

## Capitolo 8

# Euristiche per la soluzione di problemi di ottimizzazione combinatoria

Il termine *Euristica*, dal greco *euristikein* = scoprire, indica un metodo (algoritmico) per la ricerca di soluzioni ammissibili (non necessariamente ottime) di un problema di ottimizzazione. Se i vincoli o la funzione obiettivo del problema hanno una qualche particolare struttura si è talvolta in grado di sviluppare algoritmi efficienti che trovano la soluzione ottima in tempi ragionevoli anche per problemi di grande dimensione. Esempi di siffatti problemi sono il problema del cammino minimo in un grafo e il problema del massimo flusso. In molti casi, tuttavia, non sono noti metodi efficienti, e ci si deve accontentare di applicare metodi generali quali ad esempio il branch-and-bound. Purtroppo, quando la dimensione del problema cresce, il metodo del branch-and-bound potrebbe richiedere la generazione di un numero troppo elevato di sottoproblemi, e quindi non convergere in tempo ragionevole.

Per problemi di questo tipo, all'aumentare delle dimensioni ci si deve spesso accontentare di metodi che non assicurano l'ottimalità delle soluzioni trovate - le euristiche, appunto - ma che producono soluzioni di "qualità" ragionevole. In generale, se  $Z^*$  è il valore della soluzione ottima del problema dato (supposto di minimizzazione, per fissare le idee) e  $Z^E$  è il valore della soluzione trovata dalla nostra euristica, sarà  $Z^E \geq Z^*$ . Il punto cruciale è che la soluzione euristica *potrebbe* essere ottima (ovvero  $Z^E = Z^*$ ), ma noi non siamo in grado di dimostrarne l'ottimalità. Questa è la differenza sostanziale fra metodi euristici e metodi esatti (come il branch-and-bound): se il metodo esatto ha il tempo sufficiente a terminare le sue iterazioni, la soluzione finale è una soluzione ottima.

Alcuni tipi di euristiche forniscono un risultato *garantito* nel senso che è possibile dimostrare che la soluzione prodotta ha un valore che è al più  $\alpha$  volte la soluzione ottima (con  $\alpha > 1$ ), ovvero  $Z^E \leq \alpha Z^*$ .

Più spesso il risultato delle euristiche non è garantito, e la soluzione prodotta potrebbe avere un valore molto maggiore della soluzione ottima. Per poter valutare la qualità di un'euristica per un problema dato, i ricercatori hanno messo a punto un metodo detto di *benchmarking*. Si tratta semplicemente di utilizzare delle librerie di istanze del problema considerato per le quali sono noti i risultati ottenuti dalle altre euristiche proposte nella letteratura, e verificare come si posizionano rispetto ad essi i risultati prodotti dalla nuova euristica.

Di seguito ci occuperemo di euristiche per una classe particolare di problemi di ottimizzazione, introdotta nel prossimo paragrafo, e che comprende numerosissimi esempi di problemi reali.

## 8.1 L'ottimizzazione combinatoria

Una particolare classe di problemi di ottimizzazione è rappresentata dai cosiddetti *problemi di ottimizzazione combinatoria*. A questa classe appartengono, fra l'altro, i problemi di PL01, il problema dell'accoppiamento e il problema del cammino minimo su grafi visti nei capitoli precedenti.

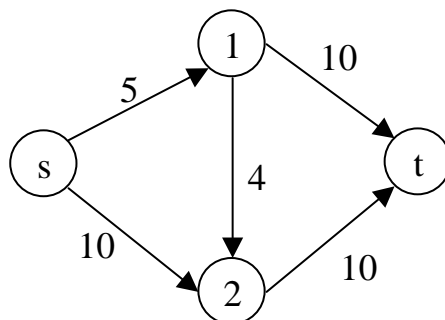


Figura 8.1: Esempio di cammino minimo

Un generico problema di ottimizzazione combinatoria può essere definito come segue. Sia  $B = \{b_1, \dots, b_n\}$  un qualsiasi insieme finito detto *insieme base* ("ground set") e sia  $\mathcal{S} = \{S_1, \dots, S_m\}$  una famiglia di sottoinsiemi di  $B$  ("Subset System"). Infine, sia  $w : \mathcal{S} \rightarrow \mathbb{R}$  una funzione obiettivo che associa a ciascun insieme  $S \in \mathcal{S}$  un numero reale  $w(S)$ . Allora, diremo *Problema di Ottimizzazione Combinatoria* il seguente problema:

$$\min_{S \in \mathcal{S}} w(S)$$

Si osservi la differenza fra la definizione di problema di ottimizzazione combinatoria e la più generale definizione di problema di ottimizzazione introdotta nel Capitolo 1. Nella definizione generale, l'insieme di soluzioni ammissibili  $\mathcal{S}$  può assumere qualunque forma, mentre nei problemi di ottimizzazione  $\mathcal{S}$  è una famiglia (finita) di sottoinsiemi di un insieme finito di elementi.

