Università degli Studi di Padova
Facoltà di Ingegneria
Dipartimento di Tecnica e Gestione dei Sistemi Industriali

Tesi di Laurea di Primo Livello

LOCALIZZAZIONE DEI SERVIZI ED IL PROGETTO DI RETI

Relatore: Ch.mo Professor Giorgio Romanin Jacur

Laureanda: Valentina Zieger

Indice

Introduzione

Questa tesi tratta del problema della localizzazione dei servizi ed il progetto di reti, in particolar modo dei principali metodi euristici e non, che sono stati studiati nel tempo per poter risolvere il problema.

La localizzazione dei servizi consiste nel determinare, tramite opportuni algoritmi o euristiche, la posizione nello spazio di determinate strutture, ottimizzando le distanze, minimizzando i costi, e garantendo l'accessibilità da parte degli utenti. Tale problema ha applicazioni sia in campo industriale, sia nel settore pubblico. Nell'ambito industriale consente di localizzare efficacemente impianti, magazzini, centri di distribuzione, … , ecc. con lo scopo magari di minimizzare i costi di trasporto delle merci o dei prodotti, oppure di collocare i magazzini in modo di poter servire al meglio e con meno costi i propri rivenditori. Nel settore pubblico svolge un compito fondamentale in quanto consente il posizionamento di strutture pubbliche, quali scuole, ospedali, stazioni di polizia, caserme dei pompieri, uffici,…; in modo da essere facilmente raggiungibili dalla maggior parte degli utenti che richiedono il servizio.

Il progetto di reti invece affronta un contesto più ampio in quanto si tratta di definire vari elementi in modo da collegare efficacemente i vari punti della rete, per gestire al meglio la distribuzione di beni o servizi. Applicazioni del progetto di reti sono innumerevoli e toccano vari contesti, si pensi ad esempio alla rete di trasporti, alle reti di distribuzione di servizi come acquedotti e gasdotti, alle reti per i servizi di telecomunicazione (telefonia fissa e mobile, internet,..), ecc.

Nel Capitolo 1 si introdurrà il problema della localizzazione dei servizi ponendo attenzione ai vari elementi che compongono il problema. I Capitoli 2 e 3 sono dedicati all'approfondimento di due importanti metodi per la risoluzione del problema di localizzazione, quali il Simple Plant Location Problem e il P-Median Problem. Il Capitolo 4 invece tratterà alcuni algoritmi meta-euristici quali il Simulated Annealing, la Tabù Search e gli Algoritmi Genetici.

Nei Capitoli 5-6 si parlerà del progetto di reti, delle configurazioni ad albero a costo minimo, con e senza vincoli di capacità, e dei metodi risolutivi ad essi legati.

Nel Capitolo 7 verrà presentato il noto problema del Travelling Salesman Problem (problema del commesso viaggiatore) con un accenno sui metodi finora utilizzati per le sue applicazioni

CAPITOLO 1

Localizzazione di servizi

1.1 Presentazione

Un problema di localizzazione consiste nella ricerca della posizione da assegnare ad un insieme di strutture, denominate *facilities*, in funzione della distribuzione di una domanda, reale o potenziale, relativa alla loro utilizzazione. Le applicazioni possibili sono diverse, possono essere infatti assimilati a servizi da localizzare: stabilimenti e impianti industriali, depositi, magazzini, uffici pubblici, ospedali, scuole, terminali di trasporto, parcheggi etc.

Nel caso di un'azienda, le scelte di localizzazione sono in grado di incidere significativamente sui costi delle attività e sulle capacità di rispondere efficacemente alle esigenze dei clienti.

In generale quando si devono prendere decisioni riguardanti la localizzazione si cerca di minimizzare i costi che cambiano a seconda del sito, e di massimizzare i ricavi e il servizio al cliente.

Possiamo distinguere due tipi di localizzazione. La localizzazione puntuale che consiste nella ricerca di uno o più siti in cui posizionare le facilities entro uno spazio all'interno del quale è definita una distribuzione di domanda. Si parla invece di progetto di reti (network design) quando i servizi presentano un'estensione tale che la rappresentazione attraverso punti risulta poco efficace.

I modelli matematici in grado di descrivere tali problemi possono presentare caratteristiche diverse secondo le ipotesi assunte a base del modello. Nel caso che le localizzazioni vadano scelte all'interno di un insieme infinito di possibilità si parla di *modelli discreti*; in caso contrario, se lo spazio all'interno del quale vanno individuate le localizzazioni è continuo si parla di *modelli continui*.

La definizione del problema implica che la decisione finale da prendere dipende da una valutazione dell'attrattività dei servizi rispetto alla domanda; per tale motivo un aspetto fondamentale è rappresentato dalla misura della distanza tra punti di domanda e potenziali servizi. Tale distanza dipende dalla scelta di metrica adottata.

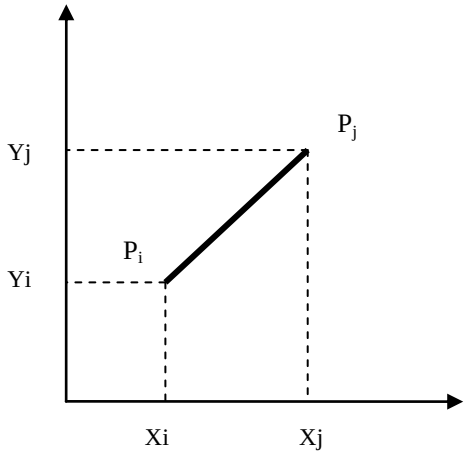
1.2 Distanze metriche:

Dati due punti $P_i=(x_i,y_i)$ e $P_j=(x_j,y_j)$ la distanza tra loro può sempre essere espressa come:

$$d_k(P_i,P_j)=\left((x_i-x_j)^k+(y_i-y_j)^k\right)^{1/k}$$

Per k=1 si parla di metrica lineare o di Manhattan mentre nel caso di k=2 si parla di metrica o distanza euclidea.

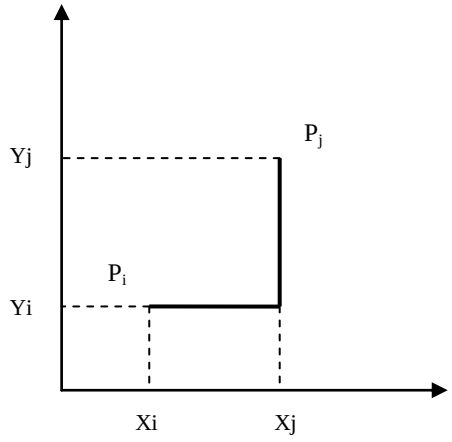
Distanza Euclidea:



$$d(P_i,P_j)=\sqrt{(x_i-x_j)^2+(y_i-y_j)^2}$$

Distanza di Manhattan:

In molti problemi di ubicazione conviene rifarsi a distanze lineari , ovvero a distanze percorse in direzioni x e y ortogonali.



$$d(P_i,P_j)=|x_i-x_j|+|y_i-y_j|$$

Classificazione dei problemi di localizzazione:

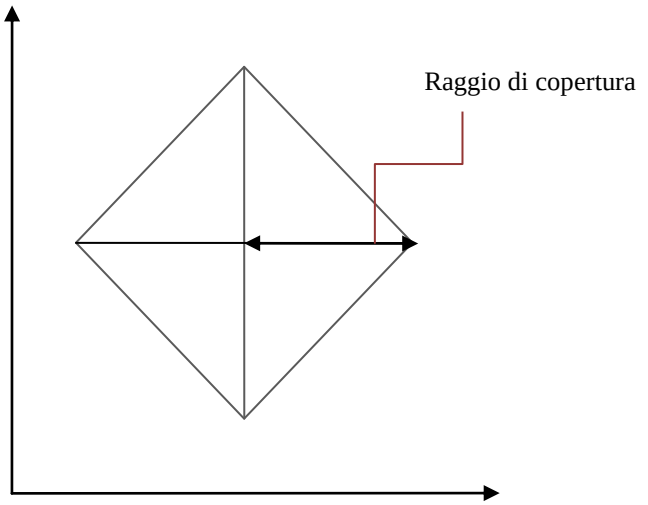
I problemi di localizzazione, come buona parte dei problemi di ottimizzazione, possono essere ricondotti ad analisi costi-benefici.

I benefici derivanti da scelte localizzative sono diversi, ma in generale si può affermare che maggiore è l'utenza che può accedere al servizio, maggiore è il beneficio ottenibile.

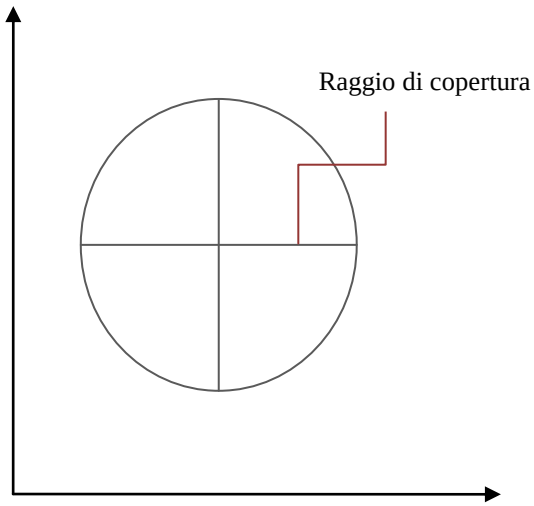
Copertura del servizio:

I criteri più utilizzati per misurare l'accessibilità sono la *distanza totale,* intesa come la somma delle distanze che gli utenti devono coprire per raggiungere il servizio più vicino, e la cosiddetta *copertura* del servizio. Per copertura si intende la superficie che si trova entro una certa distanza, detta raggio di copertura, dal servizio più vicino. La copertura è definita quantitativamente come il numero di utenti che si trovano ad una distanza dal servizio inferiore al raggio.

La copertura assume forme diverse a seconda della metrica utilizzata.



Metrica di Manhattan



Metrica di Euclidea

I costi

Per quanto riguarda i costi, in generale si può distinguere tra costi fissi e variabili. I *costi fissi*, o *costi di localizzazione*, includono tutti gli oneri finanziari e di investimento legati alla localizzazione da effettuare. Essi possono dipendere da diversi fattori come il punto di localizzazione scelto e la capacità del servizio, intesa come la quantità massima di accoglimento della domanda del servizio. I *costi variabili*, o *costi di afferenza*, sono costi che derivano dalla necessità di accedere al servizio; essi possono riferirsi tanto a costi che spettano al gestore del servizio, quanto a costi a carico degli utenti

Secondo Slack et al. (2004) le decisioni di localizzazione devono essere prese cercando di bilanciare tre voci:

Costi che cambiano a seconda della localizzazione delle facilities;

Servizio offerto al cliente;

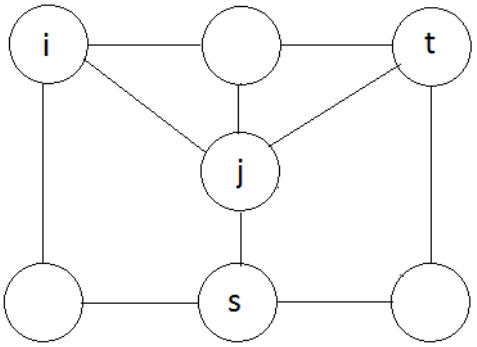
Ricavi potenziali che variano con la localizzazione.

Nei problemi di localizzazione infatti è possibile definire diversi obiettivi quali la minimizzazione dei costi fissi di localizzazione e/o dei costi variabili di afferenza, o la massimizzazione della domanda totale coperta dal servizio. Gli obiettivi possono essere considerati contestualmente, ottimizzando una combinazione lineare di essi, o singolarmente scegliendo un obiettivo di riferimento e imponendo dei vincoli sui valori minimi e/o massimi ammissibili. Oltre ai vincoli relativi al soddisfacimento della domanda, possono esserci vincoli sulla copertura del servizio, sui costi da sostenere, sulla capacità dei servizi, sulla struttura della rete dei servizi.

Successivamente verranno descritti alcuni modelli matematici fondamentali per descrivere i problemi di localizzazione.

A tale scopo viene introdotto il concetto di *grafo pesato*, non orientato. Un grafo viene normalmente rappresentato come $G = (V,A)$ dove si rende esplicito il fatto che questo è definito da due insiemi, quello dei *nodi* V, e quello degli *archi* A, definito come una famiglia di coppie non ordinate di nodi. Un grafo è definito pesato quando ad ogni suo arco viene associato un numero reale, che può indicare, ad esempio, una lunghezza.

Definiamo quindi i seguenti elementi:



$I \subseteq V$ insieme dei nodi generatori o destinazioni di domanda

$J \subseteq V$ insieme delle possibili localizzazioni puntuali

$L \subseteq A$ insieme dei possibili archi da localizzare

$d_i \geq 0$ domanda generata dal nodo $i \in I$

$d^{st} \geq 0$ domanda generata dal nodo s con destinazione nel nodo t

$D = \{d^{st}\}$ matrice delle domande origine-destinazione

$f_{ij} \geq 0$ flusso sull'arco (i, j)

$f_{ij}^{st} \geq 0$ flusso sull'arco (i, j) dovuto alla domanda d^{st}

$c_{ij} \geq 0$ costo di afferenza j della domanda generata dal nodo i

$x_{ij} \geq 0$ frazione di domanda generata da i soddisfatta dal servizio in j

y_i variabile binaria definita per ogni $j \in J$ e pari a 1 se in j è localizzato un servizio, 0 altrimenti

y_{ij} variabile binaria definita per ogni $(i,j) \in L$ e pari a 1 se (i,j) fa parte del progetto, 0 altrimenti

K_j capacità massima associata ad una localizzazione $j \in J$

K_{ij} capacità massima associata a ciascun arco (i, j)

r_j costo di localizzazione o di apertura di un servizio in $j \in J$

r_{ij} costo di localizzazione o di apertura di un arco $(i, j) \in L$

CAPITOLO 2

Simple Plant Location Problem

Il modello Simple Plant Location (SPLP) altrimenti conosciuto come *uncapacitated facility location problem*, è stato introdotto per la prima volta da Balinski nel 1965. Questo modello può essere identificato come uno dei modelli più importanti nella teoria della localizzazione, infatti in letteratura sono stati pubblicati numerosi articoli riguardanti questo problema. Uno dei più famosi metodi risolutivi per il SPLP è stato proposto da Erlenkotter nel 1978, tale metodo è il primo algoritmo *dual ascent* proposto in letteratura.

2.1 SPLP

Il *Simple Plant Location Problem* è un problema di ottimizzazione combinatoria: si devono scegliere alcuni impianti da attivare tra un insieme di possibili candidati per poi allocare ogni cliente, appartenente ad un insieme dato, ad uno degli impianti attivati, in modo tale da minimizzare il costo totale.

E' possibile quindi differenziare i due elementi che caratterizzano il SPLP :

La locazione ossia la scelta di quali impianti verranno aperti;

Allocazione cioè il processo di assegnazione di un cliente ad un dato impianto aperto.

In questo approccio classico si suppone che il soggetto che opera la locazione sia allo stesso tempo anche quello che sviluppa il processo di allocazione; equivalentemente, quando i due soggetti non coincidono, si può ritenere che l'allocatore (ad esempio i clienti come entità singola) riconosca e condivida il criterio di allocazione adottato dal locatore (l'azienda). Ciò accade, ad esempio, quando un'impresa invia i propri prodotti ai clienti da impianti diversi, oppure quando il cliente è obbligato per legge o per contratto a rivolgersi ad un dato impianto. Tuttavia se è il cliente a spostarsi verso l'impianto per ottenere un prodotto o un servizio, allora il cliente stesso non avrà alcuna costrizione nella scelta dell'impianto presso cui recarsi. Una volta attivati gli impianti, il modo in cui si comporteranno i clienti potrebbe non coincidere con ciò che si sarebbe

aspettato il locatore. Clienti diversi infatti possono avere preferenze diverse per varie ragioni (mentalità, abitudini, lavoro, età, reddito, etc.).

Per risolvere tale situazione si considera una formulazione del problema che tiene in considerazione le preferenze dei clienti : si parla infatti di *Simple Plant Location Problem with Order* (SPLPO).

Formulazione del problema:

Il modello del Simple Plant Location è orientato alla localizzazione di servizi puntuali considerando, come obiettivo, la minimizzazione del costo totale, comprensivo di costi di localizzazione e di afferenza.

