Problemi NP-completi e Algoritmi euristici

Problema dello zaino (Knapsack)

Ci sono n oggetti, ne conosciamo il valore p_j e l'ingombro w_j, $j=1,\ldots,n$. È data inoltre la capacità massima, b, di un contenitore.

Problema: quali oggetti inserire nel contenitore rispettando il limite di capacità

Obiettivo: massimizzare il valore degli oggetti inseriti

Esempio di applicazione: costruire il Cd ideale.

Ogni oggetto è un file musicale il cui valore è dato dal nostro indice di gradimento ed il cui ingombro è la dimensione in kbyte. Il contenitore è un Cd-rom con 700 Mbyte di capacità. Nell'ipotesi che la dimensione complessiva dei file musicali a disposizione ecceda i 700 Mbyte vogliamo scegliere quali brani inserire massimizzando il gradimento complessivo

Variabili=Decisioni: quali oggetti inserire

$$x_j = \begin{cases} 1 & \text{se il } j\text{-esimo oggetto viene inserito nello zaino} \\ 0 & \text{altrimenti} \end{cases} \qquad j = 1,\ldots,n$$

Modello

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

$$\sum_{j=1}^{n} w_j x_j \le b$$

$$x_j \in \{0,1\}, \quad j = 1,\ldots,n$$

in alternativa

$$0 \le x_j \le 1, \quad x_j \text{ intero}, \ j = 1,\ldots,n$$

oppure

$$0 \le x_j \le 1, \quad x_j \in \mathrm{Z}^+, \ j = 1,\ldots,n$$

Problema dell'assegnamento

Ci sono n persone in grado di svolgere n attività. Ad ogni persona deve essere assegnata una sola attività ed ogni attività deve venir svolta da una sola persona. Ad ogni coppia (persona i, attività j) è associato un costo c_{ij} che esprime le maggiori o minori attitudini di ciascuna persona a svolgere le diverse attività.

Problema: come assegnare le attività alle persone

Obiettivo: minimizzare il costo complessivo dell'assegnamento

Variabili=Decisioni: da quali persone vengono svolte le varie attività

$$x_{ij} = \begin{cases} 1 & \text{se la persona } i \text{ solge l'attivita' } j \\ 0 & \text{altrimenti} \end{cases}$$

Modello

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

$$(a) \quad \sum_{i=1}^{n} x_{ij} = 1 \quad j = 1,\ldots,n$$

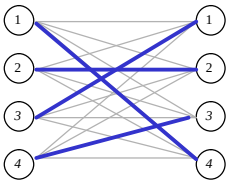
ogni attività è svolta da una sola persona

$$(b) \quad \sum_{j=1}^{n} x_{ij} = 1 \quad i = 1,\ldots,n$$

ogni persona svolge una sola attività

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

Esempio numerico di applicazione



persone attività

	1	2	3	4
1	10	12	8	5
2	8	5	9	6
3	7	9	3	8
4	10	2	7	11

Matrice dei costi c_{ij}

Questo assegnamento ha costo 5 + 5 + 7 + 7 = 24

Ne esiste uno migliore?

Una differente applicazione: compagni di camera

All'inizio dei corsi nei college americani gli studenti forniscono un indice del gradimento di condividere la camera (a due posti) con ciascuno degli altri studenti. Il personale docente assegna a coppie gli studenti alle camere in modo da massimizzare il gradimento complessivo.

N.B. In questo caso la funzione obiettivo viene massimizzata.

Problema di assegnamento e sequenziamento

Ci sono *m* macchine identiche ed *n* lavorazioni. Ogni lavorazione *j*, con *j=1,…,n*, richiede di essere processata da una qualsiasi delle *m* macchine per un tempo di processamento ininterrotto p_j. Ogni macchina processa una sola lavorazione alla volta.

Problema: come assegnare le lavorazioni alle macchine.

Obiettivo: minimizzare l'istante di completamento di tutte le lavorazioni.

Variabili=Decisioni: quali lavorazioni sono assegnate a ciascuna macchina.

$T =$ istante di completamento di tutte le lavorazioni

$$x_{ij} = \begin{cases} 1 & \text{se la macchina } i \text{ esegue la lavorazione } j \\ 0 & \text{altrimenti} \end{cases}$$

Modello

$\min T$

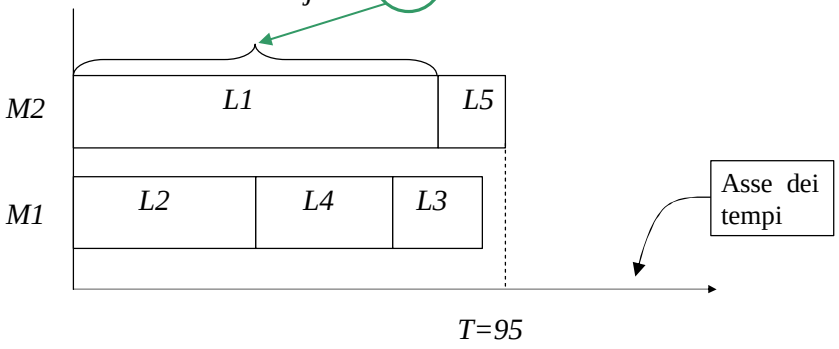
$$(a) \quad \sum_{j=1}^{n} p_j x_{ij} \le T \qquad i = 1,\dots,m$$ definizione del minimo istante di completamento

$$(b) \quad \sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1,\dots,n$$ ogni lavorazione è svolta da una sola macchina

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

Esempio numerico di applicazione

$n = 5, \quad m = 2, \quad p_j = \{80, 40, 20, 30, 15\}$

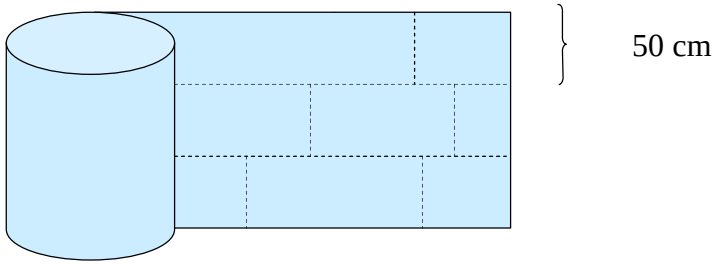


Soluzione con $x_{12} = x_{13} = x_{14} = x_{21} = x_{25} = 1$

N.B. Una volta che le lavorazioni sono assegnate alle macchine l'ordine nel quale esse vengono processate è irrilevante

Una diferente applicazione: taglio di tessuto

E' disponibile un rotolo (molto lungo) di tessuto pregiato, alto 150 cm. E' necessario ritagliare *n* pezzature di 50 cm di altezza e di lunghezza variabile fra i 50 ed i 250 cm. Come assegnare le *n* pezzature alle *m*=3 strisce ricavabili dal rotolo in modo da minimizzare i metri di tessuto utilizzati?