Ad esempio, il problema del cammino minimo da  $s$  a  $t$  in un grafo orientato  $G = (V, E)$ , con un peso  $p_e$  associato a ogni arco  $e \in E$ , visto nel capitolo precedente, è un problema di ottimizzazione combinatoria. Per questo problema, l'insieme di base è l'insieme degli archi. Per l'istanza mostrata in figura 8.1, l'insieme di base sarà  $B = \{(s, 1), (s, 2), (1, 2), (1, t), (2, t)\}$ . La famiglia di soluzioni è l'insieme di tutti i sottoinsiemi di archi che corrispondono a cammini orientati da  $s$  a  $t$ . In particolare, avremo 3 possibili cammini distinti da  $s$  a  $t$  e quindi  $\mathcal{S} = \{S_1 = \{(s, 1), (1, t)\}, S_2 = \{(s, 1), (1, 2), (2, t)\}, S_3 = \{(s, 2), (2, t)\}\}$ . La funzione obiettivo è semplicemente la somma dei pesi associati agli archi:  $w(S) = \sum_{e \in S} p_e$ . Quindi  $w(S_1) = 15$ ,  $w(S_2) = 19$  e  $w(S_3) = 20$ .

Nella maggioranza dei casi trattati, a ogni elemento dell'insieme di base è associato un peso e la funzione obiettivo è semplicemente la somma dei pesi degli elementi contenuti nella soluzione. Tuttavia la funzione obiettivo può assumere in generale qualunque forma. Un esempio di diversa funzione obiettivo verrà mostrato in seguito.

Nei prossimi paragrafi illustreremo alcuni esempi importanti di problemi di ottimizzazione combinatoria.

### 8.1.1 Il problema dell'accoppiamento massimo su grafi bipartiti

Sia dato un grafo  $G = (V, E)$  non orientato e bipartito, ovvero i)  $V = U \cup W$  e  $W \cap U = \emptyset$  e ogni arco appartenente ad  $E$  ha per estremi un nodo  $u \in U$  e un nodo  $v \in W$  (cioè non esistono archi con ambo gli estremi in  $U$  o ambo gli estremi in  $W$ ). Inoltre, con ogni arco  $(u, v) \in E$  è associato un peso  $w_{uv}$ , che rappresenta il "vantaggio" di assegnare il nodo  $u$  al nodo  $v$ . Un esempio di grafo bipartito (pesato) è mostrato in figura 8.2.a. Ora, a ogni nodo dell'insieme  $W$  vogliamo assegnare al più un nodo dell'insieme  $U$  e viceversa, in modo da massimizzare la somma dei vantaggi. Si osservi che assegnare un nodo  $u \in U$  a un nodo  $v \in W$  è equivalente a selezionare l'arco  $(u, v)$  (di peso  $w_{uv}$ ). Inoltre, se è stato selezionato l'arco  $(u, v)$ , e quindi se è stato assegnato il nodo  $u$  a  $v$  (e viceversa), nessun altro arco incidente in  $u$  può essere selezionato (perchè ciò equivarrebbe ad assegnare  $u$  a più di un nodo); analogamente, nessun altro arco incidente in  $v$  può essere selezionato. Un sottoinsieme  $M \subseteq E$  degli archi di  $G$  che gode della proprietà che, comunque presi due archi appartenenti a  $M$ , essi non hanno estremi in comune, è detto *accoppiamento* (in inglese "matching"). Un esempio di matching (di valore 19) è mostrato in figura 8.2.b.

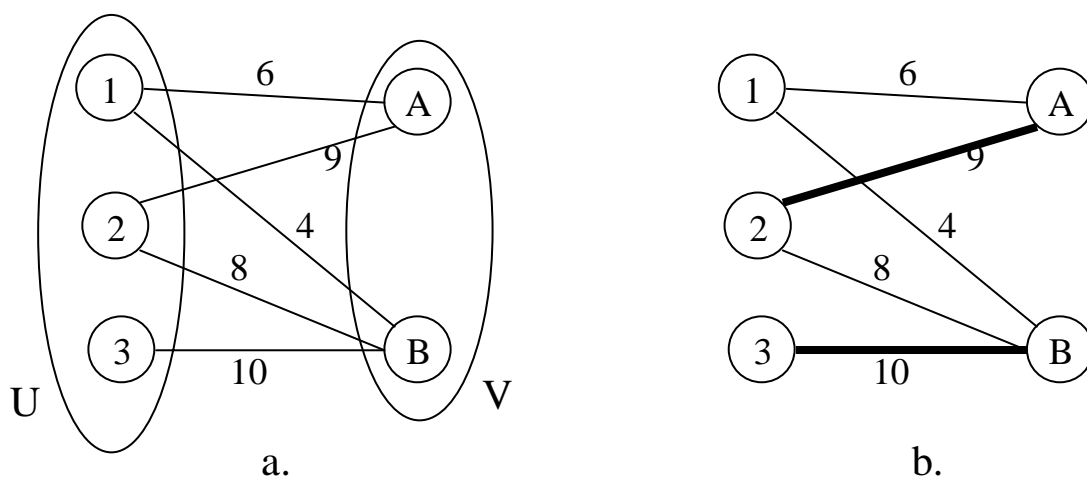


Figura 8.2: Esempio di accoppiamento bipartito

Dunque, il problema dell'accoppiamento massimo può essere così ridefinito: selezionare un sottoinsieme  $M$  degli archi di  $G$  tale che i)  $M$  sia un matching, e ii) la somma dei pesi degli archi in  $M$  sia massima.

