienti è formata da n Righe e m Colonne, $\forall$ Arco (i,j), presenta un $+1$ nella riga i e un -1 nella riga j

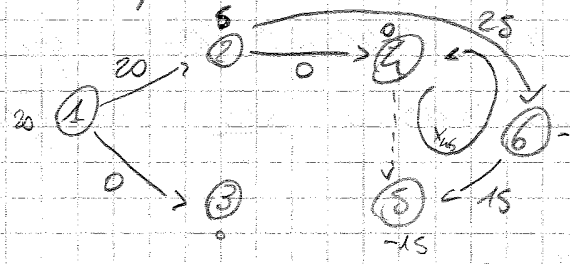
$$A = \begin{matrix} \\ i \\ \\ j \\ \\ \end{matrix} \overset{(i,j)\ \text{m colonne}}{\begin{bmatrix} 0 & \cdots \\ +1 & \cdots \\ \vdots & \\ -1 & \cdots \\ 0 & \cdots \end{bmatrix}} \quad n \text{ nodi}$$

Questa particolare tipologia di Matrice prende il nome di
MATRICE di INCIDENZA
ed è peculiare di ogni Grafo!!

Quanto vale il RANGO di A, $\rho(A)$?
Se sommiamo tutte le righe fra loro, otteniamo necessariamente 0 perché ogni colonna presenta un solo $+1$ e un solo -1 $\Rightarrow \rho(A) < m$
In A, basta eliminare 1 sola riga i e di conseguenza perdiamo un qualche $+1$ o -1 da qualche parte $\Rightarrow$ $\boxed{\rho(A) = n-1}$
Questo è molto importante perché significa che $n-1$ Archi bastano per rappresentare una Soluzione del Problema, ossia $n-1$ Archi sono LINEARMENTE INDIPENDENTI
Archi Linearmente Indipendenti sono tali da NON formare un CICLO
* 3 Archi Lin. Ind. $\Rightarrow$

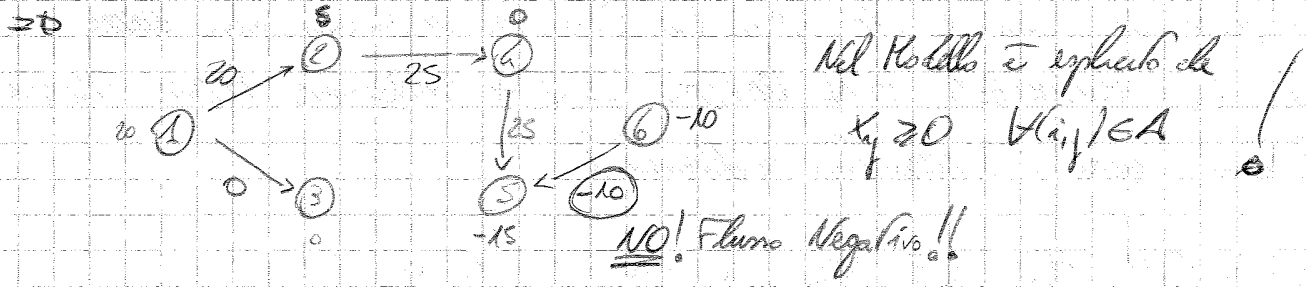


Inserendo (2,4), si forma un ciclo ⇒ Abbiamo 3 opportunità: (1,2), (1,3), (3,4). MA, la regola, non possiamo eliminare un arco che ha lo stesso verso del nuovo ⇒ NON (1,2)!
Tra (1,3) e (3,4) la scelta è chiaramente indifferente, visto che, per rispettare Vincoli Flow Balance, $x_{34} = 0$ così come i flussi x_{13} e x_{34}
Perciò, ad esempio, ELIMINIAMO (3,4).



Nuovo Spanning Tree.
Lo Valutiamo → NO ottimo
⇒ Vediamo: Nuovo Arco
Elimino uno Vecchio

Inseriamo (4,5) ⇒ Ciclo con quel verso ⇒ Candidati ad essere eliminati sono (2,6) e (6,5) che verso opposto. Di Regola si <u>elimina</u> quello con <u>FLUSSO MINORE</u>, in pratica, se eliminassimo (2,6)

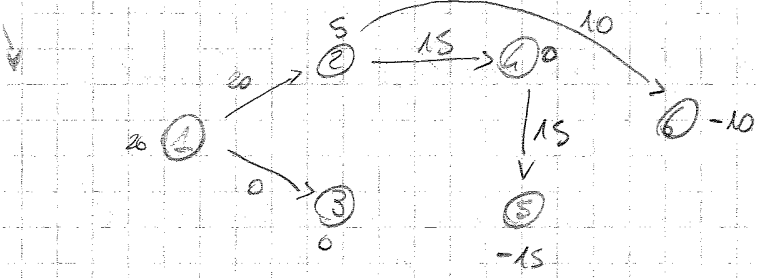


Nel Knapsack è esplicito che $x_{ij} \geq 0 \quad \forall (i,j) \in A$

<u>NO</u>! Flusso Negativo!!