Si consideri dunque un grafo connesso e non orientato, e si supponga la presenza di una domanda associata ad ogni nodo $i \in I$; la formulazione del problema è quindi la seguente:

$$\min z = \sum_i \sum_j c_{ij} x_{ij} + \sum_j r_j y_j \qquad (4.1)$$

Soggetta ai vincoli:

$$\sum_j x_{ij} = 1 \qquad \text{per ogni } i \in I \qquad (4.2)$$

$$x_{ij} \leq y_i \qquad \text{per ogni } i \in I,\ j \in J \qquad (4.3)$$

$$x_{ij} \geq 0 \qquad \text{per ogni } i \in I,\ j \in J \qquad (4.4)$$

$$Y_i = 0, 1 \qquad \text{per ogni } j \in J \qquad (4.5)$$

La funzione obiettivo (4.1) è la somma dei costi di afferenza e dei costi di localizzazione dei servizi. I vincoli (4.2) garantiscono che la domanda totale generata da ciascun nodo *i* sia soddisfatta, mentre le relazioni (4.3) assicurano che la domanda sia servita solo da servizi aperti, dal momento che $y_j=0$ implica che $x_{ij}=0$ per ogni j. Una volta individuato il valore delle y_j e, quindi, la posizione dei servizi da localizzare, ad ogni servizio afferiscono le domande associate ai nodi con costi di afferenza minori. Nel caso in cui ciascun servizio *j* presenti una capacità massima K_j, i vincoli (4.3) diventano: $\sum_i d_i x_{ij} \leq \sum_i y_i$, essendo d_i la domanda associata ad ogni $i \in I$.

Una euristica migliorativa per la risoluzione del Simple Plant Location, si fonda sull'osservazione che, data una soluzione ammissibile che presenta certi servizi attivi, l'apertura di un nuovo servizio può migliorare ulteriormente la funzione obiettivo se la riduzione dei costi di afferenza è superiore al costo di localizzazione del nuovo servizio.

L'algoritmo parte dalla soluzione non ammissibile in cui tutte le potenziali localizzazioni risultano chiuse: $y_j = 0$ per ogni j.

Al generico passo si valuta l'opportunità di aprire un nuovo servizio calcolando l'eventuale risparmio, *saving*, che si ottiene sulla funzione obiettivo . Tra le localizzazioni ancora chiuse, caratterizzate da savings positivi, si sceglie di attivare il servizio a cui è associato il saving maggiore; nel caso in cui i saving sono tutti negativi, l'algoritmo si arresta.

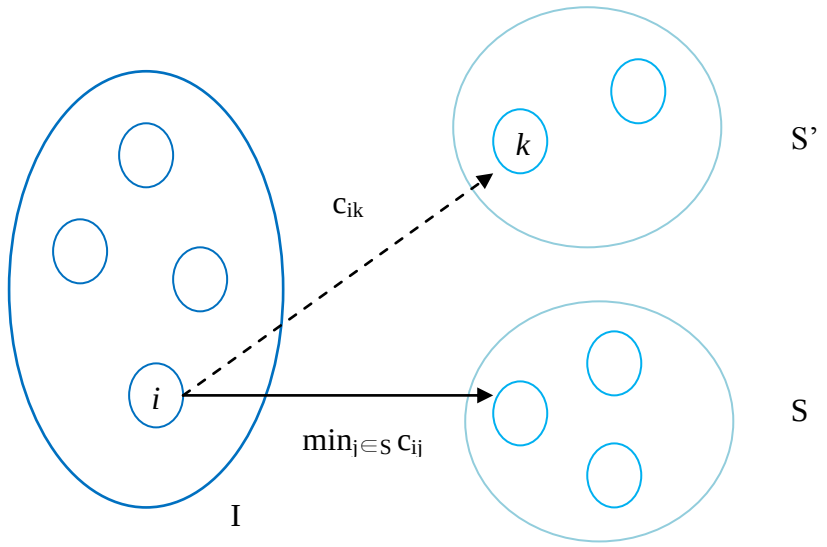
Calcolo dei saving:

Sia S l'insieme dei servizi già localizzati e S' l'insieme dei servizi candidati ad essere aperti; poiché il modello assegna la domanda al servizio aperto più vicino, il costo di afferenza da ciascun nodo i ∈ I all'insieme dei servizi S' è dato da: $\min_{j \in S} c_{ij}$.

Perciò l'apertura di un nuovo servizio k∈S' produce un risparmio nel costo di afferenza da *i* solo se $c_{ik} \leq \min_{j \in S} c_{ij}$. La differenza, se positiva, produce un risparmio nei costi di afferenza della domanda in *i* per l'apertura di una nuova localizzazione in *k*.

Se consideriamo il costo di localizzazione r_k, il saving s_k associato all'apertura del servizio in *k* risulta:

$$s_k = \sum_i \max\left(min\ c_{ij} - c_{ik}, 0\right) - r_k$$



Variazione di costi di afferenza per l'apertura di un nuovo

Algoritmo:

L'algoritmo euristico si articola quindi nel seguente modo:

Inizializzazione Per ogni potenziale localizzazione $j \in J$, si calcola: $w_j = r_j + \sum_i c_{ij}$ e si localizza il primo servizio nella posizione j^* tale che $w_{j^*} \leq w_j$ per ogni j. Si pone S={j*}, S'=J-{j*}, l(j)=+1 per ogni $j \in S'$ e $z(S)=w_{j^*}$. l(j) individua le localizzazioni che se aperte potrebbero provocare una riduzione del costo.

Calcolo dei savings per ogni $j \in S'$ si calcolano i savings $s_j = \sum_i \max(min\ c_{ik} - c_{ij}, 0) - r_j$ e si pone l(j)=+1 per ogni j tale che $s_j > 0$.

Criterio di arresto Se $s_j \leq 0$, per ogni $J \in S'$ l'algoritmo si arresta. In caso contrario si individua l'indice j* corrispondente al massimo degli $s_j > 0$ calcolati, e si pone S=S U{j*}, S'={*j* non appartenenti a S: l(j)=+1} e $z(S)=z(S)-s_{j^*}$. Se S' è vuoto allora l'algoritmo si arresta, altrimenti si ritorna allo step 2.

Al termine della procedura l'insieme S rappresenta la soluzione del problema.

La procedura descritta rappresenta un classico algoritmo migliorativo: a partire da una prima soluzione ammissibile individuata allo step 1, al passo 2 si definisce l'apertura di un nuovo servizio e si sceglie la soluzione che produce il minor costo (step 3). L'algoritmo si arresta quando non è più possibile ottenere miglioramenti e converge, così, in un minimo locale.

2.2 Simple Plant Location Problem with Order (SPLPO)

Nel modello proposto da Hanjoul e Peeters (1987) e rivisto da Canovas et al. (2006) ogni cliente stila una classifica sulla preferenza dei potenziali impianti e, una volta attuata la scelta di attivazione degli impianti stessi (locazione), il cliente stesso sarà connesso all'impianto attivato con preferenza maggiore. Il modello quindi assume come postulato che il locatore conosca l'ordine di preferenza degli impianti proprio di ogni cliente.

Formulazione del problema:

Si consideri un insieme di clienti *I* di dimensione *m* e un insieme di potenziali impianti *J* di dimensione *n*. Sono noti i costi di distribuzione dei prodotti $c_{ij} \geq 0$ dall'impianto *j* al cliente *i*, e i costi di attivazione $f_j \geq 0$ dell'impianto *j*. Sia $i \in I$ e $k, j \in J$: diremo che

l'impianto *k* è peggiore per il cliente *i* rispetto a *j* se il cliente *i* preferisce l'impianto *j* a *k*. Esprimiamo allora la preferenza dei clienti come $k <_i j$. Si supporrà inoltre che le preferenze dei clienti siano strette, ossia se $k =_i j$ allora $k = j$.

Come già visto per la formulazione classica del SPLP si utilizzeranno le seguenti variabili : x_{ij} rappresenta la frazione della domanda del cliente *i* che viene soddisfatta dall'impianto *j* ; $y_j = 1$ se l'impianto *j* è attivato (altrimenti $y_j = 0$).

Il modello considerato (proposto per la prima volta da Hanjoul e Peeters, 1987) è il seguente :

$$z = \min \left(\sum_{i,j} c_{i,j}\, x_{i,j} + \sum_k f_k\, y_k \right) \qquad (5a)$$

soggetta ai vincoli

$$\sum_j x_{i,j} = 1, \text{ per ogni } i; \qquad (5b)$$

$$x_{i,j} \le y_j \text{ , per ogni } i,j; \qquad (5c)$$

$$\sum_{\{k\,:\,k \ge_i j\}} x_{i,k} \ge y_j \text{ , per ogni } i,j; \qquad (5d)$$

$$x_{i,j} \ge 0, \text{ per ogni } i,j; \qquad (5d)$$

$$y_j \in \{0,1\}, \text{ per ogni } j. \qquad (5e)$$

Come si può notare questo modello non è altro che la formulazione classica del SPLP congiuntamente alla famiglia di disequazioni (5d) che esprime le preferenze dei clienti: se un impianto è attivato, ogni cliente *i* verrà servito dall'impianto *j* o in alternativa da un altro impianto che *i* preferisce a *j*.

Per questo motivo le disequazioni (5d) prendono il nome di vincoli di preferenza.

Hanjoul e Peeters (1987) propongono un algoritmo risolutivo di tipo *greedy* basato sull'*interchange heuristic*.

Canovas et al. (2006) rivisitano la stessa formulazione del SPLPO e tramite alcune regole di *preprocessing* riducono le dimensioni del problema consentendo una maggiore efficienza dell'algoritmo stesso.

2.3 Simple Plant Location Problem with Convex transportation costs (SPLPC)

A differenza del SPLP in cui i costi di trasporto sono lineari, nel SPLPC questi ultimi non sono lineari ma convessi. Questo aspetto assieme alla concavità dei costi di produzione (i costi fissi propri di ogni impianto) rendono il problema più complesso.

Consideriamo il modello proposto da Holmberg (1998) con *m* potenziali locazioni per gli impianti e *n* clienti. Se un impianto *i* è attivato si terrà in considerazione anche un costo fisso f_i proprio dell'impianto *i*. La domanda del cliente *j* è w_j.

Le variabili in gioco sono x_{ij} e z_i , rispettivamente variabile decisionale che esprime il flusso dall'impianto *i* al cliente *j*, e variabile ausiliaria che vale *1* se l'impianto *i* è attivato, *0* altrimenti.

Il costo di trasporto per il flusso x_{ij} tra impianto *i* e cliente *j* è descritto da $g_{ij}(x_{ij})$ che si assume sia una funzione non lineare convessa. Si ottiene quindi il seguente modello :

$$z = \min \left(\sum_i \sum_j g_{ij}(x_{ij}) + \sum_i f_i z_i \right) \qquad (6a)$$

soggetta ai vincoli

$$\sum_i x_{ij} = w_j \text{ , per ogni j;} \qquad (6b)$$

$$x_{ij} - (w_j z_i) \leq 0 \text{ , per ogni i,j;} \qquad (6c)$$

$x_{ij} \geq 0$, intera, per ogni i,j; (6d)

$z_i \in \{0,1\}$, per ogni i. (6e)

Il SPLPC è un problema intero con una funzione obiettivo non lineare, la cui risoluzione passa per una linearizzazione della stessa funzione obiettivo. Il problema risultante, lineare intero, sarà caratterizzato da un numero maggiore di variabili in gioco; il numero di variabili dipende dai valori assunti dalle domande w_j secondo la relazione $m(1+\sum_j w_j)$.

2.4 Simple Plant Location Problem with Spatial interaction (SPLPS)

Il *Simple Plant Location Problem with Spatial interaction* è un modello che ben descrive un problema di locazione di impianti o attività pubbliche, in cui i clienti sono liberi di scegliere quale impianto utilizzare.

Esso si configura allora come un modello non lineare che tenta di migliorare il classico SPLP, il quale non tiene in considerazione l'interazione spaziale dei clienti; rispetto al SPLPO il modello in questione rappresenta una formulazione che meglio rispecchia l'evoluzione dinamica della domanda dei clienti in relazione ai costi di fornitura dei prodotti o dei servizi (e quindi in relazione al prezzo sostenuto dai clienti stessi).

2.5 Simple Plant Location Problem with General cost functions (SPLPG)

Wu et al. (2006) propongono una formulazione generale per il SPLP in cui i costi di installazione di ogni impianto vengono rappresentati da una generica funzione dipendente dalle dimensioni dell'impianto stesso. In realtà questo modello rappresenta una semplificazione del Capacitated Plant Location Problem with General setup cost (CPLPG)

Formulazione del modello:

Si consideri un insieme di strutture *I* e un insieme di clienti *J*. Sia $g_i(z)$ una funzione non decrescente per ogni impianto *i* ; $g_i(z_i)$ è il costo di installazione dell'impianto *i* con

dimensione z_i , dove z_i è il numero di clienti serviti dall'impianto stesso. z_i assume valori interi non negativi e in particolare sarà $z_i>0$ se l'impianto *i* è attivato, *0* altrimenti.

Il costo di distribuzione dall'impianto *i* al cliente *j* è c_{ij}. La variabile decisionale x_{ij} vale *1* se il cliente *j* è servito dall'impianto *i*, *0* altrimenti.

Il *Simple Plant Location Problem with General cost functions* può essere allora formulato come segue :

$$\min \sum_i g_i(z_i) + \sum_i \sum_j c_{ij}\, x_{ij} \qquad (8a)$$

soggetta ai vincoli

$$\sum_i x_{ij} = 1, \text{ per ogni } j \in J; \qquad (8b)$$

$$\sum_j x_{ij} \leq z_i \text{ , per ogni } i \in I; \qquad (8c)$$

$$x_{ij} \in \{0,1\}, \text{ per ogni } i \in I, \text{ per ogni } j \in J; \qquad (8d)$$

$$z_i \geq 0 \text{ intero, per ogni } i \in I; \qquad (8e)$$

La funzione dei costi di installazione $g_i(z)$ può essere una funzione qualsiasi, concava, convessa o lineare.

Si noti che se $g_i(z) = f_i$ per $z > 0$, dove f_i rappresenta un costo fisso di installazione, allora il SPLPG si riduce al classico SPLP. In realtà la funzione $g_i(z)$ assume spesso una concavità dovuta alle economie di scala.

La risoluzione del SPLPG è ottenuta tramite un algoritmo euristico che passa per un rilassamento Lagrangeano del problema stesso.

Per completezza si cita il Dynamic Simple Plant Location Problem, dove si considera un dato orizzonte temporale per la risoluzione di un problema di locazione in cui le condizioni mutano all'interno di questo orizzonte, a differenza degli altri problemi affrontati, definiti statici poiché prevedono una soluzione in uno specifico istante

temporale e non tengono in considerazione la natura dinamica propria dei sistemi reali in cui i vari parametri cambiano nel tempo.

2.6 Esempio di Simple Plant Location Problem

Si consideri un grafo non orientato di 6 nodi, tutti potenziali localizzazioni di servizi (J≡V) .

Riportiamo nelle tabelle i costi di localizzazione, di afferenza, ed i valori di domanda.

Nodi	**1**	**2**	**3**	**4**	**5**	**6**
Costi r_j	7	3	8	5	10	6
Domanda d_i	1	1	1	1	1	1

Tab. 1 Costi di localizzazione

Nodi	*1*	*2*	*3*	*4*	*5*	*6*
1	*0*	*5*	*7*	*3*	*4*	*9*
2	*5*	*0*	*10*	*6*	*8*	*5*
3	*7*	*10*	*0*	*4*	*7*	*8*
4	*3*	*6*	*4*	*0*	*2*	*9*
5	*4*	*8*	*7*	*2*	*0*	*6*
6	*9*	*5*	*8*	*9*	*6*	*0*

Tab. 2 Matrice dei costi di afferenza

A partire dai dati della matrice dei costi di afferenza, il calcolo dei savings si riduce ad un confronto tra elementi di una riga.

Si supponga ad esempio che l'insieme S dei servizi già aperti sia {1, 2, 3}; il costo di afferenza del nodo 5 è pari a $min_{j \in S} c_{5j}$: esso si può ricavare confrontando gli elementi della riga 5 posti nelle colonne {1,2,3} ottendendo:

$min_{j \in S} c_{5j} = min(c_{51}, c_{52}, c_{53}) = c_{51} = 4$

Se invece si vuole valutare l'eventuale risparmio nel costo di afferenza dal nodo 5 ad esempio a seguito dell'apertura di un nuovo servizio in 4, bisogna calcolare la differenza:

$((min_{j \in S} c_{5j}) - c_{54}) = c_{51} - c_{54} = 2$

Poichè la differenza è positive, si ottiene una riduzione del costo di afferenza.