Così formulato il problema dell'accoppiamento massimo su grafo bipartito è chiaramente un problema di ottimizzazione combinatoria, ove:

- l'insieme di base coincide con l'insieme degli archi  $E$ .
- la famiglia di sottoinsiemi di archi coincide con la famiglia di tutti i possibili matching di  $G$
- la funzione obiettivo associa a ogni matching  $M$  la somma dei pesi degli archi in  $M$ .

### 8.1.2 Il Problema del Commesso Viaggiatore.

Un commesso viaggiatore deve visitare un certo numero di città, partendo dalla sua città di residenza e ritornandoci alla fine del giro. Questo problema può essere utilmente rappresentato su un grafo  $G = (V, E)$ , ove ogni nodo  $u \in V$  rappresenta una città da visitare, mentre gli archi rappresentano strade (o voli) che collegano coppie di città. Per semplificare la trattazione supporremo che il grafo  $G$  sia non orientato: l'estensione al caso orientato di quanto verrà detto è immediata. A ogni arco  $(u, v) \in E$  si

associa una distanza  $d_{uv} \in \mathbb{R}_+$  che rappresenta la lunghezza del percorso fra la città  $u$  e la città  $v$ . Si osservi che  $d_{uv}$  potrebbe anche rappresentare il tempo di percorrenza, o il costo del trasferimento.

Dunque, il commesso viaggiatore, partendo dalla città in cui abita (sia per esempio la città 1), vuole effettuare tutte le visite e ritornare alla fine alla città di partenza, percorrendo una strada di lunghezza complessiva minima. Si osservi intanto che il "giro" del commesso viaggiatore corrisponde a un ciclo del grafo  $G$  che attraversi ogni nodo una e una sola volta. Un ciclo siffatto è detto *ciclo hamiltoniano*. Quindi, il *Problema del Commesso Viaggiatore* (indicato anche con l'acronimo TSP, dall'inglese *Travelling Salesman Problem*) si definisce come il problema di selezionare un particolare ciclo hamiltoniano, quello (o uno di quelli) di costo (lunghezza, peso) minimo.

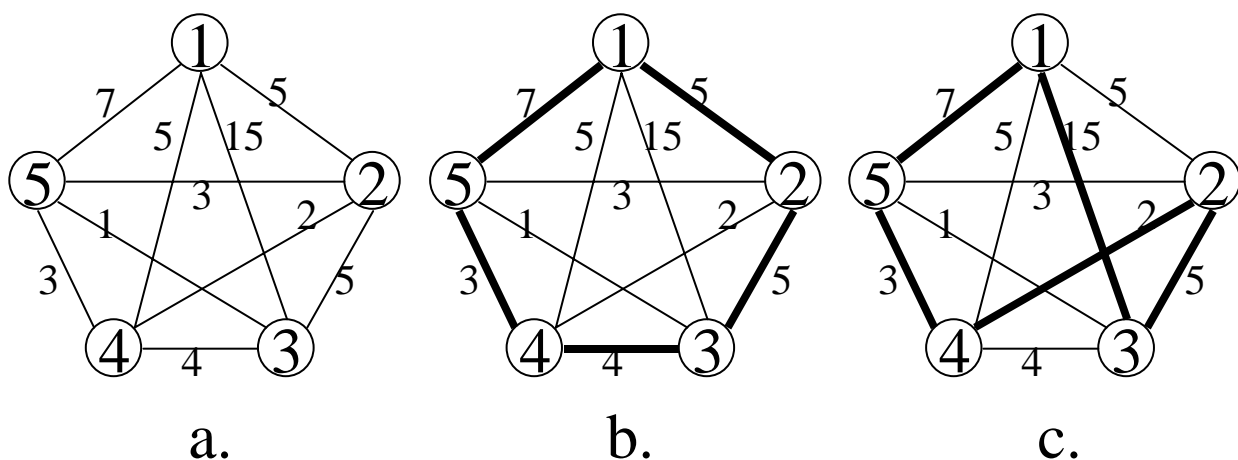


Figura 8.3: Esempi di cicli hamiltoniani

Si consideri ad esempio il grafo in Fig. 8.3. Un possibile ciclo hamiltoniano è, ad esempio, l'insieme di archi  $H_1 = \{(1,2), (2,3), (3,4), (4,5), (5,1)\}$  (Fig. 8.3.b), e il suo costo (somma dei costi degli archi) è pari a 24. Un altro ciclo hamiltoniano è  $H_2 = \{(1,3), (3,2), (2,4), (4,5), (5,1)\}$  (Fig. 8.3.c) di costo pari a 32.