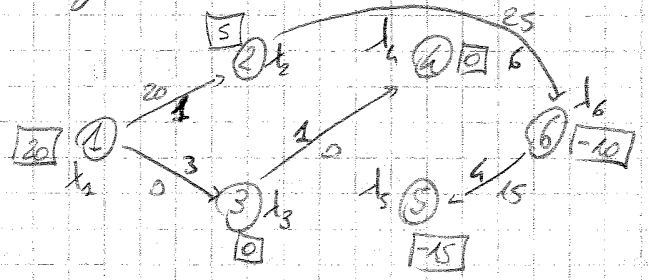
⇒ ELIMINO (6,5)

- Ci resta da Vedere COME CENTRO INIZIALE SPANNING TREE e COME LO VALUTO, decidendo se è OTTIMO o MENO!



⇒ OSS: Uno Spanning Tree è un supporto per la mia soluzione, MA NON posso dire che una data base sono ho trovato una Initial Feasible Solution in quanto abbiamo bisogno di qualcosa in più, dobbiamo soddisfare il Bilancio dei b_i, dobbiamo "caricare" questo Span. Tree con i Flussi!

Supponiamo di aver già trovato una INITIAL FEAS. SOL (per ora diciamo così!)



Procediamo con l'Algoritmo --

- A volte capita che una INIT. FEA. SOL sia già OTTIMA, pertanto vogliamo il controllare se lo è o meno. Come?

Introduciamo il concetto di NODE POTENTIAL l_i, ossia, ad esempio, il Costo dell'Arco $(1,2)$, $c_{12}=1$, lo vediamo come una DIFFERENZA di POTENZIALE fra il Potenziale l_1 del Nodo 1 e il Potenziale l_2 del Nodo 2. Pertanto abbiamo:

$$\begin{cases} c_{12} = l_1 - l_2 \\ c_{13} = l_1 - l_3 \\ c_{26} = l_2 - l_6 \\ c_{34} = l_3 - l_4 \\ c_{65} = l_6 - l_5 \end{cases}$$

m INCOGNITE ⇒ NO RISOLVIBILE!
m-1 EQUAZ

MA, i l_i sono definiti a meno di una costante
⇒ possiamo assegnare ad uno qualsiasi di essi un qualsiasi valore!

Pertanto, scegliamo di fissare il valore $l_6 = 0$

$$\Rightarrow \quad \underline{l} = [2, 6, 4, 3, -4, 0]^T$$

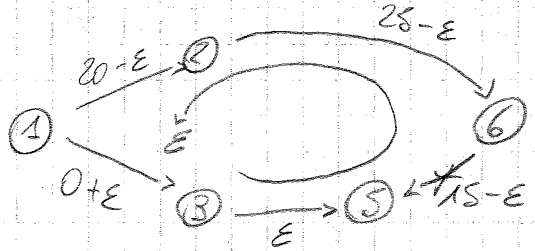
A questo punto calcoliamo π_j $\forall$ Arco di ξ all'attuale Feas. Sol.

VR: Non è necessario calcolarli tutti, ci fermiamo appena ne troviamo uno negativo!

$\pi_{23} = 1 \rightarrow 0$ $\qquad$ $\pi_{35} = -3 \rightarrow$ OK!

$\pi_{31} = 2 \rightarrow 0$

$\Rightarrow$

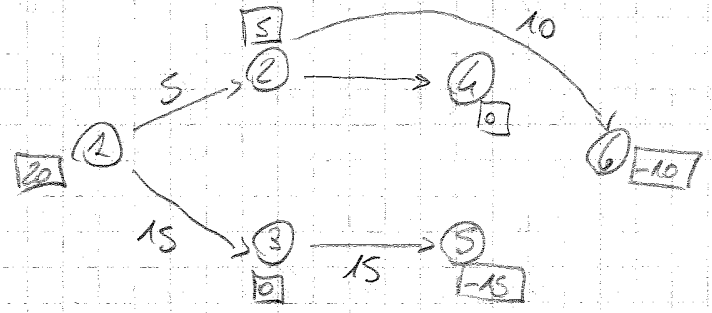


Candidati: (1,2)
(2,6)
(6,5)

Per evitare Sol NON Feas, scegliamo sempre archi con FLUSSO MINORE!