Sviluppo dell'algoritmo:

Inizializzazione:

$w_1 = r_1 + \sum c_{i1} = 7+(0+5+7+3+4+9) = 35$

$w_2 = r_2 + \sum c_{i2} = 3 +(5+0+10+6+8+5) = 38$

$w_3 = r_3 + \sum c_{i3} = 8 + (7+10+0+4+7+8) = 44$

$w_4 = r_4 + \sum c_{i4} = 5 +(3+6+4+0+2+9) = 29$

$w_5 = r_5 + \sum c_{i5} = 10+(4+8+7+2+0) =37$

$w_6 = r_6 + \sum c_{i6} = 6+(9+5+8+9+6+0) =43$

j = 4 -> S={1,2,3,5,6}; z(s)=29 l(1)=l(2)=l(3)=l(5)=l(6)= -1*

Calcolo dei savings:

$s_1=(3+1+0+0+0+0)-7 = -3$

$s_2 = (0+6+0+0+0+4)-3= +7$

$s_3= (0+0+4+0+0+1)-8 = -3$

$s_5=(0+0+0+0+2+3)-10= -5$

$s_6=(0+1+0+0+0+9)-6= +4$

l(2)=l(6)= +1

Criterio di arresto:

j=2 -> S={2,4}; S'={6}; z(S)=29-7=22 -> si ritorna al punto 2.*

Calcolo dei savings:

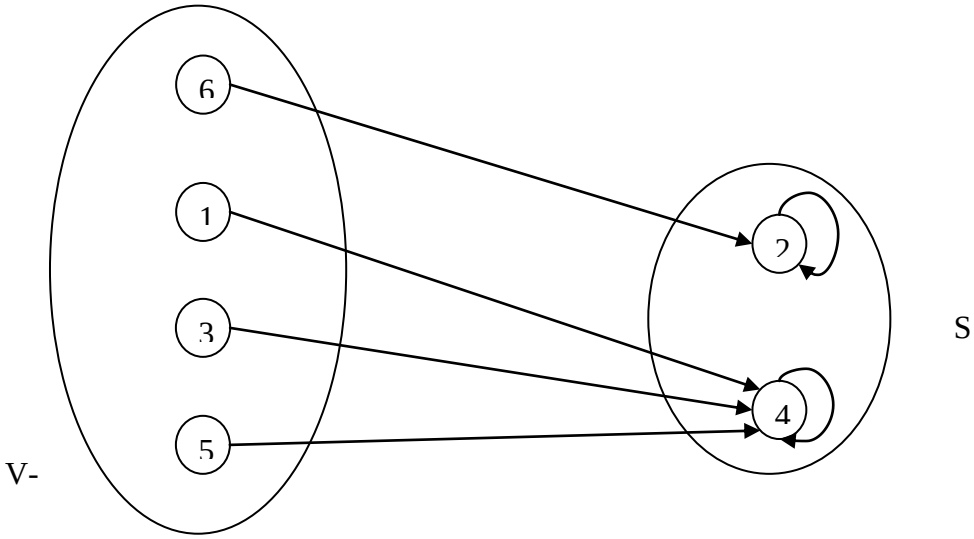
s_6=(0+0+0+0+0+5)-6= -1

Criterio di arresto

Poiché gli s_j sono negativi per ogni j∈S' l'algoritmo si arresta.

La soluzione individuata è S={2,4} con valore della funzione obiettivo pari a 22.

Inoltre si ha, in termini di variabili decisionali : $y_2=y_4= 1$ e $y_1=y_3=y_5=y_6=0$; mentre per le variabili di tipo x_{ij}; $x_{14}=x_{22}=x_{34}=x_{44}=x_{54}=x_{62}= 1$ e per tutte le altre coppie (i,j) si ha $x_{ij}=0$



2.7 Problemi NP-Hard

Vediamo ora una breve precisazione sulla tipologia di problema affrontato.

La Teoria della Complessità si occupa di determinare la complessità di un algoritmo. Per complessità si intende una funzione che associa il numero di dati da trattare al numero di operazioni da eseguire per trattare i suddetti dati.

I problemi si classificano in due categorie in base alla loro complessità:

Problema P, *polinomiale*, se esiste un algoritmo polinomiale che lo risolve;

Problema *NP-completo* (o *NP-hard*) se non si conosce alcun algoritmo polinomiale per risolverlo e se, qualora un tale algoritmo esistesse, allora esisterebbe per tutti i problemi NP-completi (per cui si *sospetta fortemente* che non esista).

Il Simple Plant Location è un problema NP-hard.

Nella Teoria della Complessità i problemi NP-difficili o NP-hard, *nondetermistic polynomial-time hard*, sono una classe di problemi che può essere definita informalmente come la classe dei problemi *almeno* difficili come i più difficili problemi delle classi di complessità P e NP. Più formalmente, un problema *H* è NP-difficile se e solo se esiste problema NP-completo *L* che è polinomialmente riducibile ad *H*, ovvero tale che $L \leq_T H$.

La categoria dei problemi NP-difficili, a differenza delle classi P, NP e degli NP-completi, non è limitata per definizione ai soli problemi di decisione; vi appartengono infatti anche problemi di ottimizzazione e di altri generi.

La classe dei problemi NP-hard ha una grande rilevanza sia teorica che pratica. In pratica, dimostrare che un problema di calcolo è equivalente a un problema notoriamente NP-difficile significa dimostrare che è praticamente impossibile trovare un modo efficiente di risolverlo, cosa che ha molte implicazioni in informatica. Da un punto di vista teorico, lo studio dei problemi NP-difficili è un elemento essenziale della ricerca su alcuni dei principali problemi aperti della complessità.

(Un tipico esempio di problema NP-hard è il calcolo del cammino minimo completo in un grafo.)

Convenzioni sulla nomenclatura della famiglia NP:

La nomenclatura dei problemi NP è confusa: i problemi NP-ardui non sono in NP, nonostante vengano etichettati con tale nome. Nonostante questa contraddizione verbale, tale nome è ormai di uso comune.

NP-completo - identifica problemi che sono 'completi' dentro NP.

NP-difficile - identifica problemi che sono almeno complessi quanto quelli in NP (ma non appartengono necessariamente ad NP);

NP-semplici - identifica problemi che sono al massimo complessi quanto quelli in NP (ma non appartengono necessariamente ad NP);

NP-equivalenti - identifica problemi che sono esattamente equivalenti ad NP, (ma non appartengono necessariamente ad NP).

CAPITOLO 3

P-median Problem

3.1 Introduzione al PMP

Analizziamo ora il caso in cui i costi di localizzazione siano uguali per ogni potenziale servizio, fissato il numero *p* di localizzazioni da determinare, il contributo nella funzione obiettivo dovuto ai costi di localizzazione risulta costante e può quindi essere trascurato ai fini dell'ottimizzazione. Il problema perciò consiste nell'individuazione di *p* nodi nei quali localizzare i servizi allo scopo di minimizzare la somma dei costi di afferenza.

La soluzione di tale problema prende il nome di *p-mediana*.

Il problema della p-mediana (o PMP) può essere considerato come un problema puramente matematico: data una matrice D, n x m, si selezionano p colonne di D in modo che la somma dei coefficienti minimi di ogni riga entro queste colonne, risulti più piccola possibile.

Il modello della p-mediana e le sue estensioni sono utili come modelli per risolvere molte situazioni nel mondo reale come ad esempio la localizzazione di stabilimenti industriali, magazzini o di strutture pubbliche.

Come il simple plant location problem, anche il PMP è un problema di tipo NP-hard, per risolverlo quindi con tempi di calcolo ragionevoli si ricorre all'uso di algoritmi euristici; le quali si dividono in due categorie:

Classical Heuristics : - Constructive (CH)

Local Search (LS)

Metaheuristics

I metodi risolutivi di ogni categoria possono essere ulteriormente classificati in base alle loro similarità.

Di seguito riportiamo un esempio di possibile classificazione dei metodi per il PMP

Tipologia	Euristica	Riferimento
CH	Greedy	Kuehn and Hamburger (1963), Whitaker (1983)
	Stingy	Feldman et al.(1966), Moreno-Pèrez et al.(1991), Salhi and Atkinson (1995)
	Dual ascent	Galvao (1980, 1993), Erlenkotter (1978), Captivo (1991)
	Composite	Moreno-Pèrez et al. (1991), Captivo (1991), Pizzolato (1994), Salhi (1997)
LS	Alternate	Maranzana (1964)
	Interchange	Teitz and Bart (1968), Whitaker (1983), Densham and Rushton (1992), Hansen and Mladenovic´ (1997), Resende and Werneck (2003), Kochetov et al. (2005)
MP	Dynamic programmino	Hribar and Daskin (1997)
	Lagrangian relaxation	Cornuejols et al. (1977), Mulvey and Crowder (1979), Galvao (1980), Beasley (1993), Daskin (1995), Senne and Lorena (2000), Barahona and Anbil (2000), Beltran et al. (2004)
	Aggregation	Hillsman and Rhoda (1978), Goodchild (1979), Erkut and Bozkaya (1999), Casillas (1987), Current and Schilling (1987), Hodgson and Neuman (1993), Hodgson and Salhi (1998), Bowerman et al. (1999), Francis et al. (2000, 2003)
MH	Tabu search	Mladenovic´ et al. (1995, 1996), Voss (1996), Rolland et al. (1996), Salhi (2002),
		Kochetov (2001), Goncharov and Kochetov (2002)
	Variable neighborhood search	Hansen and Mladenovic´ (1997), Hansen et al. (2001), Garcia-Lopez et al. (2002),
		Crainic et al. (2004)
	Genetic search	Hosage and Goodchild (1986), Dibbie and Densham (1993), Moreno-Pe´rez et al. (1994),
		Estivill-Castro (1999), Alp et al. (2003), Chaudhry et al. (2003)
	Simulated annealing	Murray and Church (1996), Chiyoshi and Galvao (2000), Levanova and Loresh (2004)
	Heuristic concentration	Rosing et al. (1998), Rosing and ReVelle (1997), Rosing et al. (1999)
	Scatter search	Garcia-Lopez et al. (2003)
	Ant colony	Levanova and Loresh (2004)
	Neural networks	Dominguez Merino and Munoz Pèrez (2002), Dominguez Merino et al. (2003)
	Decomposition	Dai and Cheung (1997), Taillard (2003)

Tabella 3.1.

Classificazione delle euristiche p-mediana.

Tipologie: Constructive heuristics (CH), Local search (LS), Mathematical programming (MP) and MetaHeuristics (MH)

3.2 Formulazione del problema

Consideriamo un insieme J di m facilities, ed un insieme I di n utenti o destinazioni di domanda.

Occorre definire due insiemi di variabili decisionali:

$Y_j = 1$ con $j \in J$ se la facility è aperta, altrimenti $Y_j = 0$;

$X_{ij} = 1$ se l'utente i è servito da una facility localizzata in $j \in J$, altrimenti $X_{ij} = 0$.

Possiamo dunque formulare il problema come segue:

Funzione Obiettivo: Min $\sum_i \sum_j c_{ij} x_{ij}$ (3.1);

Vincoli:

$$\sum_j x_{ij} = 1 \ \forall j \epsilon J \quad (3.2);$$

$$x_{ij} < y_j \ \forall i,j \in I,J \quad (3.3);$$

$$\sum_j y_j = p \quad (3.4);$$

$$x_{ij},\ y_j \in \{0,1\} \quad (3.5);$$

Il vincolo (3.2) garantisce che la domanda di ogni cliente sia soddisfatta, l'equazione (3.3) invece assicura che la domanda sia servita solo la servizi aperti, mentre la (3.4) indica che il numero totale di facility aperte è pari a p.

3.3 L' algoritmo di Teitz e Bart

Un algoritmo migliorativo per la p-mediana, basato sulla sostituzione di vertici, è l'algoritmo di Teitz e Bart.

Esso parte da una soluzione S di *p* nodi scelti a caso. Al generico passo si verifica la possibilità di introdurre, al posto di un nodo $i \in S$, un vertice $j \in S$, calcolando l'eventuale

riduzione (o *savings*) s_{ij} della funzione obiettivo. Si effettua quindi la sostituzione tra i e j caratterizzati dal maggior savings positivo.

L'algoritmo ha molte similitudine con la procedura del Simple Plant Location analizzata nel capitolo 2.1; in quel caso si valutavano i savings s_j a seguito dell'apertura di un nuovo servizio, invece nel caso della p-mediana, dovendo essere aperti p servizi, si valutano i savings s_{ij} a seguito della sostituzione del nodo i con il nodo j.

Si indica con S la soluzione corrente, con z(S) il valore della funzione obiettivo ad essa associato, e con S' l'insieme dei nodi candidati a sostituire nodi di S.

L'algoritmo si sviluppa nei seguenti passi:

Inizializzazione

Si individuano p nodi che rappresentano la soluzione corrente S, è da notare che la scelta può essere anche del tutto casuale; si calcola z(S) e si pone:

$S' = J - S\ ,\ l(j) = 1\ \forall\ j \in S\ e\ l(j) = 0\ \forall\ j \in S'.$

Calcolo dei savings

Si sceglie un vertice $j \in S'$ tale che $l(j) = 0$ e si considera la possibilità di sostituirlo a ciascun vertice $i \in S$, calcolando il savings $s_{ij} = z(S) - z(S - \{i\} \cup \{j\})$.

Criterio di arresto

Se $s_{ij} \leq 0\ \forall\ i \in S$ si pone $l(j) = -1$; nel caso in cui $l(j) = -1\ \forall j \in S'$ allora l'algoritmo si arresta, altrimenti si ritorna al passo 2.

In caso contrario, ossia se esiste qualche *i* tale che $s_{ij} \geq 0$ si individua l'indice i* corrispondente al massimo dei savings positivi calcolati, e si pone: $S = S \cup \{j\} - \{i^*\}$, $S' = S' \cup \{i^*\} - \{j\},\ l(j^*) = -1,\ l(j) = 1\ \forall j \in S$ ed $l(j) = 0\ \forall\ j \in S' - \{i^*\}$.

Il valore $l(j) = -1$ è associato al nodo di S sostituito all'iterazione corrente e ai nodi di S' per i quali è già verificato che l'inserimento in S non ha prodotto miglioramenti della soluzione. Si deduce quindi che scegliendo al passo 2 i nodi con $l(j) = 0$ si evita di effettuare sostituzioni inutili.

Al termine della procedura, S è la soluzione del problema; mentre la domanda in i afferisce al nodo $j^* \in S$ tale che $c_{ij} = min_{j \in S}\, c_{ij}$

3.4 Esempio di p-median problem

Riprendiamo le tabelle 2.1 e 2.2 dell'esempio al capitolo 2.6.

Nodi	**1**	**2**	**3**	**4**	**5**	**6**
Costi r_j	7	3	8	5	10	6
Domanda d_i	1	1	1	1	1	1

Tab. 2.1 Costi di localizzazione

Nodi	*1*	*2*	*3*	*4*	*5*	*6*
1	*0*	*5*	*7*	*3*	*4*	*9*
2	*5*	*0*	*10*	*6*	*8*	*5*
3	*7*	*10*	*0*	*4*	*7*	*8*
4	*3*	*6*	*4*	*0*	*2*	*9*
5	*4*	*8*	*7*	*2*	*0*	*6*
6	*9*	*5*	*8*	*9*	*6*	*0*

Tab. 2.2 Matrice dei costi di afferenza

Tutti i nodi sono sia potenziali localizzazioni di servizi, sia nodi di domanda; cerchiamo di individuare la p-mediana dove *p=3.*

Occorre innanzitutto valutare i savings s_{ij} ossia il valore $z(S - \{i\} \cup \{j\})$ che si ottiene sommando sulla matrice dei costi di afferenza, il minimo di riga sulle colonne associate ai nodi $(S - \{i\} \cup \{j\})$.

Considerando ad esempio la soluzione S={1,2,3}, il valore $z(S - \{1\} \cup \{4\})$, o in altri termini z(2, 3, 4), è la somma dei minimi di riga relativamente alle colonne (2,3,4) ossia: (3+0+0+0+2+5)=10.

Il numero di elementi diversi da 0 di questa somma è *n-p*, dal momento che le domande poste in corrispondenza dei nodi candidati ad essere p-mediana sono soddisfatte dai nodi stessi con costi di afferenza nulli.