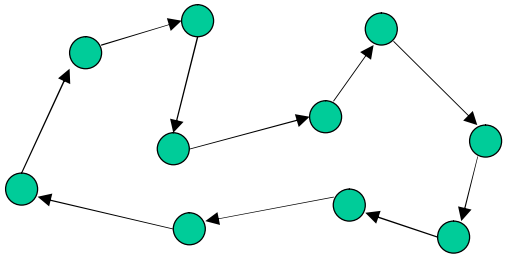


Il problema del commesso viaggiatore (T*ravelling* S*alesman* P*roblem*)

Un commesso viaggiatore deve visitare ciascuna di *n* città esattamente una volta e ritornare al punto di partenza. Il tempo necessario per andare dalla città *i* alla città *j* è c_{ij}.

Problema: deteminare la sequenza di visita delle città

Obiettivo: completare il ciclo nel minor tempo possibile



Variabili=Decisioni: l'ordine di precedenza di visita per ciascuna coppia di città

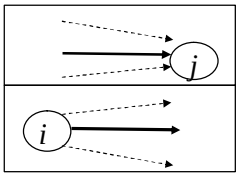
$$x_{ij} = \begin{cases} 1 & \text{se il commesso viaggiatore va direttamente dalla citta' } i \text{ alla citta' } j \\ 0 & \text{altrimenti} \end{cases}$$

$N=\{1,2,\ldots,n\}$ l'insieme delle città

Modello

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

$$(a) \quad \sum_{i=1}^{n} x_{ij} = 1 \quad j = 1,\ldots,n$$



un arco entrante **in** ogni nodo

un arco uscente **da** ogni nodo

$$(b) \quad \sum_{j=1}^{n} x_{ij} = 1 \quad i = 1,\ldots,n$$

$$(c) \quad \sum_{i\in S}\sum_{j\in S} x_{ij} \le |S| - 1 \quad \forall S \subseteq N, 2 \le |S| \le n-1$$

vincoli di eliminazione dei sottocicli

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

$$(a) \quad \sum_{i=1}^{n} x_{ij} = 1 \quad j = 1,\ldots,n$$

$$(b) \quad \sum_{j=1}^{n} x_{ij} = 1 \quad i = 1,\ldots,n$$

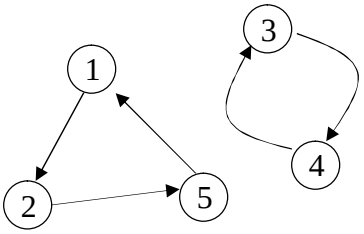
$$x_{ij} \in \{0,1\} \quad i, j = 1,\ldots,n$$

Questi vincoli formano un modello identico a quello già visto del problema dell'assegnamento.

Fra le soluzioni che li vi sono però anche quelle che collegano le città in modo da formare un insieme di cicli disgiunti

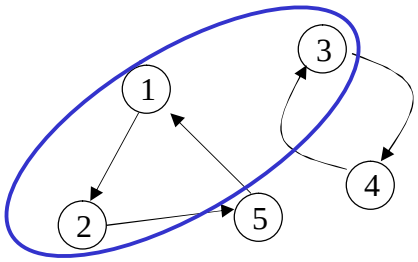
Esempio Soluzione: $x_{12} = x_{25} = x_{51} = x_{34} = x_{43} = 1$

$N = \{1,2,3,4,5\}$



Tali soluzioni non sono ammissibili per il problema TSP ed è necessario introdurre i vincoli (*c*) **di eliminazione dei sottocicli**. Essi impongono che ogni sottoinsieme proprio *S* di *N*, con almeno due elementi, non contenga cicli.

$$(c) \quad \sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \quad \forall S \subseteq N, 2 \leq |S| \leq n-1$$



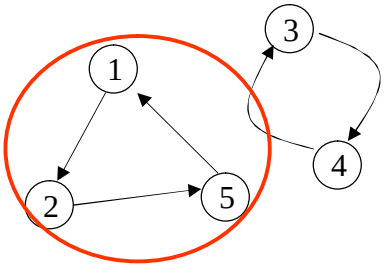
Consideriamo ad esempio la disuguaglianza associata all'insieme $S=\{1,2,3\}$
La disuguaglianza ci dice che *S* non deve contenere cicli al suo interno

$$x_{12} + x_{13} + x_{21} + x_{23} + x_{31} + x_{32} \leq 3 - 1 \quad S = \{1,2,3\}$$

$$1 + 0 + 0 + 0 + 0 + 0 \leq 2$$

disuguaglianza sodisfatta

$$(c) \quad \sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \quad \forall S \subseteq N, 2 \leq |S| \leq n-1$$



Consideriamo ad esempio la disuguaglianza associata all'insieme $S=\{1,2,5\}$
La disuguaglianza ci dice che *S* non deve contenere cicli al suo interno

$$x_{12} + x_{15} + x_{21} + x_{25} + x_{51} + x_{52} \leq 3 - 1 \quad S = \{1,2,5\}$$

$$1 + 0 + 0 + 1 + 1 + 0 > 2$$

disuguaglianza violata

Quanti sono i vincoli di **eliminazione dei sottocicli** ?

Sono uno per ciascun sottoinsieme S dell'insieme N ad eccezione dell'insieme vuoto, degli insiemi con un solo elemento, e dell'insieme N (che deve contenere un ciclo).

Quanti sono i sottoinsiemi di un insieme di n elementi?

Sono 2^n

Quindi i vincoli di eliminazione dei sottocicli sono

$$2^n - n - 2$$

Essi sono in numero **esponenziale** rispetto alla dimensione del problema.

Questo significa che non è possibile inserire esplicitamente tutti i vincoli del modello nel calcolatore se non per istanze con n piccolo.

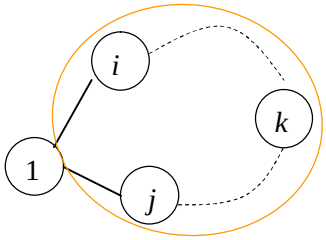
Occorre adottare tecniche opportune per gestire questa difficoltà

Il problema TSP simmetrico

Nel caso in cui si ipotizza che la matrice dei costi c_{ij} sia simmetrica, cioè andare dalla città i alla città j ha lo stesso costo che andare dalla città j alla città i, per ogni coppia di città, è possibile fornire una differente formulazione del problema PCV.

Essa si basa sulla seguente osservazione:

ogni ciclo, cioè ogni soluzione ammisibile, consiste di due lati adiacenti alla città 1 e di un cammino semplice attraverso le città $\{2,3,\ldots,n\}$



Formuliamolo ora come problema di ottimizzazione su un grafo non orientato *G=(N,E)*, con $|N| = n$, e $c_e = c_{ij}$ se il lato $e = (i, j) \in E$.

Variabili=Decisioni: quali lati del grafo fanno parte della soluzione.