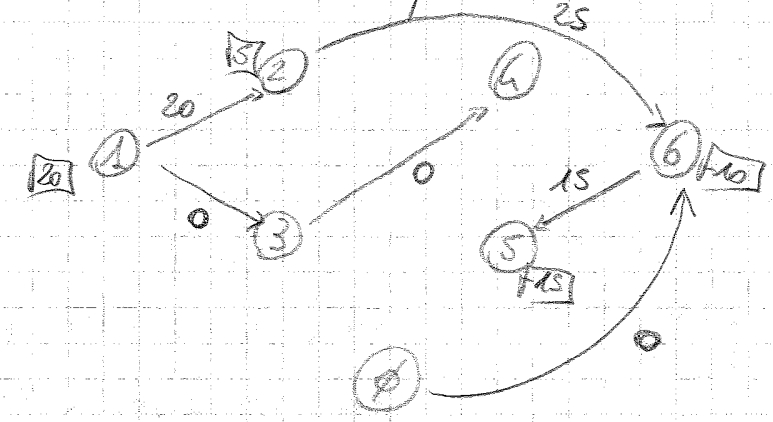
$\Leftrightarrow$ Eliminiamo (6,5) $\Rightarrow$ $\xi = 15$

$\Rightarrow$



III FEAS. SOL!!

Ma questo non crea alcun problema in quanto possiamo scegliere uno qualsiasi dei 4 archi che lo lasciano, tanto alla fine delle iterazioni il FLUSSI su questi archi saranno nulli (li dobbiamo appunto "scaricare"!!)
Quindi l'OPTIMAL SOLUTION per PROBL AUSILIARIO sarà



da cui ricaviamo IN[II] FEA. SOL per il Problema di partenza!!

DOMANDA

Come si riduce un VINCOLO un HAM-TOUR?

$\sum x_{ij} \leq |S|-1$ dove S è un qualunque sottoinsieme di nodi

NB: $2 \leq |S| \leq n-2$

Questi archi bisogna distinguerli in:

- FORWARD, ossia USCENTI dal CUT, come $(1,3)$ e $(1,2)$ e $(1,4)$
- BACKWARD, ossia ENTRANTI nel CUT, come $(3,1)$

Quindi dato S, che è un insieme di Nodi, viene definito H_S, ossia il Cut dato da S, che contiene appunto gli Archi

$\Rightarrow$ nel nostro esempio: $H_S = \{(1,3), (1,2), (1,4), (3,1)\}$ (F, F, F, B)

Definiamo a questo punto la CUT CAPACITY, C_{H_S}, che è pari alla Somma delle Capacità solo degli Archi Forwards del Cut H_S:

$$C_{H_S} = \sum_{(i,j) \in F_S} c_{ij} \quad , \quad F_S = \text{set of Forw. Arcs of cut } H_S$$

$\Rightarrow$ Nel nostro caso $C_{H_S} = 12$.

Ora: **LEMMA**:

Dato H_S, abbiamo appunto F_S e B_S. Si può verificare che

$$\sum_{(i,j) \in F_S} x_{ij} - \sum_{(i,j) \in B_S} x_{ij} = v \rightarrow \text{Current Flow on the Network, quello che alla fine dovrà risultare MAX}$$

Inoltre, il Current Flow v e la Cut Capacity C_{H_S} risulta

$$\boxed{v \leq C_{H_S}}$$

Infatti, c'è un TEOREMA (MAX FLOW = MIN CUT) che asserisce

$$\boxed{\max v = \min C_{H_S}} \rightarrow \text{teorema importante!}$$

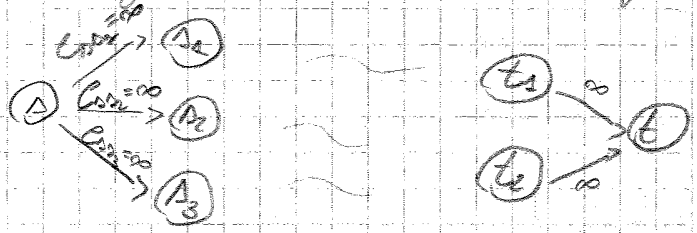
Pertanto i 2 Problemi sono DUALI: avranno lo stesso OTTIMO (= Valore Obj. Func.) Ovviamente, non la stessa opt. sol, infatti nel primo caso sarà un FLOW $x_{ij}(v)$, mentre nel secondo sarà una CAPACITY (C_{H_S}).