Vediamo in dettaglio il calcolo dei savings:

Iterazione 1:

$s_{14}=z(S)-z(\{2,3,4\})=12-(3+0+0+0+2+5) = +2$

$s_{24}= z(S)-z(\{1,3,4\})=12-(0+5+0+0+2+8)= -3$

$s_{34}= z(S)-z(\{1,2,4\})= 12-(0+0+4+0+2+5)=+1$

Iterazione 2 :

$s_{25}=z(S)-z(\{3,4,5\}) = 10-(3+6+0+0+0+6) = +2$

$s_{35}=z(S)-z(\{2,4,5\}) = 10-(3+0+4+0+0+5) = -2$

$s_{45}= z(S)-z(\{2,3,5\})= 10-(4+0+0+2+0+5) = -1$

Iterazione 3:

$S_{26}=z(S)-z(\{3,4,6\})= 10-(3+5+0+0+2+0)= 0$

$S_{36}=z(S)-z(\{2,4,6\})= 10-(3+0+4+0+2+0)=+1$

$S_{46}=z(S)-z(\{2,3,6\})= 10-(5+0+0+4+6+0)=-5$

Iterazione 4:

$S_{21}=z(S)-z(\{1,4,6\})= 9-(0+4+5+0+2+0)= -2$

$S_{41}=z(S)-z(\{1,2,6\})= 9-(0+0+7+3+4+0)= -5$

$S_{61}=z(S)-z(\{1,2,4\})= 9-(0+0+4+0+2+5)=-2$

Iterazione 5:

$S_{25}=z(S)-z(\{4,5,6\})= 9-(0+0+0+3+5+4)= -3$

$S_{45}=z(S)-z(\{2,5,6\})= 9-(4+0+7+2+0+0)= -4$

$S_{65}= z(S)-z(\{2,4,5\})=9-(3+0+4+0+0+5)= -3$

Lo sviluppo della procedura, a partire dalla soluzione iniziale S={1,2,3} con valore di z(S)=(0+0+0+3+4+5)=12, è sintetizzato nella tabella seguente:

Iteraz.	S	Z(S)	S'	*j*	s_{ij}	*i**	Sostituz.	*l(j)*					
								1	*2*	*3*	*4*	*5*	*6*
1	1 2 3	12	4 5 6	4	s_{14}=+2 s_{24}=-3 s_{34}=+1	1	4 → 1	-1	1	1	1	0	0
2	2 3 4	10	1 5 6	5	s_{25}=-5 s_{35}= -2 s_{45}=-1	-	-	-1	1	1	1	-1	0
3	2 3 4	10	1 5 6	6	s_{26}=0 s_{36}=+1 s_{46}=-5	3	6 →3	0	1	-1	1	0	1
4	2 4 6	9	1 3 5	1	s_{21}=-2 s_{41}=-5 s_{61}=-2	-	-	-1	1	-1	1	0	1
5	2 4 6	9	1 3 5	5	s_{25}=-3 s_{45}=-4 s_{65}=-3	-	-	-1	1	-1	1	-1	1

Tabella 3.2. Sviluppo dell'algoritmo di Teitz e Bart

CAPITOLO 4

Algoritmi meta euristici

Nei capitoli precedenti abbiamo visto degli algoritmi per la risoluzione del *Simple Plant Location Problem* e del *p-median problem*. I vari passaggi utilizzati da tali algoritmi possono essere inseriti ed utilizzati anche all'interno di procedure meta euristiche quali *Tabu Search* e *Simulated Annealing.*

4.1 Simulated Annealing

Il Simulated Annealing (SA) è un procedimento che consente di raggiungere la soluzione ottimale per problemi di ottimizzazione complessi.

Negli ultimi anni ha sviluppato un notevole interesse, che ebbe inizio con il lavoro di Kirkpatrick et al. (1983) e Cerny (1985). Essi dimostrarono come un modello per la simulazione del trattamento termico di ricottura dei materiali metallici, possa essere usato anche per problemi di ottimizzazione, in cui la funzione obiettivo da minimizzare corrisponde all'energia di stato del metallo.

Il Simulated Annealing viene applicato in molti problemi di ottimizzazione che riguardano diverse aree come: progettazione di computer (VLSI), elaborazioni di immagini, fisica e chimica molecolare e job shop scheduling.

Come già visto, i problemi di ottimizzazione combinatoria sono NP-hard, ciò implica che gli algoritmi impiegati necessitano di un elevato tempo di elaborazione per raggiungere la soluzione ottimale. SA è uno dei molti approcci euristici progettati per dare una buona soluzione, non necessariamente l'ottimo, in un tempo ragionevole.

Simulated Annealing è un algoritmo di tipo Local Search

Supposto che lo spazio-soluzione S sia l'insieme finito di tutte le soluzioni, e la funzione costo *f* sia una funzione reale definita in S, lo scopo è quello di trovare una soluzione o stato $i \in S$ tale che minimizzi *f* in S.

Una formulazione semplice di Local Search è l'algoritmo di discesa (*descent algorithm*) il quale inzia con una soluzione di partenza, probabilmente scelta in modo casuale; viene poi generato un *neighbours of a state,* o vicino di stato, si tratta di un nuovo stato del problema ottenuto modificando lo stato attuale, ossia la soluzione iniziale. L'azione intrapresa atta a modificare la soluzione primaria si chiama *"move"* o *movimento*; movimenti diversi danno neighbours diversi.

Una volta determinata la soluzione vicina, si calcola la differenza di costo; se viene trovata una riduzione nei costi la soluzione corrente viene rimpiazzata da quella appena generata, in caso contrario di mantiene la soluzione di partenza.

Il processo viene ripetuto fino a quando non è possibile determinare un ulteriore miglioramento nelle soluzioni vicine. L'algoritmo quindi termina in un minimo locale.

Nonostante l'algoritmo di discesa sia semplice e veloce da eseguire, presenta uno svantaggio, infatti vi è il rischio che il minimo locale trovato sia molto lontano dal minimo globale. Un modo per affinare la soluzione è quello di far partire l'algoritmo con differenti soluzioni iniziali contemporaneamente, e selezionare il migliore dei minimi locali trovati.

```
Select an initial state i ∈S

Repeat:

        Generate state j , a neighbour of  i ;

        Calculate δ = f( j )-f( i )

        If   δ≤0 then i ≡  j ;

Until:

        f( j ) ≥ f( i )   for all j in the neighbourhood of  i ;
```

Tabella 4.1: Descent algorithm in pseudo-code.

Nel Simulated Annealing, l'algoritmo tenta di evitare di rimanere intrappolato nell'ottimo locale accettando, a volte, neighbours che incrementano il valore *f*.

La probabilità di accettare o meno una soluzione vicina è specificata da una *funzione di accettazione* definita come:

$$exp^{-\delta/T}$$

Dove: δ è l'incremento di *f* e T è un parametro di controllo, che nell'analogia fisica della ricottura corrisponde alla Temperatura.

La funzione accettazione implica che venga sempre preferito un incremento δ minore. Si nota che

Più il valore di T è elevato, maggiori saranno le soluzioni accettate;

Se il valore di T→0 allora quasi tutte le nuove soluzioni trovate verranno rifiutate.

In genere l'algoritmo inizia con un valore di T elevato, che viene poi gradualmente diminuito.

```
Select: an initial state  i∈S ;
        an initial temperature T > 0 ;
Set :   temperature change counter t=0
Repeat:
    Set repetition counter n=0 ;
    Repeat:
        Generate state j, a neighbour of i ;
        Calculate δ = f( j ) - f( i ) ;
        If δ<0 then i ≡  j
        else if  random (0,1) <  exp( -δ / T )  then i ≡ j ;
        n ≡ n+1
    until: n ≡ N(T);
        t = t+1;
        T= t(T)
Until: stopping criterion true.
```

Tabella 4.2: Simulated Annealing algorithm in pseudo-code

4.2 Tabù Search

Per quanto riguarda la Tabu Search (TS), è una tecnica meta-euristica utilizzata per la soluzione di numerosi problemi di ottimizzazione, tra cui problemi di scheduling e routing , problemi su grafi e di programmazione intera. Essa rappresenta un'evoluzione del classico "metodo di discesa", utilizzato per trovare il minimo di una funzione reale f su un insieme S (spazio delle soluzioni).

Nella sua forma attuale, la Tabu Search è stata proposta da Fred Glover. Idee simili erano state abbozzate in contemporanea anche da P. Hansen, mentre il concetto fondamentale di usare proibizioni per incoraggiare una diversificazione della ricerca appare in vari lavori precedenti, come ad esempio nell' algoritmo di Kernighan–Lin per il partizionamento dei grafi.

I numerosi esperimenti computazionali effettuati dimostrano che la Tabu Search è ormai una tecnica di ottimizzazione affermata, che può competere con quasi tutti i metodi noti, e che grazie alla sua flessibilità, può battere molte delle procedure classiche di ottimizzazione. Insieme al Simulated Annealing e agli algoritmi genetici, la Tabu Search è stata indicata dal Committee on the Next Decade of Operations Research come tecnica estremamente promettente per il trattamento futuro di applicazioni pratiche.

Recentemente U. Faigle e W. Kern hanno proposto una versione probabilistica della Tabu Search che converge quasi sicuramente ad un ottimo globale. Tali risultati teorici, sebbene interessanti, sono piuttosto irrilevanti per le applicazioni pratiche.

L'obiettivo della Tabu Search, come di altre tecniche euristiche, è di trovare rapidamente soluzioni buone per problemi che spesso non permettono di determinare soluzioni ottimali in tempi ragionevoli.

Il concetto fondamentale della Tabu Search consiste nel rendere "proibite" o, appunto, "tabu", le ultime mosse eseguite nel cammino di ricerca, in modo che l'algoritmo non possa tornare sui suoi passi e ricadere nel minimo locale.

Ad esempio, se la nostra soluzione di partenza è x_n, allora si cerca nell'insieme di adiacenza V(x_n) una soluzione x migliorativa. Può accadere però che la struttura del vicinato sia simmetrica: quindi $x_n \in V(x)$, dove V(x) è l'insieme di adiacenza di x, ogni qualvolta che $x \in V(x_n)$. Si ha dunque il rischio di *cycling* ossia si rischia di oscillare sempre tra le soluzioni x ed x_n senza arrivare ad un ottimo.

La lista tabù serve appunto per evitare la situazione di *cycling,* si tratta quindi di una lista in cui si memorizzano uno o più attributi delle soluzioni recentemente analizzate, o degli ultimi “movimenti”.

Inizialization:
Select an initial solution x_1 in X;
Initialize the best value F*of F and the corresponding solution x*:
$F(x_1) \rightarrow F^*$
$x_1 \rightarrow x^*$
Tabù List is empty;
Step n= 1,2,…;
x_n denote the current solution;
$\bar{F}$ is used to store the best accessible value of F met during the exploration of the sub-neighbourhood V’(x_n).
$\bar{x}$ denote the solution in V’(x_n) for which $F(\bar{x}) = \bar{F}$
Inizialize: $\bar{F}$ to $\bar{F} \leftarrow \infty$.
For all x in V’(x_n),
If $F(x) < \bar{F}$ and (if the move $(x_n \rightarrow x)$ is not tabu or if the move is tabu but passes the aspiration criterion),
Then $\bar{F} \leftarrow F(x) \text{and } \bar{x} \leftarrow x$.
Let $x_{n+1} \leftarrow \bar{x}$.
If $\bar{F} < F^*$,
Then $x^* \leftarrow \bar{x}$ and $F^* \leftarrow \bar{F}$.
The appropriate characteristic of the move $x_n \rightarrow x_{n+1}$ enters the tabu list once the first entered characteristic has been removed if the list was full.
End: if the stopping criterion is fulfilled, then stop.

Tabella 4.3: Tabù Search in pseudo-code

Caratteristica basilare della Tabu Search è quindi l'uso sistematico della memoria: per aumentare l'efficacia del processo di ricerca, viene tenuta traccia non solo delle informazioni locali (come il valore corrente della funzione obiettivo), ma anche di alcune informazioni relative all'itinerario percorso. Tali informazioni vengono impiegate

per guidare la “mossa” dalla soluzione corrente alla soluzione successiva, da scegliersi all'interno dell'insieme di adiacenza.

La Tabù Search incorre però in alcuni problemi, infatti registrare una descrizione completa delle ultime soluzioni analizzate, e verificare per ogni soluzione candidata se è memorizzata nella lista può causare uno spreco di tempo.

4.3 Algoritmi Genetici

L'algoritmo genetico (GA) è un algoritmo di ottimizzazione e appartiene a una particolare classe di algoritmi utilizzati in diversi campi, tra cui l'intelligenza artificiale. È un metodo euristico di ricerca ed ottimizzazione, ispirato al principio della selezione naturale di Charles Darwin che regola l'evoluzione biologica.

Il nome deriva dal fatto che i suoi primi pionieri si ispirarono alla natura e alla genetica, branca della biologia.

Gli algoritmi genetici sono applicabili alla risoluzione di un'ampia varietà di problemi d'ottimizzazione non indicati per gli algoritmi classici, compresi quelli in cui la funzione obiettivo è discontinua, non derivabile, stocastica, o fortemente non lineare.

La principale differenza con i due approcci precedenti, il Simulated Annealing e la Tabù Search, consiste nel fatto che l’algoritmo genetico si occupa di una *popolazione* di soluzioni invece che di singole soluzioni.

Un vantaggio è il parallelismo intrinseco che va oltre il lasciare che le soluzioni evolvano indipendentemente parallelamente: infatti le soluzioni interagiscono, si uniscono e producono “*figli”* che assumono le caratteristiche positive dei genitori.

Anche per quanto riguarda gli *stati vicini,* nel GA si parla di *neighbourhood* di popolazioni, non di singole soluzioni e gli operatori principali usati per generare ed analizzare questi insiemi adiacenti sono *selezione, crossover e mutazione.*

La prima particolarità dell’algoritmo genetico è che gli operatori genetici sopra citati, non lavorano direttamente sull’insieme delle soluzioni; le soluzioni vengono codificate in una stringa di lunghezza finita, simile a quelle della programmazione matematica, in cui vengono memorizzate variabili booleane. In un problema di localizzazione la codifica del problema è rappresentata dalla stringa delle variabili y_i ad esempio:

0	1	1	0	1	0

Rappresenta, per un problema con 6 potenziali localizzazioni, la soluzione che prevede l'apertura dei servizi in corrispondenza dei nodi 2, 3 e 5.

Un algoritmo genetico inzia con una popolazione di partenza di N soluzioni, che evolvendosi, ad ogni iterazione, genera un'altra popolazione della medesima dimensione N .

la generazione di soluzioni (*n+1*) si ottiene dalla *n-esima* generazione $X^{(n)}$ attraverso la seguente procedura:

generazione, in maniera casuale, una popolazione iniziale;

creazione di una sequenza di nuove popolazioni, o generazioni. In ciascuna iterazione, gli individui della popolazione corrente sono usati per creare la generazione successiva, e a questo scopo si compiono degli ulteriori passi:

ciascun membro della popolazione corrente è valutato calcolandone il rispettivo valore di fitness (idoneità);

si determina un opportuno ordinamento di tali individui sulla base dei valori di fitness;

gli individui più promettenti sono selezionati come *genitori*;

a partire da tali individui si genera un pari numero di individui della generazione successiva, e ciò può avvenire secondo due modalità distinte, vale a dire effettuando cambiamenti casuali su un singolo genitore (mutazione) oppure combinando opportunamente le caratteristiche di una coppia di genitori (crossover);

gli individui così generati vanno a sostituire i genitori consentendo la formazione della generazione successiva;

infine, l'algoritmo s'interrompe quando uno dei criteri d'arresto è soddisfatto.

Ogni individuo di una popolazione $\{x_1, \ldots, x_N\}$ è valutato rispetto ad un parametro denominato *fitness*, più alto è il fitness dell'individuo più questo rappresenterà una buona soluzione. Il fitness è semplicemente il valore della funzione obiettivo massimizzata.

Analizziamo ora le operazioni genetiche:

Mutation o mutazione:

La mutazione consiste nella modifica casuale di alcune parti dei geni con valore di *fitness* più basso, in base a coefficienti definiti inizialmente. Queste modifiche puntano a migliorare il valore della funzione per il gene in questione.

Crossover o incrocio:

In base a un coefficiente stabilito inizialmente, alcune parti dei geni risultati migliori vengono scambiate, nell'ipotesi che questo possa migliorare il risultato della funzione di fitness nel successivo "passo evolutivo".