Un ciclo hamiltoniano può dunque essere rappresentato come insieme di archi. Quindi, l'insieme di base del nostro problema di ottimizzazione combinatoria è l'insieme  $E$  degli archi di  $G$ . L'insieme delle soluzioni è una famiglia  $\mathcal{H} = \{H_1, \dots, H_m\}$  di sottoinsiemi di  $E$ , ove  $H_i$  è l'insieme di archi di un ciclo hamiltoniano per  $i = 1, \dots, m$ . La funzione obiettivo è semplicemente, per ogni  $H \in \mathcal{H}$ ,  $w(H) = \sum_{(u,v) \in H} d_{uv}$ .

### 8.1.3 Il problema del partizionamento di un grafo.

Problemi di questo tipo appaiono ogni qual volta si debba scegliere una partizione di un insieme di elementi in un certo numero di classi in modo da minimizzare (o massimizzare) una funzione di distanza fra gli elementi appartenenti a una stessa classe. Sia quindi dato un grafo non orientato  $G = (V, E)$ , e sia associato un peso  $w_{uv}$  con ogni arco  $uv \in E$ . Si vuole trovare la partizione di  $V$  in  $k$  classi (o *cluster*), con  $k$  prefissato, che minimizzi la somma dei pesi degli archi incidenti in nodi appartenenti a una stessa classe. Si ricordi che se  $\pi = \{C_1, C_2, \dots, C_k\}$  è una  $k$  partizione di  $V$  si ha  $\cup_{i=1}^k C_i = V$  e  $C_i \cap C_j = \emptyset$  per  $i, j \in \{1, \dots, k\}, i \neq j$ . Il costo  $w(C_i)$  di una classe di nodi  $C$  è definito come  $\sum_{u \in C, v \in C} w_{uv}$ . Il costo di una partizione  $w(\pi) = w(\{C_1, C_2, \dots, C_k\})$  è definito come  $\sum_{i=1}^k w(C_i)$ . Il problema di Partizionamento di un Grafo consiste nel scegliere la  $k$ -partizione  $\pi^*$  che minimizza  $w(\pi^*)$ .

Anche il problema di partizionamento di un grafo può essere inserito nel *framework* dei problemi di ottimizzazione, anche se l'associazione non è così immediata. Esistono in realtà (almeno) due diverse rappresentazioni del problema come problema di ottimizzazione combinatoria. La prima consiste nel definire l'insieme di base come l'insieme degli archi  $E$  e le soluzioni ammissibili come quei sottoinsiemi di archi che sono archi completamente contenuti in qualche classe di una  $k$ -partizione di  $G$ . Cioè,  $S \subseteq E$  è una soluzione ammissibile se e solo se esiste una  $k$ -partizione  $\pi = \{C_1, \dots, C_k\}$  tale che, per ogni arco  $uv \in S$ , il nodo  $u$  e il nodo  $v$  sono entrambi contenuti in una stessa classe della partizione  $\pi$ . In questo caso, la funzione di costo  $w(S)$  è semplicemente la somma degli archi contenuti in  $S$ .

Una seconda rappresentazione è in qualche senso più naturale, perchè discende direttamente dalla definizione di  $k$ -partizione. Infatti, una  $k$ -partizione può essere definita come una funzione che associa a ogni nodo  $v$  un intero appartenente all'intervallo  $[1, \dots, k]$ , e cioè (l'indice del) la classe di appartenenza. Quindi, una soluzione è completamente definita da un insieme di  $n$  coppie  $(v_1, i_1), (v_2, i_2), \dots, (v_n, i_n)$ , ove con  $i_j$  si è indicato la classe di appartenenza del nodo  $v_j$ . Quindi, l'insieme di base è l'insieme  $B = \{(v, i) : v \in V, i = 1 \dots, k\}$ , mentre una soluzione  $S \subseteq B$  è un insieme di coppie nodo/classe  $(v, i)$  con la proprietà che ogni nodo di  $V$  è contenuto in esattamente una coppia di  $S$ . La funzione obiettivo, in questo caso, non è una funzione lineare degli elementi di  $S$ : in effetti, a ciascun elemento dell'insieme di base non è associato alcun peso. Tuttavia, il costo della soluzione  $S$  dipende dalle coppie appartenenti ad  $S$  nel senso che queste ultime determinano gli archi che appartengono a una medesima classe della partizione e quindi il costo di  $S$ . La funzione di costo sarà quindi  $w(S) = \sum_{i=1}^k \sum_{(u,i) \in S, (v,i) \in S} w_{uv}$ . Questo è un esempio di funzione obiettivo "atipica" anticipato nella parte iniziale di questo capitolo.

Una importante famiglia di problemi di partizionamento pone una restrizione sulla dimensione delle classi. In particolare, con ogni classe viene associata una cardinalità massima, e il problema diventa: trovare la  $k$ -partizione di costo minimo, col vincolo che la cardinalità della  $j$ -esima classe non ecceda la cardinalità massima  $r_j$ , per  $j = 1, \dots, k$ .