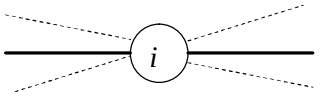
$$x_e = \begin{cases} 1 & \text{se il lato } e \in E \text{ fa parte del ciclo} \\ 0 & \text{altrimenti} \end{cases}$$

Modello

Insieme dei lati con entrambi gli estremi in *S*

$$\min \sum_{e \in E} c_e x_e$$

$$(a) \quad \sum_{e \in d(i)} x_e = 2 \quad \forall i \in N$$



$$(b) \quad \sum_{e \in E(S)} x_e \le |S| - 1 \quad \forall S \subseteq N, 2 \le |S| \le n-1$$

vincoli di eliminazione dei sottocicli

$$x_e \in \{0,1\} \quad e \in E$$

Un esempio di applicazione: assemblaggio di circuiti elettronici

La fase di assemblaggio di circuiti stampati richiede di posizionare su una piastra, in posizioni specifiche, *n* componenti (condensatori, resistenze, ecc.). Tale compito viene svolto dal braccio mobile di un robot, che partendo da una posizione iniziale passa per le *n* posizioni specifiche, esegue in ciascuna il fissaggio, e ritorna alla posizione iniziale.

Il costo c_{ij} è dato dal tempo impiegato dal braccio per passare dalla posizione *i* a quella *j*.

La sequenza con cui vengono fissate le componenti concorre a determinare il tempo di assemblaggio per ciascuna piastra

Problema di copertura con insiemi (*set covering*)

Sono dati un insieme $M=\{1,2,\ldots,m\}$ ed una famiglia di n suoi sottinsiemi $S_j \subseteq M$, con $j \in N = \{1,\ldots,n\}$. Ad ogni sottinsieme S_j è associato un costo c_j.

Problema: trovare un insieme $T \subseteq N$ tale che $\bigcup_{j \in T} S_j = M$, cioè l'unione dei sottoinsiemi scelti copre tutti gli elementi di M

Obiettivo: minimizzare il costo totale dei sottoinsiemi scelti

Esempio di applicazione: apertura di centri di emergenza

E' dato un insieme M di m regioni nelle quali attivare alcuni servizi di emergenza. E' stato individuato un insieme N di n possibili destinazioni per i centri.

Per ciascuna potenziale destinazione $j \in N = \{1,\ldots,n\}$ conosciamo il costo c_j di apertura di un centro e quali regioni vi afferirebbero, cioè S_j.

Dove aprire i centri in modo che tutte le regioni possano afferire ad almeno uno di essi minimizzando il costo totale di apertura?

Variabili=Decisioni: in quali siti aprire i centri

$$x_j = \begin{cases} 1 & \text{se il sito } j \text{ ospita un centro} \\ 0 & \text{altrimenti} \end{cases}$$

Indichiamo con $A=[m \times n]$ una matrice 0-1 di incidenza, cioè tale che

$$a_{ij} = \begin{cases} 1 & \text{se } i \in S_j \\ 0 & \text{altrimenti} \end{cases}$$

Si noti che i valori di A sono coefficienti e non variabili

Modello:

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

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad i = 1,\ldots,m$$

$$x_j \in \{0,1\} \quad j = 1,\ldots,n$$

Esempio numerico di applicazione

$M=\{1,2,\ldots,5\}$, $N=\{1,2,\ldots,6\}$

$S_1=\{3,5\}$ $S_2=\{1,3,5\}$ $S_3=\{1,2,5\}$ $S_4=\{1,2,4\}$ $S_5=\{1,4,5\}$ $S_6=\{3,4\}$

$\underline{c}^T = (4,6,10,14,5,6)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Matrice di incidenza

Soluzione ottima $x^* = (0,0,1,0,0,1)$

$$S_3 \cup S_6 = M$$

Costo = 10 + 6 = 16

Che cosa accomuna questi problemi e altri di precedenti lezioni?

Il problema dello zaino (Knapsack)

Il problema dell'assegnamento

Il problema di sequenziamento ottimale

Il problema del commesso viaggiatore (TSP) (simmetrico e non)

Il problema di copertura con insiemi (*set covering*)

Ad eccezione del problema dell'assegnamento, sono tutti problemi ***NP-difficili***: non conosciamo alcun algoritmo che possa risolvere in modo efficiente qualsiasi loro istanza

Sono tutti problemi che ammettono un numero ***esponenziale*** ma ***finito*** di soluzioni ammissibili: sono problemi di natura ***combinatorica***

In linea di principio possono tutti essere risolti ***enumerando*** tutte le loro soluzioni, valutandone il costo e scegliendo la migliore

Tale approccio si rivela impraticabile:

(a) Risoluzione esatta: ⇨ ***Branch & Bound***, metodi poliedrali, ecc…

(b) Risoluzione approssimata: metodi euristici (***greedy, Ricerca Locale***, ecc…)

Motivazioni

• trovare la soluzione ottima di un problema NP-difficile nei casi pratici può essere troppo oneroso (es. problemi di grandi dimensioni)

• i parametri che descrivono il problema possono essere soggetti a errore (es. applicazioni reali): da *ottimo* a *buono*

• l'ambito di applicazione richiede di fornire soluzioni in tempi *strettissimi* (es. ottimizzazione *in tempo reale*) (situazione critica anche per problemi facili ma con complessità, ad es., del tipo $O(n^3)$)

Algoritmo euristico (*dal greco scoprire****)***: metodo che fornisce una soluzione ammissibile, non necessariamente ottima, di un problema

• Euristiche *con garanzia di approssimazione*

valutazione nel **caso peggiore** (e in alcuni casi nel *caso medio*, più difficile e meno frequente)

• Euristiche *senza garanzia di approssimazione*

Sviluppare una euristica è un problema creativo:
elementi *teorici* e di *buon senso*

Tecniche:

- costruttive (di tipo greedy)
- evolutive (basate su ricerca locale)
- basate sull'enumerazione implicita
- basate sui rilassamenti
- deterministiche / casualizzate

Algoritmi *greedy* (euristiche costruttive)

Idea base: la costruzione della soluzione avviene per passi e ad ogni passo viene fatta la scelta più favorevole, compatibile con i vincoli (greedy = ingordo)

```
greedy (input: E; output: S);
begin   S:=∅;
        while E ≠ ∅ do
            e:=elemento di E che dà il miglior valore di S ∪ {e};
            E := E – {e};
            if  S ∪ {e} è ammissibile then S:= S ∪ {e};
        endwhile
        return S;
end;
```

questi passi devono essere efficienti

Osservazioni

Estrema diffusione

Di semplice implementazione

Estrema rapidità di esecuzione

Quando le soluzioni ammissibili di un problema presentano una particolare struttura (***matroide***) l'euristica greedy trova sempre la soluzione ottima: es. algoritmo di Kruskal per il problema dell'albero di copertura di costo minimo

Sono il punto di partenza più frequente per l'utilizzo degli algoritmi basati su *ricerca locale*

Esempio — Il problema del commesso viaggiatore (TSP)