Single Point Crossover: Ci sono varie tecniche di crossing over. Una delle più semplice è la "single point crossing over" che consiste nel prendere due individui e tagliare le loro stringhe di codifica in un punto a caso. Si creano così due teste e due code. A questo punto si scambiano le teste e le code, ottenendo due nuovi geni.

Il crossing over non è applicato sempre, ma con una probabilità p_c.

Nel caso in cui non viene applicato i figli sono semplicemente le copie dei genitori. Sperimentalmente si può vedere che il miglioramento diventa apprezzabile solo dopo un certo numero di passi.

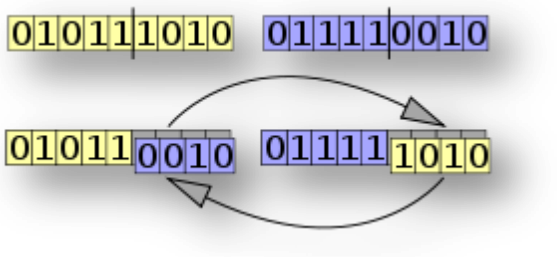


Figura 1 Single Point Crossover

CAPITOLO 5

Il progetto di reti

La distribuzione di beni e servizi necessita spesso dell'utilizzo di reti . Ad esempio, la domanda di comunicazione viene soddisfatta attraverso le reti di telefonia fissa e mobile, gli spostamenti delle persone vengono realizzati utilizzando reti di trasporto pubblico e privato, lo scambio di informazioni passa attraverso reti di comunicazioni locali o internet; la distribuzione di energia, acqua e gas avviene attraverso apposite reti, i beni prodotti devono essere consegnati presso centri di smistamento o punti di vendita attraverso una rete di trasporto.

Progettare una rete, dunque, significa definire gli elementi in modo da consentire una gestione efficace ed efficiente del servizio da realizzare sulla rete stessa. Il problema presenta caratteristiche diverse in funzione del contesto applicativo.

I modelli e gli algoritmi di localizzazione hanno una notevole rilevanza per problematiche relative alla pianificazione e alla gestione di reti di servizio. Questi modelli hanno come obiettivo generale quello di decidere dove localizzare i centri di servizio su una rete per soddisfare la domanda di servizio da parte di utenti che si muovono sulla rete stessa.

E' possibile distinguere due classi principali di modelli di localizzazione; da un lato i classici modelli di localizzazione, simple plant, p-mediana, visti nei capitoli precedenti; dall'altro, una nuova generazione di modelli nota come *Modelli di Intercettazione dei Flussi* o *Flow Interception Problems* il cui studio è più recente.

La differenza sostanziale tra queste due categorie, sta nel fatto che i primi ipotizzano che la domanda di servizio da parte degli utenti si origini dai nodi della rete, ossia che la domanda di servizio sia la motivazione dello spostamento dell'utente sulla rete, mentre nei modelli di intercettazione di flusso si suppone che il motivo del viaggio non sia la richiesta di un servizio, ma che l'esigenza di usufruire di un servizio intercorra durante il viaggio. Ciò determina una diversità negli obiettivi della localizzazione.

Per i modelli classici si vuole definire una localizzazione tale da ottimizzare una funzione obiettivo che dipende dalla distanza che gli utenti devono percorrere per usufruire del servizio; mentre i modelli di intercettazione cercano di localizzare punti di

servizio in modo da ottimizzare una funzione obiettivo che dipende dalla quantità di utenti intercettati. Applicazioni reali di tali modelli di intercettazione sono ad esempio la localizzazione di punti di monitoraggio per il controllo del trasporto di materiali pericolosi o la localizzazione di punti di stazionamento per l'intercettazione di particolari categorie di flusso. A seconda degli ambiti applicativi si distinguono tre categorie:

Modelli di tipo Punitivo

Modelli di tipo Preventivo

Modelli di tipo Estimativo

Definiamo ora più nel dettaglio il problema della progettazione di una rete.

5.1 Definizione di rete

Una rete si compone dei seguenti elementi:

Nodi sorgenti o origini: sono i luoghi in cui si producono i beni e i servizi da distribuire attraverso la rete; l'offerta del servizio ha orgine proprio dai nodi sorgenti.

Nodi terminali o destinazioni: sono i luoghi presso i quali vanno consegnati i beni o sono consumati i servizi; nei nodi terminali ha destinazione la domanda del servizio.

Nodi di interscambio: sono nodi caratteristici che assolvono a funzioni particolari nella gestione dei flussi sulla rete; i nodi di interscambio sono attraversati dal flusso proveniente dalle origini e diretto verso le destinazioni.

Collegamenti: sono le strutture fisiche e/o tecnologiche che connettono i vari punti della rete e sono caratterizzate da parametri fisici, tecnologici e di costo. A parità di caratteristiche tecnologiche, il costo è funzione dei parametri fisici; ad esempio un collegamento in fibra ottica presenta un costo che dipende dalla lunghezza e dalla capacità.

In Tabella 5.1 sono indicati, nei diversi casi, i possibili significati degli elementi della rete.

Servizio	Sorgenti	Terminali	Nodi di Interscambio	Collegamenti
Distribuzione merce	Luoghi di raccolta o produzione	Depositi o punti vendita	Porti, interporti, scali marittimi o ferroviari	Strade, ferrovie, vie marittime, linee aeree
Distribuzione energia elettrica	Centrali di produzione	Utenti	Stazioni di trasformazione	Linee elettriche di alta e media tensione
Distribuzione acqua	Sorgenti	Utenti	Serbatoi, dighe	Condotte idriche
Raccolta rifiuti	Cassonetti, depositi rifiuti	Discariche, centri di smaltimento	Isole ecologiche, depositi	Strade urbane ed extraurbane
Distribuzione posta	Uffici postali	Utenti, uffici postali	Centri di raccolta e smistamento	Collegamenti logistici
Servizi di posta elettronica	Server di rete	Server di rete	Providers	Collegamenti di rete
Comunicazioni	Centraline telefonia fissa o mobile	Utenti, centralini	Centrali, multiplexer, modulatori	Linee telefoniche, collegamenti aerei
Trasporto persone	Luoghi di residenza	Luoghi di lavoro o di studio	Stazioni, fermate	Linee di trasporto pubblico
Traffico veicolare	Luoghi generatori di traffico	Luoghi attrattori di traffico	Confluenze, incroci, rotatorie, piazze	Strade e autostrade

Tabella 5.1: Elementi della rete in funzione delle applicazioni.

Una rete può essere rappresentata da un grafo, orientato o non; sulla base di questa schematizzazione, il progetto di rete consiste nella determinazione di un insieme di

collegamenti e delle loro caratteristiche che ottimizzi una certa funzione obiettivo, nel rispetto dei vincoli presenti.

Nella pratica i problemi di progetto si pongono quando va definita una rete ex novo o quando, a partire da una rete esistente, si devono definire nuovi collegamenti per aumentare la capacità di distribuzione.

5.2 Formulazione del problema

Riconducendo il problema in termini di minimizzazione dei costi, anche in questo caso, come nei problemi di localizzazione, le principali voci di costo possono essere ricondotte a costi di localizzazione e costi di afferenza.

I vincoli da rispettare sono diversi a seconda della funzione obiettivo adottata; oltre ai vincoli legati alla struttura della rete, come la continuità del flusso dei nodi o la capacità sugli archi, possono esistere vincoli legati alle caratteristiche tecnologiche e geometriche.

Se indichiamo con:

d^{st} la domanda dell'origine *s* alla destinazione *t;*

f_{ij}^{st} il flusso che circola sull'arco (*i,j)* con origine *s* e destinazione *t*

Il problema di progetto di rete può essere formulato attraverso il seguente modello di programmazione matematica:

funzione obiettivo:

$$min \quad z = \sum_{i,j \in A} c_{ij} f_{ij} + \sum_{ij \in L} r_{ij} y_{ij} \tag{5.1}$$

Sottoposta ai vincoli

$$\sum_{j \in V} f_{ij}^{st} - \sum_{j \in V} f_{ji}^{st} = \begin{cases} +d^{st} \; per \; i \equiv s \\ -d^{st} \; per \; j \equiv t \\ 0 \; \forall \; i \in V - \{s,t\} \end{cases} \tag{5.2}$$

$$f_{ij} = \sum_{(s,t):d^{st}>0} f_{ij}^{st} \leq K_{ij} y_{ij} \qquad \forall \; i \in I, j \in J \tag{5.3}$$

$$y_{ij} \in Y \quad \forall \; (i,j) \in A \tag{5.4}$$

$$f_{ij}^{st} \geq 0 \quad \forall \; (i,j) \in A \tag{5.5}$$

La funzione obiettivo (5.1) rappresenta la somma dei costi di afferenza e dei costi di localizzazione degli archi .

Le relazioni (5.2) sono i vincoli di continuità del flusso, con riferimento al generico nodo *i* ed alla domanda avente origine in *s* e destinazione in *t*; essi indicano che la differenza tra il flusso uscente ed entrante è d^{st} se il nodo è *s*, - d^{st} se il nodo è *j*, 0 in tutti gli altri casi.

Le espressioni (5.3) impongono che il flusso totale sul generico arco *(i, j)* non superi la sua capacità K_{ij}, quando l'arco *(i,j)* fa parte della rete (quindi $y_{ij}=1$).

I vincoli 5.4 rappresentano eventuali vincoli aggiuntivi imposti dalla configurazione della rete e specificano il significato delle variabili y_{ij}. In particolare, se Y è costituito dai soli elementi {0,1}, il modello è di tipo discreto; ossia in questo caso y_{ij} è una variabile binaria che, se pari ad 1, indica che l'arco (i,j) fa parte del progetto; altrimenti se pari a 0 significa che l'arco non è presente.

Infine i vincoli (5.5) impongono la non negatività dei flussi.

Nel caso di problema binario e/o in presenza di vincoli complessi, il problema di progetto di rete rientra nella classe di problemi NP-hard.

Un particolare progetto di network design è il *Budget Design Problem* caratterizzato dalla presenza di un vicolo aggiuntivo:

$$\sum r_{ij} y_{ij} \leq B$$

(5.6)

Il quale rappresenta un limite superiore al costo del progetto.

In molti casi il processo decisionale viene suddiviso in fasi, o livelli, attraverso l'individuazione di un ordine gerarchico tra i collegamenti della rete. Il numero di livelli in cui si divide il progetto dipende dalle dimensioni e dall'estensione della rete.

Ai livelli più alti della gerarchia, la rete sarà caratterizzata da collegamenti di capacità superiore che hanno il compito di sostenere volumi maggiori di traffico; mentre il collegamento tra reti di livello successivo è assicurato da opportuni nodi che svolgono funzioni di interscambio tra i vari livelli, a seconda del servizio in questione.

Comunque, per ogni livello della gerarchia, è possibile definire un problema di progetto articolato nelle seguenti fasi:

Individuazione della localizzazione dei nodi terminali e di interscambio;

Individuazione dei collegamenti tra i nodi della rete selezionati;

Assegnazione dei nodi del livello inferiore ai nodi del livello superiore.

5.3 Transportation Network Desing (NDP)

Un esempio di problema di progettazione di reti è quello del progetto di una rete di trasporti.

Il Transportation Network Design Problem consiste nel progettare ex novo, o migliorare, o espandere una rete di trasporti considerando dei vincoli di risorse. Più specificatamente l'NDP implica la scelta tra un insieme di progetti in modo da ottimizzare una determinata funzione obiettivo soggetta a vincoli di costo, risorse e capacità.

Gli obiettivi generalmente si riferiscono ai benefici di cui possono godere gli utenti della rete stessa mentre i vincoli riguardano i mezzi atti a generare tali benefici a spese del gestore della rete. Normalmente quando la domanda origine-destinazione (O/D) è fissa, i vantaggi sono dati dalla riduzione del tempo totale di percorrenza.

Sia N(V,A) il grafo rappresentante la rete in questione, con V insieme dei nodi ed A insieme degli archi. Definiamo *y* come il vettore delle decisioni di progetto; $y_{ij} \in \{ 1,0 \}$ ciò significa che se y_{ij} =1, l'arco (i,j) è accettato, altrimenti y_{ij}=0 .

A_y è l'insieme dei possibili spigoli, e A_{y1} è quello dei collegamenti accettati, ossia è l'insieme tale che $A_{y1} = \{(i,j) \in A_y : y_{ij} = 1\}$.

Ogni arco (i,j) ha un costo C_{ij}, mentre il costo totale di realizzazione è limitato superiormente da il livello di budget B.

Definiamo inoltre un'altra variabile x_p^{ks} che indica il flusso del percorso *p* dall'origine *k* alla destinazione *s*, $p \in P_{ks}$ dove P_{ks} è l'insieme dei possibili percorsi che portano da k ad s. I flussi sono generati dalla domanda O/D (origine-destinazione) d^{ks} con $(k,s) \in P$, insieme delle coppie origine-destinazione.

La funzione che descrive il tempo di percorrenza dell'arco (i,j) è $t_{ij}(x_{ij})$, con $(i,j) \in AUA_y$, rappresenta il tempo medio, il quale si presume essere solo in funzione del flusso di

(i,j); inoltre $t_{ij}(x_{ij})$ è continua, differenziabile, integrabile con Riemann, strettamente convessa e definita per $x_{ij} \geq 0$.

La formulazione del problema è la seguente:

Funzione obiettivo

$$\min \quad f(y) = \sum_{(i,j)\in A\cup A_y} x_{ij}^{*}\, t_{ij}(x_{ij}^{*}) \qquad (5.7)$$

Soggetta ai vincoli:

$$\sum_{i,j\in A_y} C_{ij} y_{ij} \leq B \qquad (5.8)$$

$$y_{ij} \in \{0,1\} \quad \forall\, i,j \in A_y \qquad (5.9)$$

Il vincolo (5.8) stabilisce che i costi totali non superino il limite di budget previsto.

x_{ij}^{*} è detta User Equilibrium (UE) ossia equilibrio degli utenti; per definire lo User Equilibrium bisogna innanzitutto specificare che si parla di equilibrio stazionario, ossia che i flussi di domanda tra le O/D possono essere considerati costanti durante il periodo di analisi, che deve essere abbastanza più grande del tempo medio di spostamento O/D.

La definizione di UE perciò è la seguente: per ogni coppia O-D, all'equilibrio il tempo di spostamento è uguale in tutti i percorsi utilizzati, inoltre deve essere minore o uguale al tempo che si sarebbe sperimentato su qualunque percorso non utilizzato.

La variabile x_{ij}^{*} rappresenta la soluzione del Problema di Equilibrio, che può essere formulato matematicamente come segue:

Funzione Obiettivo (6.10)

$$\min \sum_{(i,j)\in A\cup A_{y1}} \int_{0}^{x_{ij}} t_{ij}(u)\,du$$

Sottoposta ai vincoli:

$$\sum x_p^{ks} = d^{ks} \qquad \forall\ (k,s) \in P \qquad (6.11);$$

$$x_{ij} = \sum_{k,s\in P} \sum_{p\in P_{ks}} x_p^{ks} \delta_{ij,p}^{ks} \qquad \forall\ (i,j) \in A \cup A_{y1} \qquad (6.12);$$

$$\delta_{ij,p}^{ks} = \begin{cases} = 1 & se \quad i,j \in p\ e\ p \in P_{ks} \\ = 0 & altrimenti \end{cases} \qquad (6.13)$$

Il Transportation Network Design Problem è quindi un problema di programmazione a due livelli e quindi molto complesso da risolvere.

Fu studiato per la prima volta da LeBlanc (1975) che propose una procedura di tipo Brench and Bound per risolverlo; successivamente furono pensati un'ampia gamma di approcci tali da risolvere efficacemente il problema valutando anche il trade-off tra accuratezza e rapidità della soluzione. Qui di seguito ne elencheremo alcuni:

Arrivare ad un problema di programmazione convesso tramite la sostituzione del flusso di equilibrio UE con il flusso di equilibrio del sistema.

Abbassare il livello di complessità del problema assumendo le funzioni di costo costanti, al fine di ottenere una funzione del tempo di percorrenza lineare.

Accelerare la procedura di soluzione attraverso l'aggregazione di link e nodi alla rete.

Scomporre il problema basandosi sulle tipologie di collegamenti.

Facilitare la procedura di progettazione utilizzando un approccio intrinseco che definisce un nuovo problema di programmazione meno complesso dell'NDP

Utilizzo di procedure euristiche quali Tabù Search, Algoritmi Genetici, Simulated Annealing ed Ant System.