#### 8.1.4 PL01

Le soluzioni ammissibili di un problema di programmazione lineare (0,1) sono vettori booleani  $x \in \{0, 1\}^n$ . In genere, ogni componente di un vettore booleano rappresenta il fatto che un certo elemento di un insieme di base  $B = \{b_1, \dots, b_n\}$  è scelto oppure no. Tipicamente, se la componente  $i$ -esima  $x_i = 1$  allora l'elemento  $b_i$  è scelto, se  $x_i = 0$  allora  $b_i$  non è scelto. Ad esempio, nel problema di scelta dei progetti più remunerativi, l'insieme di base è l'insieme dei possibili progetti attivabili e  $x_i = 1$  significa che il progetto  $i$ -esimo è attivato,  $x_i = 0$  significa che il progetto  $i$ -esimo non è attivato. In altri termini, un vettore booleano identifica tutti gli elementi dell'insieme di base che sono stati scelti, e cioè un particolare sottoinsieme  $S \subseteq B$ . Al contrario, dato un qualunque sottoinsieme  $S \subseteq B$ , possiamo associare a  $S$  un vettore booleano  $x^S$  detto *vettore d'incidenza di  $S$* , tale che la componente  $i$ -esima  $x_i^S = 1$  se e solo se  $b_i \in S$ .

In definitiva, un problema di programmazione lineare (0,1)  $\min\{w^T x : Ax \geq b, x \in \{0, 1\}^n\}$  può sempre essere interpretato nel seguente modo:

- Le variabili  $x_1, \dots, x_n$  sono in corrispondenza agli elementi di un insieme di base  $B = \{b_1, b_2, \dots, b_n\}$ .
- I vincoli del problema definiscono la regione ammissibile e quindi ci dicono quali vettori booleani sono ammissibili e quali no. Poichè ogni vettore booleano è in corrispondenza con un sottoinsieme dell'insieme di base, i vincoli definiscono quali sottoinsiemi dell'insieme di base sono soluzioni ammissibili del nostro problema e quali no.
- La funzione obiettivo (lineare) associa un costo  $w_i$  a ogni variabile  $x_i$  per  $i = 1, \dots, n$  e quindi a ciascun elemento  $b_i$  dell'insieme di base  $B$ . E' facile vedere che il costo di una soluzione  $x^S$ , e cioè di un sottoinsieme  $S$  di  $B$ , è semplicemente la somma dei costi degli elementi appartenenti a  $S$ .

## 8.2 Un sistema multitaxi per il servizio Aeroporto Fiumicino - Roma Centro

Descriveremo di seguito un'applicazione reale e recente che ha richiesto l'utilizzazione di diverse tecniche dell'ottimizzazione combinatoria (algoritmi greedy e ricerca locale) e dell'ottimizzazione su reti (cammini minimi e assegnamento). In effetti, gli algoritmi di soluzione descritti nei prossimi paragrafi sono stati scelti come "esempi" di algoritmi proprio perché necessari alla soluzione dell'applicazione descritta. Altre applicazioni avrebbero ovviamente richiesto algoritmi diversi. Si osservi inoltre che la maggior parte dei problemi di ottimizzazione presenti in "natura" non sono puri e la scelta dei modelli da utilizzare non è univoca dipendendo dalla dimensione delle istanze d'interesse, dalla disponibilità di algoritmi efficienti, etc.

Nel 1999 l'Ente Nazionale per le Energie Alternative (ENEA) è stato incaricato dal ministero dell'ambiente di sviluppare sistemi alternativi per il trasporto urbano. Investigando una serie di strategie per la politica dei trasporti, l'ENEA si è resa conto dell'esistenza di alcuni problemi di ottimizzazione connessi alle proposte allo studio. Si tratta di un buon esempio di "sinergia" fra diverse competenze, e soprattutto si tratta di un buon esempio di utilizzazione delle metodologie dell'ottimizzazione intese non come solutori universali, ma come "mattoncini" (in inglese "building block") per sviluppare sistemi complessi.

Fra le varie strategie individuate dall'ENEA, una consisteva nello sviluppo di un sistema integrato di *taxi collettivo* (detto anche *multitaxi*) per il trasporto di persone dall'aeroporto di Fiumicino alle loro destinazioni cittadine e viceversa.

Il taxi collettivo è un'interessante alternativa ai tradizionali sistemi di trasporto pubblico. La principale caratteristica di questo tipo di servizio risiede nel fatto che i percorsi dei veicoli non sono prefissati (come nel caso delle linee degli autobus), ma sono calcolati di volta in volta in base alla domanda; questa flessibilità è propria del servizio taxi "classico", ma a differenza di quest'ultimo, il servizio multitaxi non è individuale. Un sistema di trasporto multitaxi è composto da:

- Una flotta di vetture, non necessariamente uguali tra loro, il cui compito è prelevare dei clienti dai loro punti di partenza e trasportarli ciascuno nelle rispettive destinazioni.
- Uno o più depositi in cui gli automezzi ritornano alla fine del turno di servizio.
- Una centrale di gestione che raccoglie le richieste di servizio degli utenti e comunica alle vetture i percorsi da seguire e i clienti da servire.