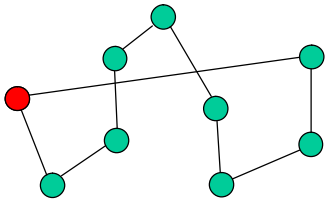
Euristica *Nearest Neighbourhood*

1. scegli un nodo di partenza *p* e marcalo

2. ripeti (*n*–1) volte:

   collega l'ultimo nodo marcato con il nodo non marcato a lui più vicino

3. collega l'ultimo nodo marcato con il nodo *p*



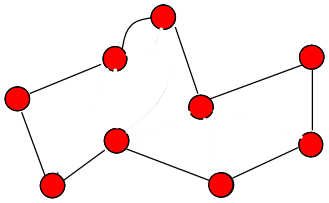
Esempio

Euristica *Nearest Insertion*

1. scegli due nodi e costruisci un ciclo parziale

2. ripeti (n–2) volte:

   inserisci nel ciclo parziale il nodo più vicino a uno di quelli esistenti;



Algoritmi di Ricerca Locale

Obiettivo: ottenere (rapidamente) buone soluzioni ammissibili per problemi di programmazione a numeri interi

Idea: migliorare attraverso una procedura iterativa le soluzioni ottenute applicando le tecniche costruttive (ottenute ad es. con algoritmi di tipo *greedy*)

Tecnica:

(a) individuare modifiche (perturbazioni) alla struttura delle soluzioni ammissibili che ne preservino l'ammissibilità

(b) applicare le modifiche fintantoché il valore della funzione obiettivo migliora

Consideriamo il generico problema

$$\min_{s \in S} f(s)$$

dove S rappresenta l'insieme finito delle soluzioni

ed $f(s)$ è la funzione obiettivo da ottimizzare

Definizione

Chiamiamo *mossa m* da una soluzione ad un'altra un operatore del tipo

$$m: S \longrightarrow S$$

Data una soluzione ammissibile $s \in S$ l'operatore mossa m applicato ad s restituisce una soluzione ammissibile $m(s) \in S$

L'intorno

In generale si utilizzano mosse la cui applicazione non altera in modo eccessivo la struttura di una soluzione

cioè si preferisce che le soluzioni $m(s)$ siano *vicine* alla soluzione s

Definizione

Chiamiamo *intorno* di una soluzione s l'insieme

$$N(s) = \{\bar{s} \in S \mid \exists m : \bar{s} = m(s)\}$$

$N(s)$ è l'insieme di tutte le soluzioni ammissibili che si possono ottenere applicando ad s tutte le *mosse* possibili

Esempio:

si consideri la seguente sequenza s=(c,a,d,e,b) corrispondente alla soluzione di un problema di ordinamento di oggetti (o attività).

Consideriamo le mosse che scambiano di posizione una coppia di oggetti.

L'intorno di s diventa

$N(s)$={(a,c,d,e,b); (d,a,c,e,b); (e,a,d,c,b); (b,a,d,e,c);

(c,d,a,e,b); (c,e,d,a,b); (c,b,d,e,a);

(c,a,e,d,b); (c,a,b,e,d);

(c,a,d,b,e) }

Un modo alternativo di definire l'intorno di una soluzione è quello di definire una *metrica* nell'insieme S, introdurre cioè una nozione di ***distanza*** fra le soluzioni di S

Definizione

Chiamiamo *l-intorno* di una soluzione s l'insieme

$$N_l(s) = \{\bar{s} \in S \,|\, d(\bar{s}, s) \leq l\}$$

dove $d : S \times S \rightarrow \Re^+$ definisce una misura di *distanza* in S

$N_l(s)$ è l'insieme di tutte le soluzioni ammissibili che si trovano ad una distanza al più pari a l dalla soluzione s

Quando possiamo rappresentare le soluzioni di un problema mediante vettori booleani ad *n* componenti si può utilizzare la distanza di Hamming

Definizione

Si chiama distanza di Hamming $d_H(s_1,s_2)$ fra due vettori booleani ad *n* componenti s_1 ed s_2 il numero delle componenti in cui essi differiscono

Esempio: s_1=(0 0 0 1 1 0 1 0 0 1) e

s_2=(1 0 1 1 0 0 1 0 1 1)

hanno distanza di Hamming $d_H(s_1,s_2)$ pari a 4

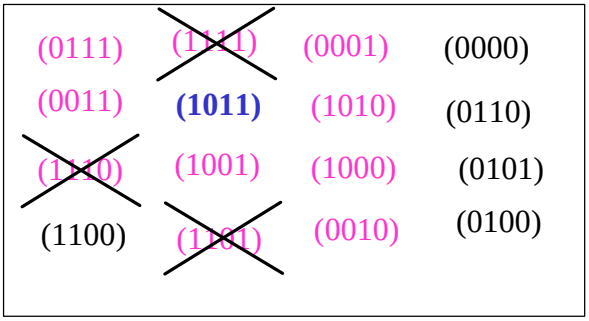
Consideriamo una soluzione ammissibile *s*=(1,0,11) di un problema dello zaino con 4 oggetti

l'intorno $N_1(s) = \{\bar{s} \in S \,|\, d_H(\bar{s}, s) \leq 1\}$
è dato da tutte le soluzioni ammissibili che differiscono da *s* in al più una componente

(0111)	~~(1111)~~	(0001)	(0000)
(0011)	**(1011)**	(1010)	(0110)
(1110)	(1001)	(1000)	(0101)
(1100)	(1101)	(0010)	(0100)

Consideriamo una soluzione ammissibile s=(1,0,11) di un problema dello zaino con 4 oggetti

l'intorno $N_2(s) = \{\bar{s} \in S \,|\, d_H(\bar{s}, s) \leq 2\}$
è dato da tutte le soluzioni ammissibili che differiscono da s in al più due componenti



L'algoritmo

Partendo dalla definizione di *intorno* delle soluzioni possiamo introdurre l'algoritmo ***Ricerca_Locale***

```
Ricerca_Locale(s);
begin   s*:=s; fine=false;
        repeat
            s:=miglior soluzione in N(s)
            if  f(s) < f(s*) then
                s*:= s;
            else fine=true;
        until  not fine;
        return s*;
end;
```

Problema di minimo

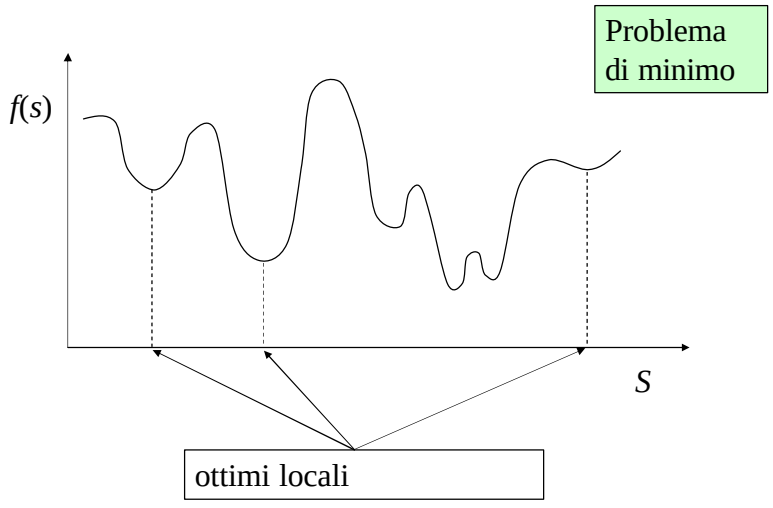
Ottimi locali

Che tipo di soluzioni fornisce un algoritmo di ricerca locale?