19/12/12

MAX FLOW PROBLEM

Continuiamo esercizio della altra volta:

> BASIC IDEA: vogliamo trovare (se c'è) un Augmenting Path tra s e t



NB: Se avessimo più Sinks e più Sources, possiamo sempre ricondurci al caso con una sola Sink e una sola Source introducendo s e t e collegando ai Sinks e alle Sources con degli archi FITTIZI con CAPACITÀ INFINITA

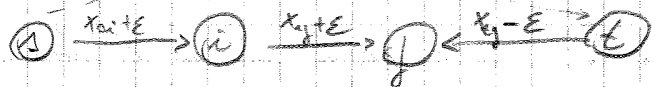
Vogliamo trasportare la maggior quantità di "roba" possibile da s a t. $\rightsquigarrow$ Cerco Flow

Vogliamo MAX v

pertanto cerchiamo un Augmenting Path per incrementare $v \longrightarrow v+\varepsilon$ con $\varepsilon > 0$, dello FLOW AUGMENTATION

DEF. AUGMENTING PATH P

Consideriamo:



assumiamo che questo sia un AUG. PATH da s a t: $s \xrightarrow{\text{AUG. PATH}} t$

Definiamo: FARWARD ARC, un arco che ha lo stesso verso del PATH

BACKWARD ARC, un arco che ha verso opposto al PATH

NB: non confondere con F e B dei Cut!

Visto che un AUG. PATH è un cammino che individuiamo sul nostro grafo e grazie al quale possiamo incrementare flusso di una quantità ε, allora:

- $s \xrightarrow{F} j \quad (i,j) \in P \overset{SE}{\equiv} x_{ij} < c_{ij}$
- $j \xrightarrow{B} t \quad (t,j) \in P \overset{SE}{\equiv} x_{tj} > 0$

La prima ci dice che sull'arco abbiamo flusso minore di Capacità, perciò abbiamo una Capacità extra non utilizzata, e quindi se aumentiamo flusso di ε, (i,j) riesce a trasportarlo. La seconda dice che ora su (t,j) abbiamo flusso $x_{tj}-\varepsilon$, perché

09/01/13

MAX FLOW PROBLEM

Ricordiamo le Principali caratteristiche di questo Problema

Abbiamo $G=(N,A)$ CAPACITATED NETWORK

- Vogliamo MAX Flusso tra SOURCE (s) a SINK (t)
- Modello:

$$\max v \longrightarrow \text{Variabile}$$

Capacity Constraints → $x_{ij} \le c_{ij} \quad \forall (i,j) \in A$

Network Balance → $\sum_{j \in N} x_{ij} - \sum_{j \in N} x_{ji} = \begin{cases} v, & \text{if } i \text{ is SOURCE} \\ 0, & \text{if } i \text{ is TRANSIT NODE} \\ -v, & \text{if } i \text{ is SINK} \end{cases} \quad \forall i \in N$

$$x_{ij} \ge 0$$

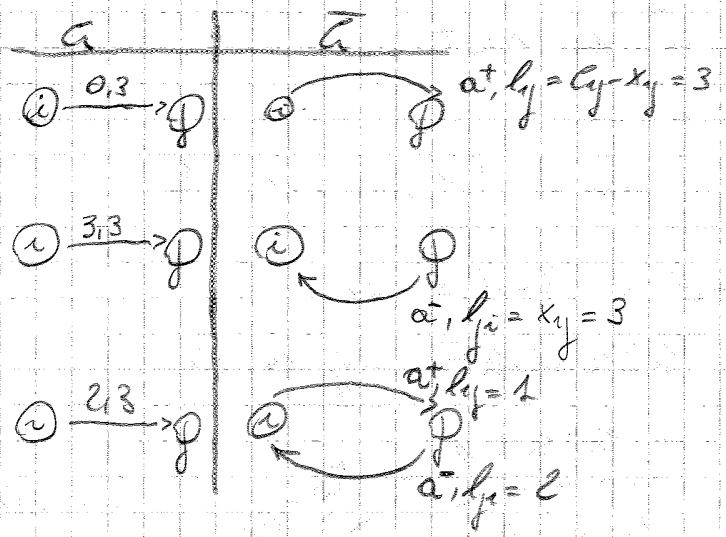
- THEOREM. MAX FLOW = MIN CUT
- AUGMENTING PATH

$$\varepsilon = \min(\varepsilon_1, \varepsilon_2)$$

$$\varepsilon_1 = \min_{(i,j) \in P_F} (c_{ij} - x_{ij}) \qquad P_F = \text{Set of Forward Arcs of Path } P$$

$$\varepsilon_2 = \min_{(i,j) \in P_B} x_{ij} \qquad P_B = \text{Set of Backward Arcs of Path } P$$

- RESIDUAL GRAPH: $\bar{G} = (N, \bar{A})$



AUG. PATH $\Leftrightarrow$ ELEMENTARY PATH in $\bar{G}$

Più semplice da trovare perché in essi TUTTI gli Archi sono FORWARD!