Applicazione di algoritmi ibridi.

Attualmente comunque le ricerche sono in continuo sviluppo.

CAPITOLO 6

Albero a costo minimo

In molti problemi di progetto di rete risulta particolarmente efficiente la configurazione ad albero perché, assicurando la connessione tra tutti i nodi, consente di minimizzare i costi di collegamento.

Un albero è un grafo non orientato, connesso e senza cicli.

Da questa definizione ne consegue che esiste un solo percorso che unisce ciascuna coppia di nodi e che, non essendovi cicli, il numero di spigoli è *n-1*.

Data una matrice dei costi r_{ij} simmetrica e rappresentativa dei costi di collegamento tra ciascuna coppia di nodi (i,j); il problema dell'*albero a costo minimo* o *minimum spanning tree* consiste nell'individuare l'albero che connette tutti i nodi e tale che la somma dei costi associati agli spigoli presenti sia minima.

6.1 Minimum Spanning Tree (MST)

Il problema dell'albero a costo minimo può essere rintracciato, in forma implicita, in diversi contesti già all'inizio del XX secolo. Il problema tuttavia fu formulato la prima volta nel 1926 dal matematico O. Borůvka che ipotizzò una prima soluzione nell'articolo *On a certain minimum problem;* venne rivisto anche successivamente nel 1956 da Joseph B. Kruskal.

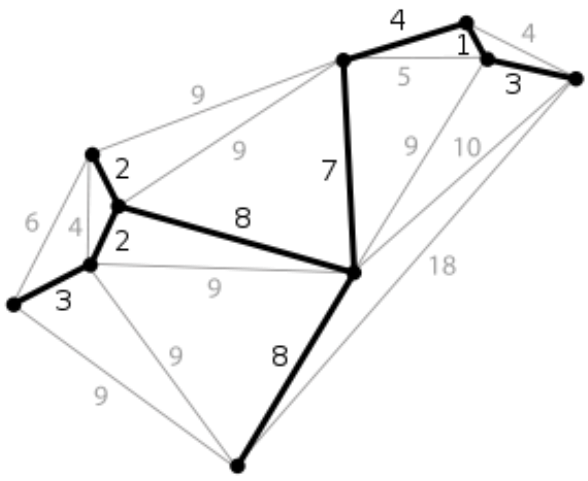


Figura 2

I problemi dell'albero a costo minimo generalmente possono essere di due tipi, diretti o indiretti. Nelle applicazioni dirette, normalmente si desidera collegare un insieme di punti minimizzandone le distanze e quindi i costi che ne derivano; molto

spesso infatti i nodi rappresentano entità fisiche che necessitano di essere collegate tra loro. Le applicazioni indirette a prima vista hanno poco a che fare con il MST, ma vengono poi ricondotte ad un problema di albero a costo minimo tramite modelli appropriati; un esempio di applicazione indiretta del MST è un modello per la verifica del grado di omogeneità nella composizione di un oggetto bimetallico.

Nella terminologia attuale il MST può essere formulato come segue:

dato un grafo G(V,E) indiretto, connesso e pesato, con *n* nodi, *m* spigoli, dove □(□, □) ∈ $\mathbb{R}^+$ è una funzione peso per G; trovare tra tutti gli alberi ricoprenti di G, l'albero T=V, E' tale per cui la somma dei pesi degli archi: □(□) = ∑(□(□)) ∀ □ ∈ □′, è minima. L'albero T è quindi l'albero con un sottoinsieme di nodi dai pesi minimi.

Algoritmo generico per il MST (GMST):

Il GMST è definito per un grafo G(V,E)

Dove V={1,…,*n*} è l'insieme dei vertici, E={(i,j): i, j ∈V i<j } l'insieme degli archi a cui è associata una matrice dei costi (c_{ij}) non negativa; V è formato dall'unione di K sottogruppi V_k dove k=1,…,N.

La formulazione secondo C. Feremans et al (2001) è la seguente:

Funzione obiettivo: $minimize \sum_{i<j} c_{ij} x_{ij}$ (6.1)

Soggetta ai vincoli:

$\sum_{i \in V_k} y_i \geq 1 \ \ (k = 1, \dots, K)$ (6.2)

$\sum_{i,j \in S, i<j} x_{ij} \leq \sum_{i \in S\{v\}} y_i \ \ (v \in S \subseteq V, |S| \geq 2)$ (6.3)

$\sum_{i,j \in V, i<j} x_{ij} = \sum_{i \in V} y_i - 1$ (6.4)

$x_{ij} \in \{0,1\} \qquad (i,j) \in E$ (6.5)

$y_i \in \{0,1\} \qquad i \in V$ (6.6)

Il vincolo (6.2) assicura che l'albero si estenda a tutti i sottogruppi, l'espressione (6.3) previene la generazione di cicli, mentre la (6.4) forza la soluzione a contenere tanti archi quanti sono i vertici meno uno.

I vincoli (6.5) e (6.6) invece implicano che x_{ij} e y_i siano variabili binarie tali per cui $x_{ij}=1$ se lo spigolo (i,j) appartiene alla soluzione, ed $y_i = 1$ se il nodo i appartiene alla soluzione.

Esistono diversi algoritmi esatti che risolvono il problema dell'albero a costo minimo in tempo polinomiale. Prima di esaminare i tre metodi classici di *Prim, Kruskal* e *Borůvka,* è necessario sottolineare che tutti i metodi conosciuti finora rispettano le seguenti due proprietà:

Cut Rule: la soluzione ottima T per il minimum spanning problem, contiene l'arco con il peso minimo in ogni taglio.

Circuit Rule: l'arco del circuito C, il cui peso è maggiore di quello degli altri archi rimanenti di C, non può appartenere alla soluzione ottima T.

6.2 Algoritmo di Prim

L'algoritmo di Prim è di particolare interesse, in quanto pur essendo un classico algoritmo costruttivo, ossia un algoritmo nel quale la soluzione viene ottenuta a partire da zero, attraverso una sequenza di soluzioni parziali che vengono via via completate, consente di ottenere la soluzione ottima del problema.

L'algoritmo viene inizializzato ponendo in S un primo nodo scelto a caso ed individua la soluzione ottima comunque si scelga il primo nodo.

Dato un grafo G=(V,E), dove V è l'insieme dei vertici o nodi ed E è l'insieme degli archi, ed un albero di soluzione S in cui porremo i nodi raggiunti nei vari passi dell'algoritmo procediamo nel seguente modo:

Pongo in S un nodo di partenza (arbitrario) dal quale poi sceglierò l'arco incidente di peso minimo non collegato a nodi facenti parte dell'albero di soluzione S.

Effettuata la scelta del ramo dal passo precedente includerò in S il vertice collegato al vertice di partenza dal ramo appena scelto.

Ad ogni vertice che includo in S, anche i rami incidenti di quel vertice si aggiungeranno ai rami, tra cui sceglierò quello meno costoso che collega ad un vertice non appartenente ad S.

L'algoritmo termina quando la cardinalità di S è pari a quella di V e ciò vale solo se il grafo è connesso.

Consideriamo ad esempio il problema descritto dalla matrice dei costi riportata nella tabella 6.1

	0	1	2	3	4	5
0	-	12	4	9	10	5
1	12	-	7	5	9	6
2	4	7	-	8	7	7
3	9	5	8	-	10	3
4	10	9	7	10	-	6
5	5	6	7	3	6	-

Tabella 6.1: matrice dei costi di collegamento

Lo sviluppo dell'algoritmo di Prim assumendo 0 come nodo iniziale è sintetizzato nella tabella seguente:

Iterazione	S	*z(S)*	S'	$c_{ik(i)}$						$c_{i^*k(i^*)}$	i^*	k(i*)
				0	1	2	3	4	5			
1	0	0	1,2,3,4,5	-	-	-	-	-	-	4	0	2
2	0,2	0+4=4	1,3,4,5	5	-	7	-	-	-	5	0	5
3	0,2,5	4+5=9	1,3,4	9	-	7	-	-	3	3	5	3
4	0,2,3,5	9+3=12	1,4	10	-	7	5	-	6	5	3	1
5	0,1,2,3,5	12+5=17	4	10	-	7	10	-	6	6	5	4
6	0,1,2,3,4,5	17+6=23	-									

Tabella 6.2: sviluppo dell'algoritmo di Prim

E’ possibile vedere l’evoluzione dell’algoritmo anche in figura 2.

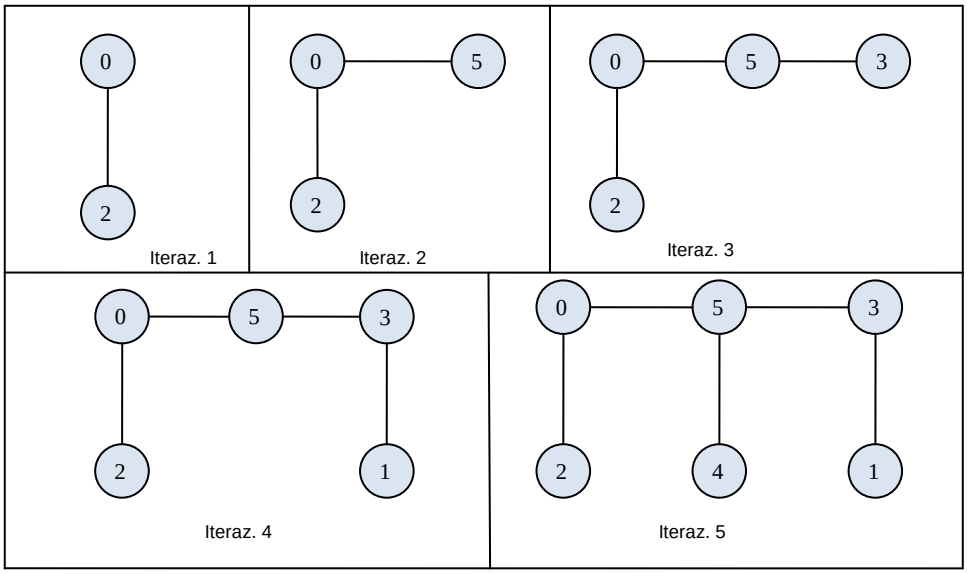


Figura 3 Evoluzione della soluzione con Prim

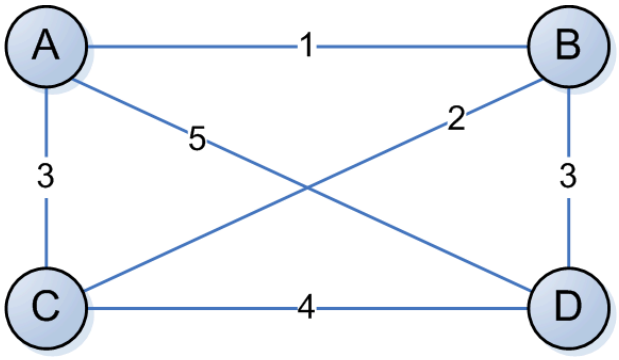
E’ dimostrato, (C. Martel, 2002), che se si inizializza l’algoritmo ad un grafo arbitrario e si assegna casualmente un peso ad ogni arco, il tempo necessario all’algoritmo di Prim per trovare l’ottimo è dato da una funzione del tipo: O*(m + n* log *n* log*(*1 + *m/n))*.

6.3 Algoritmo di Kruskal

L’algoritmo di Kruscal, (1956), si basa su un concetto molto semplice: ordinare gli archi in base ai costi e successivamente esaminarli in questo ordine, inserendoli man mano nella soluzione che stiamo costruendo, a patto che essi non formino cicli con archi precedentemente selezionati.

Si può notare che ad ogni passo, se abbiamo più archi con lo stesso costo, è indifferente quale viene scelto.

Esaminiamo un grafo composto da 4 nodi (A, B,C, D) e 6 archi di collegamento aventi una certa distanza tra loro nota (figura3) ; prima di procedere alla costruzione dell’albero a costo minimo è necessario ordinare gli archi in base alla loro distanza in modo crescente: si otterrà la tabella 6.3.



AB	1
BC	2
AC	3
BD	3
CD	4
AD	5

Tabella 6.3

Procediamo evidenziando il lato AB che è quello con distanza minore

AB	1
BC	2
AC	3
BD	3
CD	4
AD	5

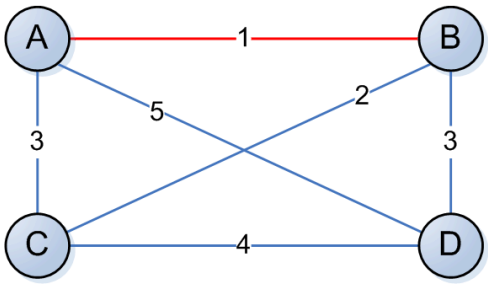
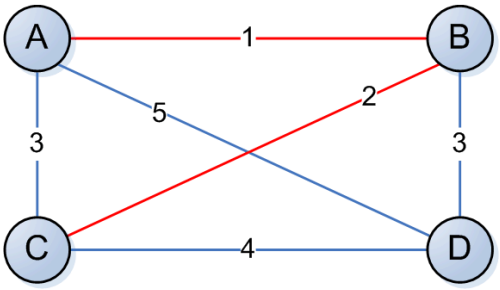


Figura 4

Successivamente si passa all'arco BC e poi a BD



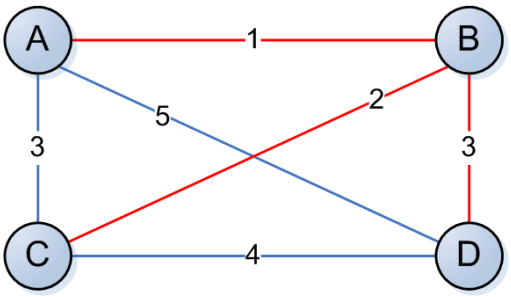


Figura 5

Figura 6

Una volta selezionati tutti i nodi il problema è concluso.

Si nota che se avessimo scelto AC al posto di BD si sarebbe formato un circuito chiuso, contro le ipotesi formulate.

Il costo totale della rete è dato da: AB+BC+BD = 1+2+3= 6

L'algoritmo di Kruscal si basa essenzialmente sull'algoritmo generico per la creazione di un Minimum Spanning Tree, di cui sfrutta il corollario collegato al teorema di ricerca degli archi sicuri (Robert E. Tarjan, 1983).

6.4 Algoritmo di Borůvka

fu pubblicato nel 1926 da Otakar Borůvka al quale venne chiesto di ideare un metodo efficiente per la distribuzione dell'energia elettrica nella Moravia.

L'algoritmo consente di trovare l'albero a costo minimo di un grafo a patto che ogni arco abbia un peso ben distinto, nel caso in cui due spigoli abbiano lo stesso peso l'algoritmo non è applicabile, a meno di non variare anche solo leggermente il peso di uno dei due.

Il procedimento ha inizio esaminando ciascun vertice ed aggiungendo lo spigolo di costo minore da quel vertice ad un altro nel grafo, senza considerare gli archi già aggiunti precedentemente, e continua unendo questi raggruppamenti in modo simile finchè si completa un albero ricoprente tutti i vertici.

Lo pseudocodice per l'algoritmo è il seguente:

1. Inizialmente si ha un grafo connesso G contenente archi di peso diverso e un insieme di archi vuoto T;
2. Finché i vertici di G connessi da T sono disgiunti :
   2.1. Inizializza un insieme vuoto di archi E
   2.2. Per ciascun componente:
      2.2.1. Inizia con un insieme di archi vuoto S
      2.2.2. Per ciascun vertice del componente:
         2.2.2.1. Aggiungi ad S l'arco di costo minore dal vertice del componente verso un altro vertice di un componente disgiunto;
      1.2.3 Aggiungi l'arco di costo minore di S ad E;
   1.3 Aggiungi l'insieme di archi risultante da *E* a *T*.
3. L'insieme di archi T risultante è il Minimum Spanning Tree del grafo G.

E' dimostrato che l'algoritmo di Borůvka ha una complessità computazionale O(*E* log *V*), dove *E* è il numero di archi, e *V* è il numero di vertici in *G*.

Si possono ottenere algoritmi più veloci combinando l'algoritmo di Prim e quello di Borůvka. Una versione randomizzata più veloce realizzata da Karger, Klein, e Tarjan lavora in tempi asintotici lineari nel numero di archi.