Definizione

Una soluzione $s \in S$ è detta ***ottimo locale***, rispetto all'intorno $N(s)$, se vale la relazione $f(s) \leq f(\bar{s}), \forall \bar{s} \in N(s)$

Le soluzioni individuate da un algoritmo di ricerca locale sono ottimi locali rispetto all'intorno adottato



Ottimi globali

Che relazione esiste fra le soluzioni fornite da un algoritmo di ricerca locale e la soluzione ottima?

Proprietà

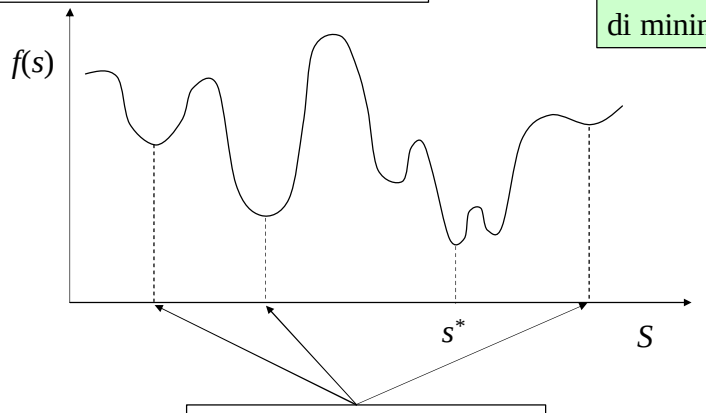
Se S è un insieme convesso ed $f(s)$ è una funzione convessa allora ogni *ottimo locale* è anche ***ottimo globale*** (in un problema di minimizzazione)

Le soluzioni individuate da un algoritmo di ricerca locale sono ottimi globali solo per problemi molto particolari.

si veda ad esempio l'algoritmo del simplesso e la programmazione lineare

In generale quindi le soluzioni ottenute applicando la sola ricerca locale non sono soluzioni ottime

Problema di minimo



ottimi locali (non globali)

Come migliorare le soluzioni ottenute applicando la sola ricerca locale (visto che spesso non sono soluzioni ottime) ?

L'***idea***: accettare di eseguire anche mosse di tipo peggiorante

Il ***problema***: impedire che l'algoritmo entri in ciclo

Le ***soluzioni***:

a) introdurre delle scelte *casuali:* es. Simulated Annealing

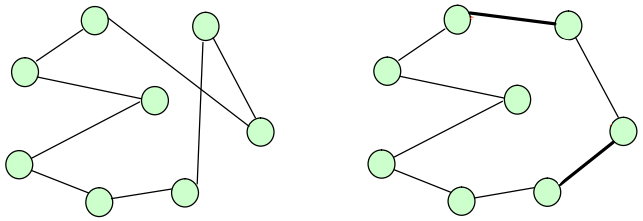
b) memorizzare tutto o parte delle soluzioni generate:

es. Tabu Search

Esempi di *intorno*

Il problema del commesso viaggiatore (TSP) simmetrico

L'intorno 2-opt

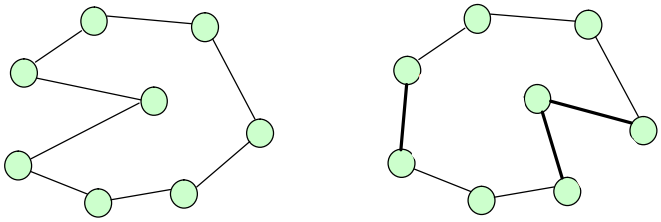


La *mossa* è lo scambio di una coppia di lati con un'altra coppia

N.B. dopo l'eliminazione di una coppia di lati la scelta dei lati da inserire è univoca

Il problema del commesso viaggiatore (TSP) simmetrico

L'intorno 3-opt



La *mossa* è lo scambio di una terna di lati con un'altra terna

N.B. dopo l'eliminazione di una terna di lati la scelta dei lati da inserire non è univoca: quante alternative ci sono?

Il problema del commesso viaggiatore (TSP) simmetrico

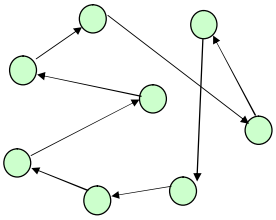
Osservazioni:

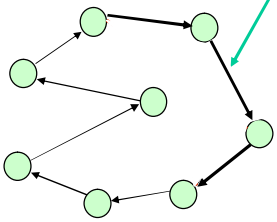
L'intorno 2-opt contiene un ordine di $O(n^2)$ soluzioni, una per ciascuna coppia di lati che viene eliminata

L'intorno 3-opt genera, in media, soluzioni di costo inferiore a quelle del 2-opt, ma ha una costo computazionale maggiore: esso contiene un ordine di $O(n^3)$ soluzioni, tre per ciascuna terna di lati che viene eliminata

Il problema del commesso viaggiatore (TSP) asimmetrico

L'intorno 2-opt

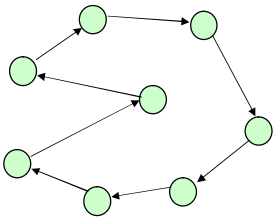


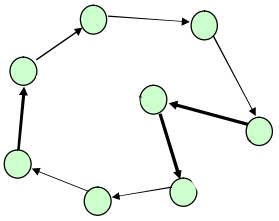


La *mossa* è lo ***scambio*** di una coppia di archi con un'altra coppia e l'***inversione*** degli archi di un tratto del percorso corrente

Il problema del commesso viaggiatore (TSP) asimmetrico

L'intorno 3-opt





La *mossa* è lo scambio di una terna di archi con un'altra terna con l'accortezza di mantenere l'orientazione prevalente

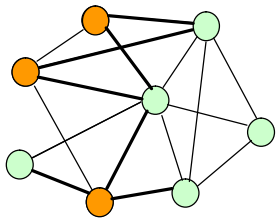
Il problema del taglio di costo massimo

È dato un grafo non orientato $G=(N,E)$ con un costo non negativo c_e associato ad ogni lato e di E

Ricordiamo che si definisce *taglio indotto dall'insieme di nodi S* l'insieme $\delta(S)$ dei lati che hanno un estremo in S ed uno in $N\backslash S$