Il servizio può essere di tipo immediato o a prenotazione. Nel primo caso le richieste dei clienti sono soddisfatte appena possibile, a partire dal momento della loro formulazione, oppure sono subito rifiutate se non è possibile (o conveniente) soddisfarle. Nel secondo caso le domande devono pervenire alla centrale di gestione con un certo anticipo per essere servite in un secondo momento.

Le richieste degli utenti sono espresse in termini di località di prelievo, località di destinazione e, nel caso di servizio a prenotazione, orario desiderato (di partenza o di arrivo).

Il servizio, inoltre, può essere dei seguenti tipi:

- Molti a molti. In questo modo si indica l'esistenza di molteplici (e distinti) punti di raccolta e di consegna. Questo implica che le vetture, durante i loro viaggi, effettuano sia prelievi che consegne di clienti.
- Uno a molti. In questo caso, invece, si indica l'esistenza di un unico punto di raccolta e di molteplici (e distinti) punti di consegna; di conseguenza le vetture effettuano un solo prelievo di clienti, all'inizio del viaggio, dopodiché si limitano a portare a destinazione tutti i passeggeri a bordo.
- Molti a uno. Le vetture effettuano una serie di prelievi e concludono il loro viaggio nell'unico punto di discesa per i clienti.

Un sistema di trasporto multitaxi è particolarmente adatto in situazioni di utenza debole (anziani e disabili) e nel caso di domanda scarsa (trasporto urbano notturno, servizio in aree poco abitate).

Nel caso specifico del servizio multitaxi dall'aeroporto di Fiumicino a Roma centro, si tratta di un servizio immediato di tipo uno a molti. Coloro che intendono avvalersi di questo servizio si presentano ad uno sportello di accettazione dove vengono registrati i loro dati (nominativo, ora di arrivo allo sportello, indirizzo di destinazione,...). Terminata la fase di registrazione, i clienti restano in attesa fino al momento in cui viene loro comunicato di salire su una vettura per partire.

Per semplicità supporremo che tutte le autovetture abbiano la stessa capacità (ad esempio, possono trasportare al massimo 5 passeggeri). Il sistema automatico, ad intervalli regolari (ad esempio ogni minuto), suddivide i clienti in attesa in gruppi composti al massimo da cinque persone, assegna ciascun gruppo a un taxi specifico, decide quali gruppi debbano partire e, infine, calcola per ogni taxi l'itinerario da seguire, ovvero la sequenza delle consegne. I clienti che compongono i gruppi selezionati per la partenza vengono fatti salire sulle vetture e queste ultime iniziano il loro viaggio, seguendo le indicazioni di percorso fornite dalla centrale di gestione. I dati dei clienti partiti vengono quindi rimossi dal sistema. Infine, le vetture devono tornare all'aeroporto per accogliere nuovi passeggeri. Bisogna quindi decidere a) come si compongono i gruppi (cioè gli equipaggi dei singoli taxi) e b) quale sia, per ogni vettura, il percorso migliore da seguire. Benchè gli obiettivi possono essere molteplici, in prima analisi potremmo assumere che il gestore debba soddisfare le richieste dei clienti cercando di minimizzare il tempo medio (o le distanze) percorse dai suoi taxi. Si tratta evidentemente di un problema di ottimizzazione. Esiste una naturale decomposizione del problema in due fasi: 1. fase di clustering, o di partizionamento, in cui i clienti attualmente in attesa vengono suddivisi in un numero di gruppi di dimensione al più pari alla capacità dei veicoli. Tipicamente si preferisce aggregare clienti con destinazioni il più possibile vicine fra loro. 2. fase di istradamento (in inglese "routing"). In questa fase si decidono i percorsi che i taxi devono seguire per scaricare i passeggeri. I percorsi devono essere tali da minimizzare, ad esempio, le distanze percorse o i tempi di viaggio.

Per poter modellare quanto sopra descritto in termini di problema di ottimizzazione è conveniente definire una struttura matematica adeguata per rappresentare strade, incroci e destinazioni.