Il più conosciuto algoritmo di calcolo del minimo albero ricoprente è stato realizzato da B.Chazelle si basa su Borůvka e lavora in tempo O(*E* α(V)), dove α è l'inverso della Funzione di Ackermann.

6.5 L'albero a costo minimo con vincoli di capacità

In molti problemi del progetto di reti, soprattutto nel campo delle comunicazioni, si ha l'obiettivo di collegare un nodo centrale con un insieme di nodi terminali caratterizzati da una domanda di traffico verso il nodo centrale.

Tra ogni coppia di nodi, compreso il nodo centrale, è possibile inserire un collegamento cui è associato un costo di localizzazione.

Il problema dell'albero a costo minimo con vincoli di capacità consiste nell'individuare l'albero a costo minimo che connette il nodo centrale ai nodi terminali, in modo che il traffico su ciascun collegamento sia inferiore ad un prefissato valore di capacità Q.

Per descrivere il problema, si consideri un grafo completo, non orientato G(V,A) con V={1,2,…,n} insieme dei vertici a ciascuno dei quali è associata una domanda di traffico d_i che deve essere indirizzata verso il nodo centrale "0" detto anche *radice*.

Si definisca inoltre, una matrice dei costi r_{ij} simmetrica il cui elemento generico rappresenta il costo di collegamento tra i nodi *(i,j)*.

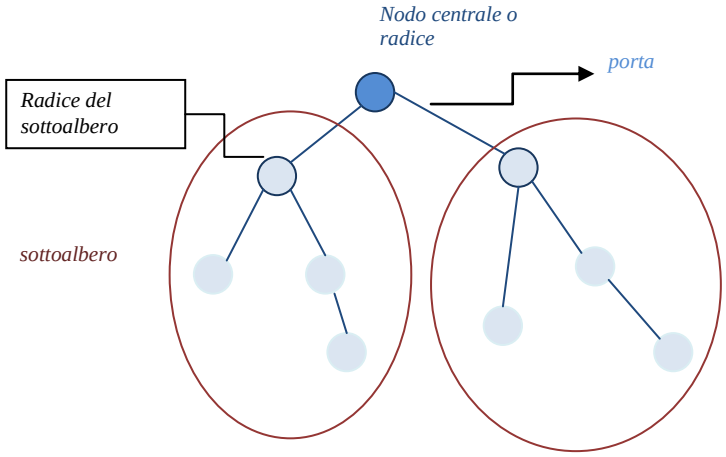
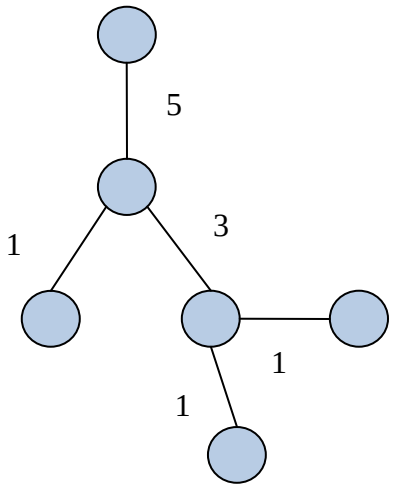


Figura 7 Sottoalberi, radici e porte di un albero

Se si considera un albero e un nodo di esso come radice, è facile dimostrare che, eliminando gli spigoli incidenti nella radice, si individua un insieme di *m* sottoalberi $T_1,\ldots,T_m$. Gli spigoli che connettono ciascun sottoalbero alla radice sono detti *porte*. I nodi direttamente connessi alla radice, sono, a loro volta, le radici dei sottoalberi.



Si dice che un sottoalbero T_i è ammissibile se la somma delle domande associate ai suoi nodi non supera il valore della capacità massima.

$$\sum_{k \in T_j} d_k \leq Q$$

Infatti si può dimostrare che, se un sottoalbero è ammissibile, il traffico su ciascun spigolo di esso non può superare la capacità massima: infatti, la distribuzione di traffico verso la radice è tale per cui ,per ogni sottoalbero, i flussi maggiori si riscontrano lungo la porta dove si concentra un flusso pari alla somma delle domande provenienti da ciascun nodo.

Per tale ragione, per soddisfare il vincolo di capacità su tutti gli spigoli, è sufficiente garantire l'ammissibilità di ciascun sottoalbero.

Sulla base di queste considerazioni, il problema dell'*albero minimo con vincoli di capacità* (*Capacitated Minimum Spanning Tree*) può essere visto come un problema di partizione dell'insieme dei vertici V-{ 0 } in sottoalberi ammissibili.

E' un problema NP-hard.

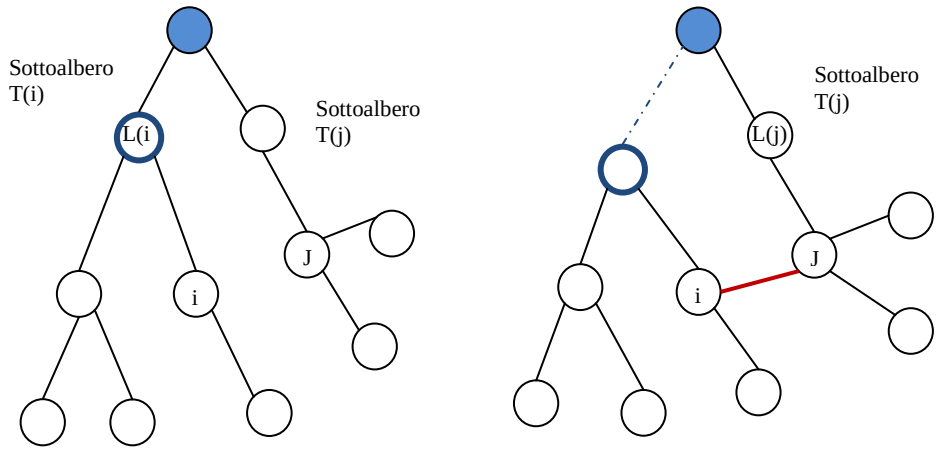
Algoritmo di Esau e Williams

Un algoritmo euristico migliorativo in grado di risolvere efficacemente il Capacitated Minimum Spanning Tree è quello di Esau e Williams (EW) che si basa su una mossa *cut and paste,* ossia taglia-incolla.

Si tratta di un'euristica estremamente semplice per quanto riguarda i suoi requisiti di calcolo ed ha una complessità temporale di O($n^2 \log n$). Come conseguenza, l'algoritmo EW viene incorporato in molte altre meta-euristiche ed utilizzato come procedura ausiliaria nelle Local Search.

L'euristica può essere sintetizzata come segue.

Si consideri una soluzione ammissibile del problema e due nodi *i* e *j* appartenenti a due sottoalberi diversi. Si indichino con T(i) e T(j) i sottoalberi cui appartengono rispettivamente i nodi i e j, mentre L(i) e L(j) sono le radici di questi sottoalberi.



Sottoalberi di partenza

Sottoalbero dopo l'operazione di cut and paste

Operazione di cut and paste tra i e j

L'operazione di cut and paste tra i nodi i e j consiste nella cancellazione (cut) dello spigolo (L(i), 0) che rappresenta la porta di T(i), e nell'unione (paste) di T(i) e T(j) aggiungendo l'arco (i,j) (Vedi figura).

Perché la nuova soluzione ottenuta risulti migliore della precedente, devono essere garantite le seguenti condizioni:

$\sum_{s\in T(i)} d_s + \sum_{s\in T(j)} d_s \leq Q$
(6.7);

$r_{ij} < r_{L(i),0}$
(6.8)

```
function CMST(c,C,r):
    T = {c_1r, c_2r, ..., c_nr}
    while have changes:
        for each node a_i
            a_i = closest node in a different subtree
            t(a_i) = g_i - c_ij
        t_max = max(t(a_i))
        k = i such that t(a_i) = t_max
        if ( cost(i) + cost(j) <= c)
            T = T - g_k
            T = T union c_kj
    return T
```

Algoritmo di Esau e Williams in pseudocode

La prima verifica che il sottoalbero formatosi a seguito dell'operazione di cut and paste risulti ammissibile, il vincolo (6.8) invece assicura che la nuova soluzione presenta un valore di funzione obiettivo migliore della precedente.

Sulla base dell'espressione (6.8), ad ogni potenziale collegamento (i,j) si può associare un *savings* (risparmio) $s_{ij} = r_{L(i),0} - r_{ij}$, che rappresenta la variazione di funzione obiettivo qualora si effettua il cut and paste tra i e j. Si nota che, in generale, $s_{ij} \neq s_{ji}$.

L'algoritmo di Esau e Williams parte da una soluzione cosiddetta "a stella", ossia in cui tutti i nodi sono direttamente collegati alla radice e ordina i valori di s_{ij} in senso decrescente. Viene esaminata, nell'ordine, la lista dei savings e si effettuano le operazioni di cut and paste tra i e j, sempre che siano rispettate le condizioni (6.7) e (6.8) e che non si formino cicli, poiché in tal caso i e j apparterrebbero allo stesso sottoalbero.

Effettuato un cut and paste si aggiornano gli s_{ij} e si considera il valore di s_{ij} successivo della lista. L'aggiornamento dei savings si rende necessario per il fatto che, dopo un'operazione di cut and paste, i nodi del sottoalbero T(i) presentano una porta diversa, quindi diversi valori di savings.

La procedura termina quando non esistono più s_{ij} positivi.

Formulazione dell'algoritmo:

1. *Inizializzazione*

   Si costruisce una prima soluzione S "a stella" connettendo tutti i nodi 1,…, n alla radice 0; si pone $z(S) = \sum_i r_{i0}$ e si inizializzano le etichette L(j) ponendo $\mathrm{L(j)} = \mathrm{j} \forall\ \mathrm{j}$.

2. *Calcolo dei savings*

   Si calcola la matrice dei savings nel seguente modo:

$$s_{ij} = \begin{cases} -\infty \quad se \sum_{s \in T(i)} d_s + \sum_{s \in T(j)} d_s > Q \ oppure \ L(i) = L(j) \\ r_{L(i),0} - r_{ij} \qquad\qquad altrimenti \end{cases}$$

   La condizione L(i)=L(j) attesta che i e j appartengono allo stesso sottoalbero.

3. *Criterio di arresto*

   Se $s_{ij} \leq 0 \ \forall\ (i,j)$ l'algoritmo si arresta.

   Altrimenti si individua $s_{i^*j^*} = \max s_{ij}$ e si effettua l'operazione di cut and paste tra i^* e j^* aggiungendo l'arco (i^*, j^*) ed eliminando l'arco $(L(i^*),0)$. Si pone allora $\mathrm{L(k)} = \mathrm{L(j^*)} \ \forall\ \mathrm{k} \in \mathrm{T(i^*)}$ e $\mathrm{z(S)} = \mathrm{z(S)} - \mathrm{s_{i^*j^*}}$.

La posizione $\mathrm{L(k)} = \mathrm{L(j^*)} \ \forall\ \mathrm{k} \in \mathrm{T(i^*)}$ indica che, poiché il sottoalbero T(i*) è stato unito a T(j*), i suoi nodi presentano la stessa radice L(j*).

Esempio di problema di albero a costo minimo con vincoli di capacità.

Facendo riferimento al vettore di domanda e alla matrice dei costi di collegamento, rispettivamente tabelle 6.4 e 6.5, cerchiamo di individuare l'albero a costo minimo con vincoli di capacità, ad esempio ponendo Q=6.

Nodo i	1	2	3	4	5
D_i	4	1	2	3	2

Tabella 6.4: Vettore della domanda generata dai nodi

	0	1	2	3	4	5
0	-	12	4	9	10	5
1	12	-	7	5	9	6
2	4	7	-	8	7	7
3	9	5	8	-	10	3
4	10	9	7	10	-	6
5	5	6	7	3	6	-

Tabella 6.5: Matrice dei costi di collegamento

Sviluppiamo l'algoritmo di Esau e Williams.

1. *Inizializzazione*

   Si connettono tutti i nodi alla radice "0" ottenendo la soluzione a stella cui corrisponde z=40.

   Si riportano in tabella 6.6 i valori di L(i), $r_{L(i),0}$ e $\sum_{s\in T(i)} d_s$

Nodo i	1	2	3	4	5
L(i)	1	2	3	4	5
$r_{L(i),0}$	12	4	9	10	5
$\sum_{s\in T(i)} d_s$	4	1	2	3	2

Tabella 6.6

2. *Calcolo dei savings s_{ij}*

   I savings si calcolano mediante la relazione $s_{ij} = r_{L(i),0} - r_{ij}$; quindi fissata la riga i, gli s_{ij} si ricavano sottraendo $r_{L(i),0}$ ai valori della riga i.

   Se il cut and paste tra i e j produce una violazione del vincolo di capacità (vedi l'espressione 6.7), si pone il valore -∞.

	1	2	3	4	5	
1	-∞	5	7	-∞	6	r_{10}-r_{1j}=12- r_{1j}
2	-∞	-∞	-4	-3	-3	r_{20}- r_{2j}=4- r_{2j}
3	4	1	-∞	-1	6	r_{30}- r_{3j}=9- r_{3j}
4	-∞	3	0	-∞	4	r_{40}- r_{4j}=10- r_{4j}
5	-1	-2	2	-1	-∞	r_{50}- r_{5j}=5- r_{5j}

Tabella 6.7: calcolo dei savings

3. *Criterio di arresto*

Il risparmio massimo si ha per s_{13} =7, per cui si ottiene i^*=1 e j^*=3;
si effettua quindi un cut and paste tra 1 e 3 aggiungendo lo spigolo (1,3) e cancellando (1,0).
Si ottiene così la soluzione z=40-7 = 33.
Avendo posto L(1)=L(3)=3, vanno modificati i valori della tabella 6.6

Nodo i	1	2	3	4	5
L(i)	**3**	2	**3**	4	5
$r_{L(i),0}$	**9**	4	**9**	10	5
$\sum_{s\in T(i)} d_s$	**6**	1	**6**	3	2

Tabella 6.6 (2) aggiornata

Poiché non abbiamo ancora raggiunto una soluzione finale, si riparte dal punto 2.

2. *Calcolo dei savings s_{ij}*

	1	2	3	4	5	
1	-∞	-∞	-∞	-∞	-∞	r_{30}-r_{1j}=9- r_{1j}
2	-∞	-∞	-∞	-3	-3	r_{20}- r_{2j}=4- r_{2j}
3	-∞	-∞	-∞	-∞	-∞	r_{30}- r_{3j}=9- r_{3j}
4	-∞	3	-∞	-∞	4	r_{40}- r_{4j}=10- r_{4j}
5	-∞	-2	-∞	-1	-∞	r_{50}- r_{5j}=5- r_{5j}

3. *Criterio di arresto*

s_{ij} max = s_{45} = 4 quindi i^*=4 e j^*=5.
Si effettua un cut ad paste tra 4 e 5 aggiungendo lo spigolo (4,5) e cancellando lo spigolo (4,0); si ottiene così la soluzione z=33-4 = 29.
Avendo posto L(4)=L(5) aggiorniamo la tabella 6.6:

Nodo i	1	2	3	4	5
L(i)	3	2	3	**5**	**5**
$r_{L(i),0}$	9	4	9	**5**	**5**
$\sum_{s\in T(i)} d_s$	6	1	6	**5**	**5**

Tabella 6.6 (3)

2. *Calcolo dei savings s_{ij}*

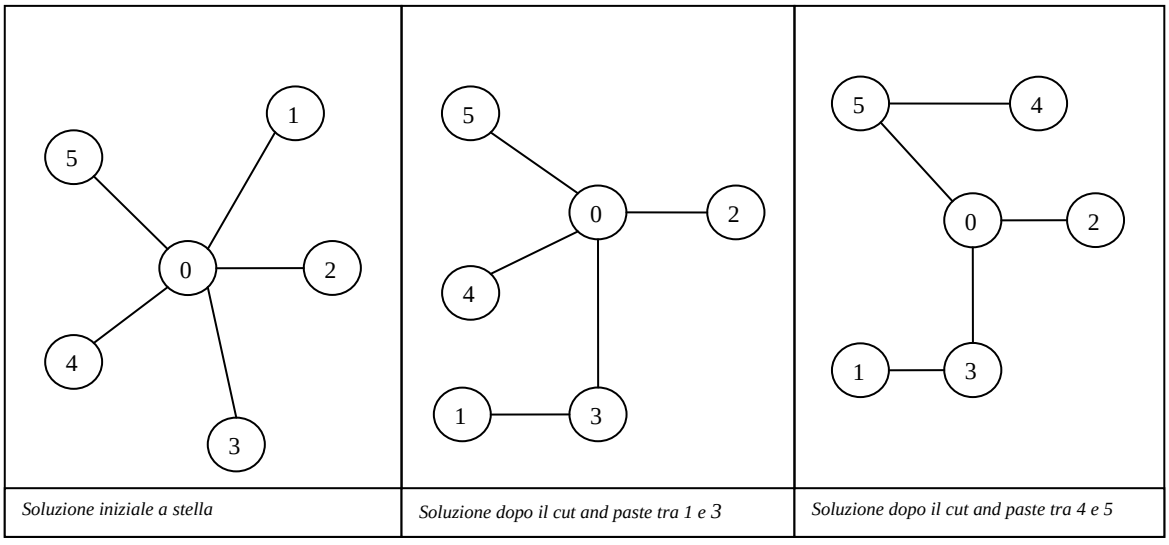
	1	2	3	4	5	
1	-∞	-∞	-∞	-∞	-∞	r_{30}-r_{1j}=9- r_{1j}
2	-∞	-∞	-∞	-3	-3	r_{20}- r_{2j}=4- r_{2j}
3	-∞	-∞	-∞	-∞	-∞	r_{30}- r_{3j}=9- r_{3j}
4	-∞	-2	-∞	-∞	-∞	r_{50}- r_{4j}=5- r_{4j}
5	-∞	-2	-∞	-1	-∞	r_{50}- r_{5j}=5- r_{5j}

3. *Criterio di arresto*

Poiché i savings sono tutti negativi, l'algoritmo si arresta.