Si deve determinare il taglio di costo massimo

Esempio con $c_e=1$ per ogni lato e



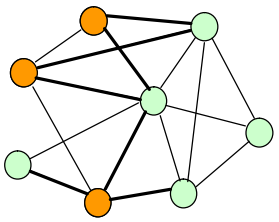
S contiene i nodi

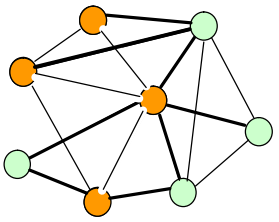
$N\backslash S$ contiene i nodi

Il taglio $\delta(S)$ è formato da 7 lati

Il problema del taglio di costo massimo

L'intorno *spostamento*



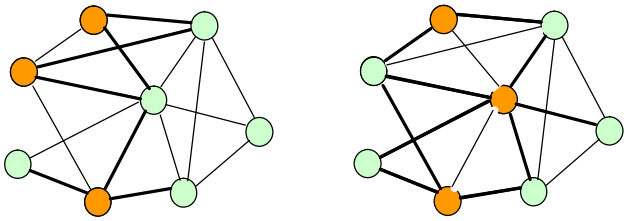


La *mossa* è lo spostamento di un nodo di S in $N\backslash S$ o viceversa

Nella nuova soluzione il taglio $\delta(S)$ è formato da 8 lati

Il problema del taglio di costo massimo

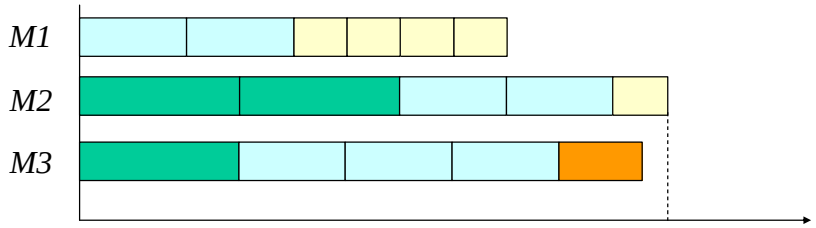
L'intorno *scambio*



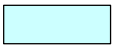
La *mossa* è lo scambio di un nodo di *S* con uno di *N/S*

Nella nuova soluzione il taglio $\Omega(S)$ è formato da 10 lati

Il problema di assegnamento e sequenziamento



Durata 15



Durata 10

Durata 8

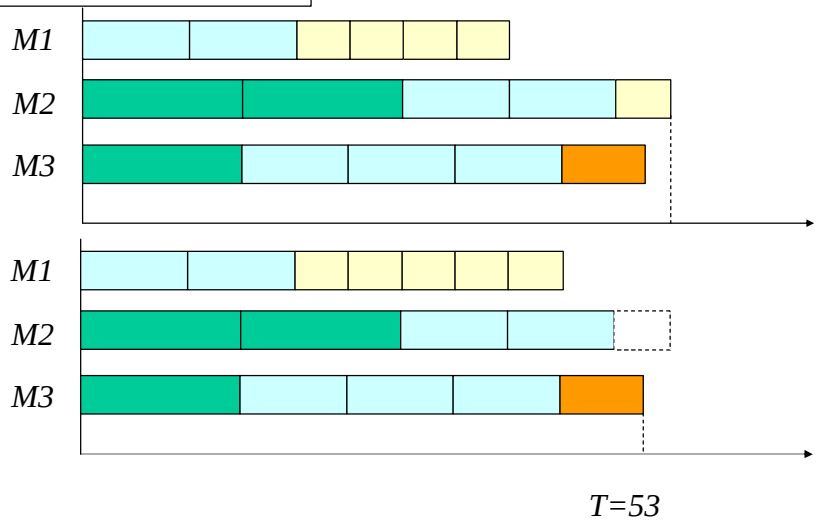
Durata 5

T=55

Costo della soluzione

Il problema di assegnamento e sequenziamento

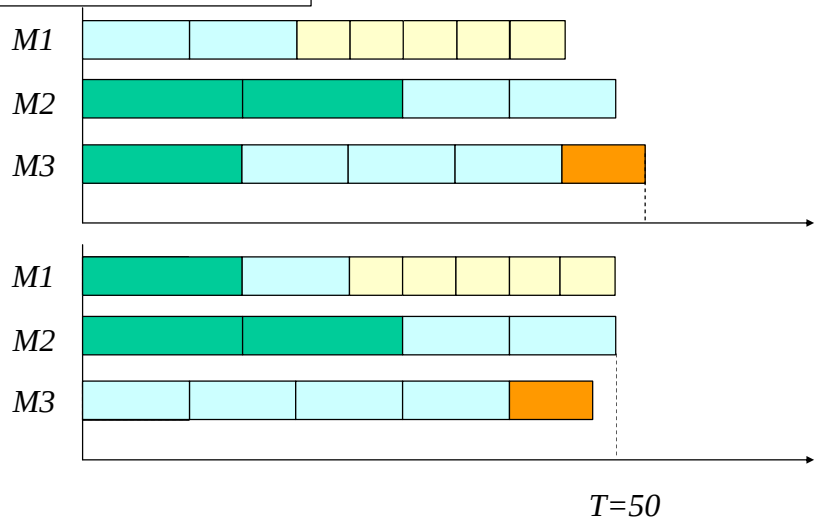
L'intorno *spostamento*



T=53

Il problema di assegnamento e sequenziamento

L'intorno *scambio*



T=50

Euristiche *con garanzia di approssimazione*

Una richiesta in più:

- desideriamo un algoritmo euristico ed
- una indicazione del massimo errore commesso

Dato un problema di ottimizzazione P, dove denotiamo con con z_{opt} il valore della soluzione ottima e con z_A il valore fornito dall'algoritmo euristico A, chiamiamo

- errore assoluto $E_A = |z_{opt} - z_A|$
- errore relativo $R_A = |z_{opt} - z_A| / |z_{opt}|$

N.B. Ipotizziamo $z_{opt} \neq 0$. Nei casi critici si effettua una opportuna modifica all'istanza.

Indichiamo con *I* una istanza (cioè un caso particolare) del problema P

Un algoritmo A è un algoritmo *assolutamente* approssimato per un problema P se e solo se per ogni *I*

$$|z_{opt}(I) - z_A(I)| \leq k$$

per una certa costante $k > 0$

Un algoritmo A è un algoritmo $g(n)$- approssimato per un problema P se e solo se per ogni *I* di dimensione *n*

$$|z_{opt}(I) - z_A(I)| \leq g(n)|z_{opt}(I)|$$

Un algoritmo A è un algoritmo *e*- approssimato per un problema P se e solo se per ogni *I*

$$|z_{opt}(I) - z_A(I)| \leq e|z_{opt}(I)|$$

per una certa costante $e > 0$

Esempio

Problema di copertura con insiemi (*set covering*)

Sono dati un insieme $M=\{1,2,\ldots,m\}$ ed una famiglia di n suoi sottoinsiemi $S_j \subseteq M$, con $j \in N=\{1,2\ldots,n\}$. Ad ogni sottoinsieme S_j è associato un costo c_j. Si cerca l'insieme di sottoinsiemi di costo minimo la cui unione copre tutti gli elementi di M.