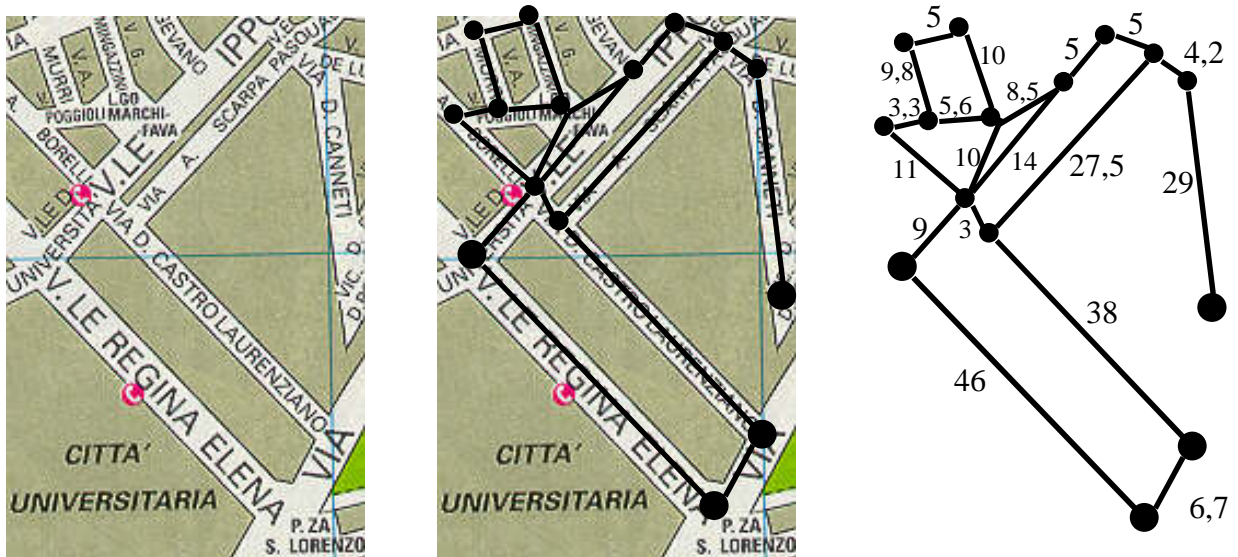


Figura 8.4: Dettaglio del grafo di Roma

**Il grafo di Roma** Tutte le possibili destinazioni di clienti sono rappresentate dal grafo di Roma. Si tratta di un grafo orientato  $R = (W, A)$  dove l'insieme dei nodi  $W$  rappresenta gli incroci (o piazze) della città mentre gli archi sono i tratti di strada compresi fra coppie di incroci (vedi Fig. 8.4). La lunghezza  $c_{uv}$  di ogni arco  $(u, v) \in A$  rappresenta la lunghezza effettiva del tratto di strada corrispondente.

Per semplificare la trattazione successiva, supporremo sempre che la destinazione dei clienti coincida con una piazza o un incrocio, cioè con un nodo del grafo di Roma. Qualora questa ipotesi sia troppo restrittiva, in corrispondenza cioè a tratti di strada molto lunghi fra due incroci adiacenti, si può sempre aggiungere un incrocio *fittizio* suddividendo il tratto in due tratti consecutivi separati (vedi figura 8.5).

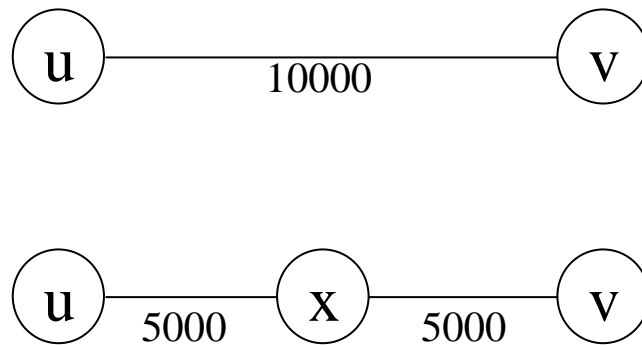


Figura 8.5: Aggiunta di nodi fittizi

Poichè ogni cliente ha una destinazione e ogni destinazione è un nodo del grafo di Roma, i clienti attualmente in attesa sono in corrispondenza con un sottoinsieme di nodi del grafo di Roma. Quindi, il problema di partizionare i clienti in modo che le destinazioni di clienti appartenenti a una stessa classe siano fra loro il più vicino possibile può essere riformulato come un problema di clustering (partizionamento dei nodi) nel sottografo del grafo di Roma indotto dai clienti in attesa.

Una volta che i clienti siano stati assegnati ai veicoli si devono ottimizzare i percorsi dei taxi. È facile vedere che il problema di minimizzare la lunghezza del percorso di un taxi che deve scaricare i suoi passeggeri in alcuni nodi del grafo di Roma e quindi tornare al (nodo di) Fiumicino è esattamente il problema di calcolare un ciclo hamiltoniano di lunghezza minima nel grafo indotto dai nodi di destinazione dei passeggeri e dal nodo di Fiumicino.

Quindi, il problema di gestione di un servizio multitaxi può essere affrontato risolvendo in sequenza due problemi di ottimizzazione su grafi: il problema di clustering (sul grafo indotto da tutti i clienti in attesa) e il problema del commesso viaggiatore (TSP) sul grafo indotto dai nodi destinazione dei clienti di un taxi. Chiaramente, questo secondo problema andrà risolto per ogni vettura in partenza.

Un'altra assunzione che faremo di seguito per semplificare le descrizioni degli algoritmi è l'ipotesi di *grafo completo*.

**Definizione 8.2.1** Un grafo  $R = (W, A)$  si dice completo se, per ogni coppia di nodi distinti  $u, v \in W$ , l'arco  $(u, v)$  appartiene ad  $A$ .