La soluzione finale perciò è z= 29.

Infine in figura 7 possiamo vedere la rappresentazione grafica dell'evoluzione della soluzione mediante l'algoritmo di Esau e Williams.



Soluzione iniziale a stella | *Soluzione dopo il cut and paste tra 1 e 3* | *Soluzione dopo il cut and paste tra 4 e 5*

Evoluzione della soluzione

CAPITOLO 7

Travelling Salesman Problem

Spesso il cammino minimo da *s* a *t* deve soddisfare anche la richiesta di passare per un insieme prefissato di nodi. Se la richiesta riguarda solo un particolare nodo *k* il problema si risolve facilmente con due problemi di cammino minimo, uno da *s* a *k* e il secondo da *k* a *t*.

Se invece la richiesta riguarda due nodi *k* e *h*, bisogna risolvere 6 problemi di cammino minimo, precisamente $s \rightarrow k$, $s \rightarrow h$, $h \rightarrow k$, $k \rightarrow h$, $k \rightarrow t$, $h \rightarrow t$, e valutare se il cammino $s \rightarrow k \rightarrow h \rightarrow t$ è più conveniente del cammino $s \rightarrow h \rightarrow k \rightarrow t$.

Normalmente il vincolo di inclusione riguarda tutti i nodi e il problema viene quasi sempre formulato come circuito piuttosto che come cammino.

Si tratta del celebre *Problema del commesso viaggiatore*, TSP (*Traveling Salesman Problem*).

7.1 Introduzione al TSP

Il nome nasce dalla sua più tipica rappresentazione: data una rete di città, connesse tramite delle strade, trovare il percorso di minore lunghezza che un commesso viaggiatore deve seguire per visitare tutte le città una e una sola volta.

Espresso nei termini della teoria dei grafi è così formulato: dato un grafo completo pesato, trovare il ciclo hamiltoniano con peso minore. La rete di città può essere rappresentata come un grafo in cui le città sono i nodi, le strade gli archi e le distanze i pesi sugli archi.

Può essere dimostrato che specificare o meno il ritorno alla città di partenza non cambia la complessità computazionale del problema.

Il problema è di considerevole importanza pratica, al di là delle ovvie applicazioni nella logistica e nei trasporti.

La più grande istanze di TSP risolta tuttora, (85900 nodi) proviene dal'applicazione del TSP nella progettazione di circuiti stampati,più specificatamente nella pianificazione del percorso del trapano per creare i fori nella piastra. Se invece si considerano le applicazioni di foratura o di rifinitura automatica eseguite da robot, le "città" sono pezzi da rifinire o fori (anche di varie dimensioni) da praticare, e il "costo di viaggio" include i tempi morti (ad esempio il tempo che il robot impiega, se necessario, per cambiare la punta con cui lavora).

Il problema è Np-hard e quindi dobbiamo adottare delle strategie opportune di calcolo, ma anche così i tempi di calcolo possono essere elevati e bisogna spesso ricorrere a procedure euristiche che forniscono una risposta non necessariamente ottima, ma in tempi di calcolo accettabili.

Il TSP riveste un ruolo centrale nei problemi di ottimizzazione combinatoria ed è stato oggetto di ricerche molto approfondite.

7.2 *Cammini e circuiti hamiltoniani*

Il TSP è Strettamente connesso al problema di determinare un circuito hamiltoniano. Normalmente nel TSP il grafo è completo e ad ogni arco è assegnato un costo.

Si tratta quindi di determinare un circuito hamiltoniano di costo minimo.

Diamo innanzitutto la definizione di circuito hamiltoniano:

Dato un grafo G = (N,E), un circuito (o un cammino) si dice hamiltoniano se contiene tutti i nodi esattamente una volta. Un grafo si dice hamiltoniano se nel grafo esiste un circuito hamiltoniano.

Il TSP non è più facile del problema del circuito hamiltoniano (anche se il grafo è completo). Infatti disponendo di un algoritmo che risolve il TSP, basta, dato un grafo qualsiasi, assegnare costo 0 agli archi del grafo, rendere completo il grafo aggiungendo tutti gli archi necessari e assegnare costo 1 a questi archi.

Il grafo è hamiltoniano se e solo se il problema del TSP ha valore ottimo 0.

Una variante del TSP è quella formulata su un grafo non necessariamente completo e viene chiesto un circuito che passa per tutti i nodi *almeno* una volta.

Questo problema viene facilmente trasformato in un TSP definendo un grafo completo sugli stessi nodi, ma con archi corrispondenti al cammino minimo fra i due nodi nel grafo originario. Quindi il circuito hamiltoniano nel grafo completo viene poi interpretato come una successione di cammini nel grafo originario (e quindi vi possono essere nodi ripetuti, nonché archi percorsi avanti e indietro). Per essere risolto il problema richiede preliminarmente il calcolo dei cammini minimi fra tutte le coppie di nodi.

Un'ulteriore variante del TSP prevede un nodo speciale (detto deposito) e una soluzione consistente in uno o più circuiti che insieme coprono tutti i nodi, passano tutti per il deposito e negli altri nodi sono disgiunti.

7.3 TSP Simmetrico

Analizziamo ora il problema del TSP simmetrico.

Dato un grafo completo di n nodi con costi c_e per ogni arco $e \in E$. Si vuole determinare un circuito hamiltoniano di costo minimo.

Il problema può esser formulato efficacemente, definendo variabili $x_e \in \{0,1\}$ per ogni $e \in E$. Ad ogni vettore di questo genere viene naturalmente associato un sottoinsieme di archi.

Bisogna quindi determinare degli opportuni vincoli in modo che gli unici sottoinsiemi soddisfacenti i vincoli siano tutti e soli i circuiti hamiltoniani.

Un primo vincolo è dato dal fatto che il sottoinsieme deve avere grado 2 in ogni nodo, vincolo soddisfatto da ogni circuito hamiltoniano.

Quindi

$$\sum_{e \in \delta(i)} x_e = 2$$

Tuttavia vi sono sottoinsiemi che soddisfano il vincolo senza essere circuiti hamiltoniani, come ad esempio qualsiasi insieme di circuiti. Quindi bisogna trovare un vincolo che escluda sottoinsiemi formati da più circuiti disgiunti.

Si può notare che ogni taglio del grafo è attraversato da almeno due archi (in generale è attraversato da un numero pari di archi). Quindi, per ogni sottoinsieme proprio di nodi S deve valere:

$$\sum_{e \in \delta(S)} x_e \geq 2$$

Tali diseguaglianze prendono il nome di *diseguaglianze di sottocircuito* (*sub tour inequalities*). Gli unici sottoinsiemi che soddisfano sia (7.2) che (7.3) sono i circuiti hamiltoniani. Quindi il TSP potrebbe essere formulato nel seguente modo:

Funzione Obiettivo (7.1)

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

Soggetta ai vincoli:

$\sum_{e \in \delta(i)} x_e = 2 \qquad i \in N$ (7.2)

$\sum_{e \in \delta(S)} x_e \geq 2 \qquad S \subset N$ (7.3)

$x_e \in \{0,1\}$ (7.4)

Il numero di vincoli in (7.3) è esponenziale e il modo per risolvere il rilassamento di (7.3) consiste nel generare solo quei vincoli che si dimostrano necessari. Inizialmente si eliminano del tutto i vincoli (7.2) e si risolve

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

$\sum_{e \in \delta(i)} x_e = 2 \qquad i \in N$ (7.5)

$0 \leq x_e \leq 1$ (7.6)

Se la soluzione è un circuito hamiltoniano il problema è risolto. Altrimenti si individua una diseguaglianza violata in (7.3) che viene aggiunta a (7.5) e si procede così fino ad ottenere un circuito hamiltoniano oppure una soluzione frazionaria che non viola nessuno dei vincoli (7.2) e (7.3). Quindi si tratta di risolvere in sequenza i seguenti problemi di PL

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

$$\sum_{e \in \delta(i)} x_e = 2 \qquad i \in N$$

$$\sum_{e \in \delta(S_k)} x_e \geq 2 \qquad k \in [q]$$

$$0 \leq x_e \leq 1$$

per $q := 0, 1, \ldots$, dove $S_1, S_2 \ldots$, sono sottoinsiemi di nodi generati in modo che la soluzione ottima $\hat{x}^q$ del problema q-esimo nella sequenza violi il vincolo successivo, cioè

$$\sum_{e \in \delta(S_{q+1})} \hat{x}_e^q < 2$$

Questo garantisce $v_0 \leq v_1 \leq \ldots$, e quindi di ottenere limitazioni inferiori sempre migliori. Ovviamente se, per un particolare problema, la soluzione $\hat{x}^q$ è un circuito hamiltoniano, questa soluzione è necessariamente ottima.

7.3 Euristiche per il TSP

IL TSP è forse il problema a cui è stata applicata ogni tecnica risolutiva, quasi un termine di paragone per valutare la bontà dei metodi proposti.

La local search dà dei risultati soddisfacenti per il TSP. Un semplice tipo d'intorno si ottiene togliendo da un circuito hamiltoniano due archi arbitrari non adiacenti e "riattaccando" i due spezzoni di circuito nell'altro senso.

Più esattamente, se gli archi che vengono tolti sono (*i, j*) e (*h, k*), con *i* e *h* appartenenti alla stessa componente connessa del circuito dopo la rimozione degli archi, gli archi che si aggiungono sono (*i, k*) e (*j, h*).

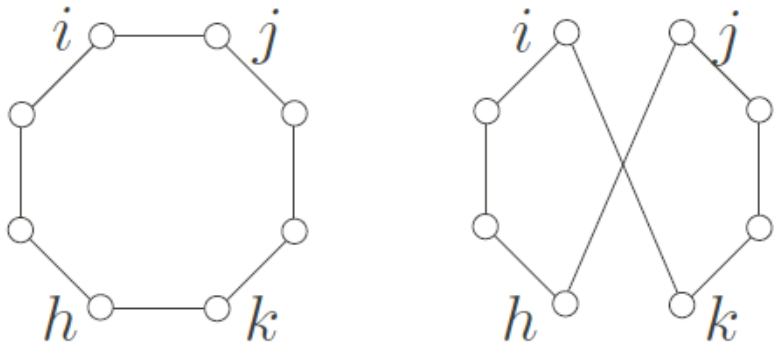


Figura 8

In questo modo un intorno contiene n (n − 3)/2 soluzioni. Intorni più grandi si ottengono togliendo tre archi non adiacenti e aggiungendone tre in modo da formare un circuito. In generale se si tolgono k archi non adiacenti rimangono k pezzi di circuito che si possono riattaccare assieme secondo (k − 1)! permutazioni e orientare in due modi. Tenendo conto che ogni soluzione compare due volte (anche nel verso opposto) vi sono (k −1)! 2k−1 −1 soluzioni nell'intorno.

Vi è un aspetto favorevole in queste definizioni di intorno e riguarda la possibilità di confrontare rapidamente due soluzioni x e y senza dover valutare $f(x)$ e $f(y)$. Nel caso del TSP, infatti, il calcolo di $f(x)$ ha complessità $O(n)$ perché richiede la somma di n costi.

Bisogna comunque tener presente che una ricerca esaustiva del primo intorno costa $O(n2)$ e del secondo costa $O(n3)$. Ad esempio, se n = 100, nel primo intorno si hanno 4.850 soluzioni, mentre nel secondo ve ne sono 1.064.000 soluzioni, valore troppo elevato per una ricerca esaustiva dell'intorno. Se n è dell'ordine delle migliaia bisogna eseguire una ricerca parziale nell'intorno e accettare la prima soluzione migliore.

E' stato visto empiricamente per il TSP che i miglior intorni sono quelli in cui si opera con profondità variabile. L'euristica di Lin e Kernigham (1973) dà ottimi risultati. Sotto opportune ipotesi si è visto che la probabilità che un ottimo locale sia globale è attorno al 5%. Quindi la migliore soluzione fra 100 soluzioni ottenute da altrettante soluzioni iniziali diverse ha una probabilità di essere ottima globale pari a 1−0.95100=1− 0.0059 cioè più del 99%.

L'euristica è stata migliorata da Helgsaun (2000).

Per il TSP sono molti i metodi greedy proposti e nessuno di essi si è dimostrato superiore agli altri. Tipicamente la soluzione viene costruita aggiungendo un nodo alla volta secondo regole diverse:

selezionare il nodo più vicino all'ultimo nodo inserito;

dato un circuito parziale, selezionare un nodo a caso e inserirlo fra la coppia più favorevole del circuito parziale; si inizia con un circuito a caso di tre nodi;

dato un circuito parziale, selezionare il nodo più vicino al circuito e inserirlo fra la coppia più favorevole del circuito parziale;

dato un circuito parziale, selezionare il nodo più lontano al circuito e inserirlo fra la coppia più favorevole del circuito parziale.

Conclusioni

IL problema della localizzazione ed il progetto di reti sono di notevole importanza, date le numerose applicazioni pratiche, in diversi ambiti, che si basano su esso. Pertanto sono molti gli algoritmi sviluppati dagli inizi del ‘900 ad oggi, che si propongono di trovare soluzioni sempre più ottimali in tempi più ragionevoli.

Nel Capitolo 2 è stato presentato il Simple Plant Location Problem, questo approccio non tiene conto di limitazioni di capacità, ma si limita a scegliere quali facilities attivare tra un insieme di possibili candidati. Sulla base del SPLP sono stati proposti altri modelli, ognuno di essi presenta alcune caratteristiche che lo distinguono dagli altri: ad esempio nel SPLPO si tiene conto dell’ordine di preferenza dei clienti per determinati impianti, nel SPLPG si rappresentano i costi di installazione delle strutture tramite una generica funzione, piuttosto che adottare un costo fisso, nel SPLPC si considerano costi di trasporto non lineari, ecc.

IL P-median Problem invece, consiste nel determinare, fissato un numero *p* di localizzazioni, i nodi nei quali è possibile localizzare i servizi minimizzando i costi di afferenza, in quanto i costi di localizzazione sono ipotizzati costanti e quindi trascurabili ai fini dell’ottimizzazione. Ad esso si riconducono varie euristiche, sia classiche che meta-euristiche, quali la Tabù Search, il Simulated Annealing e gli Algoritmi Genetici.

Anche per il progetto di reti sono stati analizzati vari procedimenti che consentono di individuare il cammino minimo di un grafo di *n* nodi; si parla dunque di Albero a costo minimo. Un esempio è il Travelling Salesman Problem, in cui il circuito minimo deve passare per tutti i nodi del grafo.

Da questa analisi è possibile dedurre che non esiste un algoritmo risolutivo in assoluto superiore agli altri, o che mi garantisca la soluzione ottima al 100%, in quanto i problemi sono di tipo NP-hard. Ad esempio, nel problema del Commesso Viaggiatore, spesso la soluzione viene costruita partendo da un nodo iniziale e man mano aggiungendone altri, secondo regole diverse.

Le varie euristiche, infatti, vengono rivisitate in continuazione, migliorate ed adattate alle singole necessità del caso considerato.