Algoritmo *greedy*

1. ordina in L i sottoinsiemi per valori non decrescenti del rapporto "costo su numero di elementi scoperti che essi coprono"

2. ripeti fino a ché tutti gli elementi di M sono coperti

   togli da L il prossimo sottoinsieme, S, nell'ordine;

   etichetta come coperti gli elementi scoperti contenuti in S;

   aggiorna l'ordinamento di L;

Esempio numerico

$M=\{1,2,\ldots,5\}$, $N=\{1,2,\ldots,6\}$ $\underline{c}^T=(4,6,10,14,5,6)$

$S_1=\{3,5\}$ $S_2=\{1,3,5\}$ $S_3=\{1,2,5\}$ $S_4=\{1,2,4\}$ $S_5=\{1,4,5\}$ $S_6=\{3,4\}$

Passo 1: Rapporti = (4/2, 6/3, 10/3,14/3, 5/3,6/2); Scegli S_5;

Passo 2: Rapporti = (4/1, 6/1, 10/1,14/1, -- ,6/1); Scegli S_1;

Passo 3: Rapporti = (-- , ∞ , 10/1,14/1, -- , ∞); Scegli S_3;

$$A=\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Matrice di incidenza

$S_5 \cup S_1 \cup S_3 = M$ $z_A = 19$

$S_3 \cup S_6 = M$ $z^* = 16$

Indicando con $k = \max_j \{ |S_j| \}$ si può dimostrare che

L'algoritmo è $\log(k)$-approssimato

Come si ricavano i limiti di approssimazione degli algoritmi approssimati?

Come vedremo negli esempi occorrono tre componenti (consideriamo ad es. problemi di minimo) :

• Una stima per eccesso (upper bound) z_A del valore della soluzione ottima, ottenuta dall'euristica

• Una stima per difetto (lower bound) z_{LB} del valore della soluzione ottima, ottenuta ad es. con un rilassamento

• Una funzione $f(z_{LB})$ non decrescente il cui valore sia non inferiore a z_A in modo da ottenere una relazione del tipo

$$z_{LB} \le z^* \le z_A \le f(z_{LB}) \le f(z^*)$$

Esempio

Il problema dello zaino intero

Consideriamo il problema

$$z^* = \max \{c^T x : a\,x \le b,\ x \in Z^n_+\}$$

dove $b, a_1, \ldots, a_n \in Z_+$ e per *ipotesi* $a_j \le b$, con $j = 1,\ldots,n$ e vale la relazione $c_1/a_1 \ge c_j/a_j$ per $j = 2,\ldots,n$

Algoritmo *greedy*

1. riempi lo zaino con il maggior numero possibile di copie dell'oggetto che ha il miglior rapporto "costo su ingombro"

Consideriamo la soluzione di tipo *greedy* $x^H = (\lfloor \frac{b}{a_1} \rfloor, 0, \ldots, 0)$

di valore $z^H = c_1 \lfloor \frac{b}{a_1} \rfloor \leq z^*$

La soluzione del rilassamento lineare fornisce un limite superiore $z^{LP} = c_1 \, b / a_1 \geq z^*$

Da $a_1 \leq b$ segue $\lfloor \frac{b}{a_1} \rfloor \geq 1$. Ponendo $\frac{b}{a_1} = \lfloor \frac{b}{a_1} \rfloor + f$, con $0 \leq f < 1$

si ricava $\lfloor \frac{b}{a_1} \rfloor \Big/ \frac{b}{a_1} = \frac{\lfloor \frac{b}{a_1} \rfloor}{\lfloor \frac{b}{a_1} \rfloor + f} \geq \frac{\lfloor \frac{b}{a_1} \rfloor}{\lfloor \frac{b}{a_1} \rfloor + \lfloor \frac{b}{a_1} \rfloor} = \frac{1}{2}$

Quindi $z^H / z^* \geq z^H / z^{LP} = \frac{c_1 \lfloor \frac{b}{a_1} \rfloor}{c_1 \frac{b}{a_1}} = \frac{\lfloor \frac{b}{a_1} \rfloor}{\frac{b}{a_1}} \geq \frac{1}{2}$.

L'algoritmo è 1-approssimato

Esempio numerico

c = (20, 27, 9, 24, 6)

a = (5, 7, 4, 9, 3) b=22

$x^H = (\lfloor 22/5 \rfloor, 0,0,0,0) = (4,0,0,0,0)$ $z^H = 80$ $z^{LP} = 88$

$z^H / z^{LP} = 80/88 \approx 0{,}909$

$x^* = (0,3,0,0,0)$ $z^* = 81$

$z^H / z^* = 80/81 \approx 0{,}99$

$z^H / z^* \geq z^H / z^{LP} \geq 0{,}5$

In questo caso l'errore commesso è di circa l'1,2 %.
Per nessuna istanza sarà superiore al 100%.

Esempio

Problema di assegnamento e sequenziamento

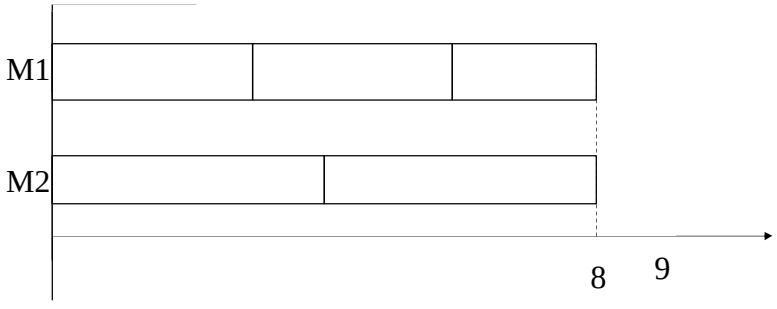
Ci sono m macchine identiche ed n lavorazioni. Ogni lavorazione j, con $j=1,\ldots,n$, richiede di essere processata da una qualsiasi delle m macchine per un tempo di processamento ininterrotto p_j. Ogni macchina processa una sola lavorazione alla volta. Vogliamo minimizzare l'istante di completamento z^* di tutte le lavorazioni

Algoritmo *greedy* generico

1. Ordina le lavorazioni in un ordine qualsiasi

2. Attribuisci le lavorazioni alle macchine nell'ordine dato assegnando ciascuna lavorazione alla macchina più scarica

Esempio numerico



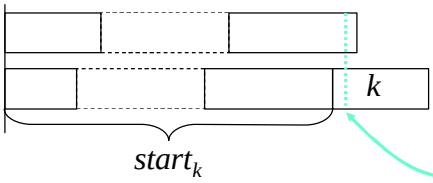
Sia z_A il valore della soluzione fornita dall'algortimo *greedy* generico.
Valutiamo l'errore relativo nel caso di $m=2$

Le quantità $LB = \frac{1}{2}\sum_{j=1}^{n} p_j$ e $\max_j \{p_j\}$ sono stime per difetto di z^*

Vale quindi la relazione $LB \le z^* \le z_A$

Sia k l'indice dell'ultima lavorazione eseguita e sia $start_k$ il suo istante di inizio.
Quindi $z_A = start_k + p_k$

Poiché le lavorazioni vengono assegnate alla macchina meno carica ne deriva che quando k è stata assegnata l'altra macchina era occupata almeno fino all'istante $start_k$



Quindi

$$start_k \le \frac{1}{2}\sum_{\substack{j=1 \\ j\neq k}}^{n} p_j = LB - p_k/2$$

Si ha
$z_A = start_k + p_k \le LB + p_k/2 \le z^* + z^*/2 = 3/2\, z^*$

L'algoritmo è 1/2 -approssimato

Esempio

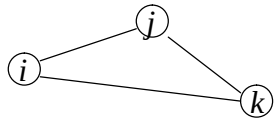
Il problema del commesso viaggiatore (TSP) simmetrico

Dato un grafo non orientato completo $G=(N,E)$ con un costo non negativo c_e per ciascun lato $e=(i,j)$ di E, si determini il ciclo Hamiltoniano di costo minimo

Il problema è NP-difficile, ma nell'ipotesi che in G valga la *disuguaglianza triangolare* è possibile fornire degli algoritmi e -approssimati

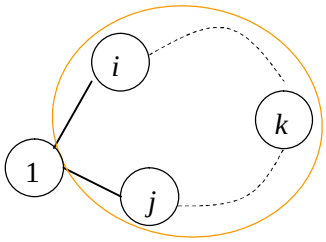
Disuguaglianza triangolare:

Per ogni terna di nodi i,j,k in N vale la relazione $c_{ij} + c_{jk} \ge c_{ik}$

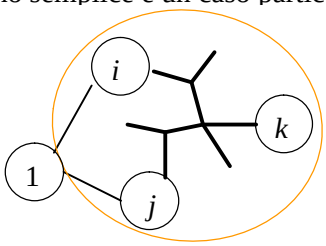


Esempio : il TSP simmetrico

Come già illustrato ogni soluzione ammisibile consiste di due lati adiacenti al nodo 1 e di un cammino semplice attraverso i rimanenti nodi {2,3,…,*n*}



Ma un cammino semplice è un caso particolare di albero



Definizione
Un ***1-albero*** è un sottografo formato da due lati adiacenti al nodo 1 e da un albero sui nodi rimanenti {2,3,…,*n*}

Ogni ciclo è un 1-albero e quindi il problema di trovare l'1-albero di costo minimo è un rilassamento del TSP simmetrico

Come si risolve il problema dell'1-albero di costo minimo?

1. si individua l'albero di costo minimo che tocca i nodi {2,3,…,*n*}

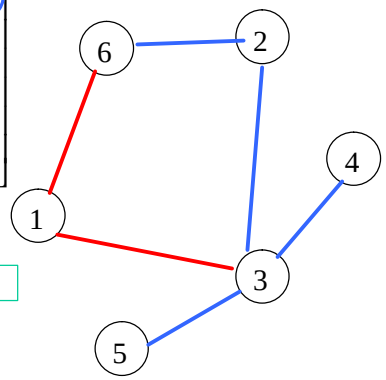
2. si aggiungono i due lati di costo minimo incidenti nel nodo 1

N.B. per eseguire il passo 1 si può adottare l'algoritmo di Prim o quello di Kruskal

Esempio numerico:

Si consideri la seguente matrice di costi

$$c_e = \begin{bmatrix} - & 30 & 26 & 50 & 40 & 20 \\ - & - & 24 & 40 & 50 & 18 \\ - & - & - & 24 & 26 & 30 \\ - & - & - & - & 30 & 28 \\ - & - & - & - & - & 35 \\ - & - & - & - & - & - \end{bmatrix}$$



1-albero ottimo di costo 138

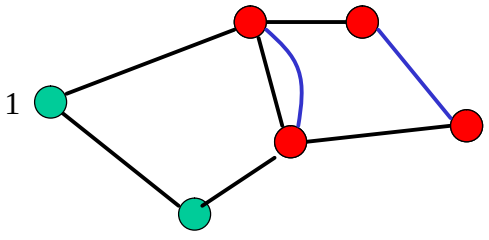
Algoritmo di Christofides

1. costruiamo $1T^*$, l'1-albero di costo minimo in G

Osservazione: ogni ciclo Hamiltoniano è un 1-albero di costo non inferiore a $c(1T^*)$: $c(1T^*) \leq c(H^*)$

2. troviamo D, l'insieme di nodi di G con *grado* dispari in $1T^*$

3. costruiamo un accoppiamento perfetto di lunghezza minima, M^* , fra i nodi di D. ($M \acute{I} E$ *tale che ogni nodo di* D *è toccato da esattamente un lato di* M)



N.B. $|D|$ è pari e quindi M^* esiste sempre

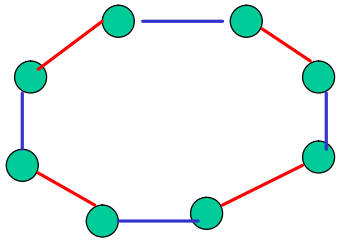
Osservazioni:

Il ciclo Hamiltoniano ottimo (con n pari) si può scomporre in due accoppiamenti perfetti, M1 ed M2, e vale

$$c(\mathrm{M1}) + c(\mathrm{M2}) = c(H^*)$$

Poiché $2c(\mathrm{M}^*) \le c(\mathrm{M1}) + c(\mathrm{M2})$ vale la relazione $c(\mathrm{M}^*) \le 1/2\ c(H^*)$

N.B. la relazione vale anche con n dispari e per ogni sottoinsieme dei nodi (con qualche passaggio in più sfruttando la disuguaglianza triangolare)

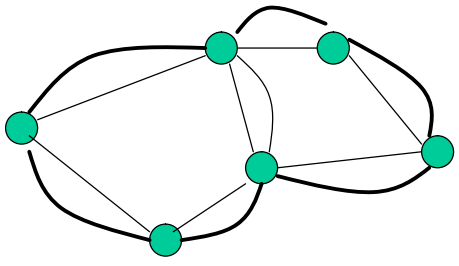


Osservazione: il grafo formato dall'1-albero ottimo e dall'accoppiamento perfetto è *euleriano*, ossia contiene un ciclo C che visita tutti i *lati* una ed una sola volta.

4. costruiamo un ciclo euleriano C

5. trasformiamo il ciclo euleriano C in un ciclo Hamiltoniano CH di costo non superiore (*per la disuguaglianza triangolare*): $c(CH) \le c(C)$

Quindi $c(CH) \le c(\mathrm{M}^*) + c(1T^*) \le c(H^*) + 1/2\ c(H^*) = 3/2\ c(H^*)$



L'algoritmo è

1/2 -approssimato