Chiaramente, il grafo di Roma non soddisfa l'ipotesi di completezza, a causa della presenza di moltissime coppie di piazze o incroci (la maggior parte) non collegati direttamente da archi. Tuttavia, per i nostri scopi è possibile completare il grafo, e cioè aggiungere gli archi mancanti, sfruttando le seguenti

considerazioni. Quando un taxi si deve spostare da un generico nodo  $u$  a un nodo  $v$  distinto e non collegato direttamente a  $u$ , dovrebbe preferibilmente scegliere il percorso più breve che da  $u$  porti a  $v$ , e cioè il cammino orientato di lunghezza minima da  $u$  a  $v$ . Nell'ipotesi di taxi "intelligente", quindi, possiamo rimpiazzare l'arco mancante con un arco fittizio di lunghezza pari alla lunghezza del cammino minimo da  $u$  a  $v$ . In altri termini, prima di applicare gli algoritmi di ottimizzazione, il grafo di Roma subirà una fase di *pre-processamento* nella quale:

1. per ogni coppia (orientata) di nodi distinti  $u$  e  $v$  viene calcolato (e memorizzato in opportune strutture dati) il cammino minimo da  $u$  a  $v$ : sia  $c_{uv}^*$  la lunghezza di tale cammino. A tal scopo può essere adoperato l'algoritmo di Dijkstra descritto nel Capitolo ??.
2. per ogni coppia (orientata) di nodi distinti  $u$  e  $v$  si aggiunge l'arco  $(u, v)$  all'insieme di archi  $A$ : la lunghezza associata sarà posta uguale a  $c_{uv}^*$ .

### 8.3 Euristiche di tipo "Greedy"

Una prima classe di euristiche per la soluzione di problemi di ottimizzazione combinatoria va sotto il nome di *euristiche di tipo "greedy"*. Il termine inglese "greedy" può essere tradotto come *avid* ed è stato scelto per rappresentare una caratteristica fondamentale dell'algoritmo greedy, appunto la cosiddetta *scelta greedy*. Per costruire la soluzione finale, che ricordiamo essere un sottoinsieme  $S$  dell'insieme di base  $B$ , l'algoritmo parte in genere da una soluzione parziale che estende aggiungendo un elemento per volta fino a costruire la soluzione finale. L'elemento scelto di volta in volta è quello che minimizza il costo della funzione obiettivo (da cui "avid"). Inoltre, una volta scelto un elemento, questo non viene mai rimosso dalla soluzione (altra caratteristica dell'avidità). Di seguito, denoteremo con  $B = \{b_1, \dots, b_n\}$  l'insieme di base e con  $\mathcal{S} = \{S_1, \dots, S_m\}$  l'insieme delle soluzioni ammissibili del problema di ottimizzazione combinatoria. Inoltre, supponiamo che un costo sia associato a *qualsiasi* sottoinsieme di  $B$  (non solo alle soluzioni ammissibili). E cioè, indicando con  $\mathcal{T}$  la famiglia di tutti i sottoinsiemi di  $B$ , supporremo che sia definita una funzione  $w : \mathcal{T} \rightarrow \mathbb{R}$  per ogni sottoinsieme  $T \in \mathcal{T}$ . L'obiettivo è sempre quello di trovare una soluzione ammissibile  $S \in \mathcal{S} \subseteq \mathcal{T}$  tale che  $w(S)$  sia minimo.

Per introdurre formalmente l'algoritmo greedy, abbiamo bisogno della seguente definizione:

**Definizione 8.3.1** *Un sottoinsieme  $T$  dell'insieme di base  $B$  è detto soluzione parziale (o anche sottosoluzione), se esiste una soluzione  $S \in \mathcal{S}$  tale che  $T \subseteq S$ .*

Si osservi che:

- a. Se  $S$  è una soluzione, allora  $S$  è anche una soluzione parziale. Infatti  $S \subseteq S$ .
- b. Se  $T$  è una soluzione parziale, allora è sempre possibile costruire una soluzione  $S$  tale che  $T \subseteq S$  semplicemente inserendo elementi a  $T$ .

L'algoritmo greedy costruisce una sequenza di soluzioni parziali  $T_0, T_1, \dots, T_q$  con la proprietà che  $T_0$  è l'insieme vuoto,  $T_q$  è una soluzione ammissibile del problema e  $T_i = T_{i-1} \cup \{e\}$ , ove  $e \in B - T_{i-1}$  è l'elemento tale che  $T_{i-1} \cup \{e\}$  sia ancora una soluzione parziale e abbia costo minimo (fra tutte le soluzioni parziali ottenibili aggiungendo un elemento a  $T_{i-1}$ ). Il seguente schema fornisce una versione semplificata dell'algoritmo greedy:

#### Algoritmo Greedy

- a. *Inizializzazione.* Poni  $T_0 = \emptyset$ . Poni  $i = 1$ .
- b. *Iterazione  $i$ -esima.* Scegli  $e \in B - T_{i-1}$  tale che  $T_{i-1} \cup \{e\}$  sia una soluzione parziale e  $w(T_{i-1} \cup \{e\})$  sia minimo.

- c. Poni  $T_i = T_{i-1} \cup \{e\}$ .
- d. *Terminazione* Se  $T_i$  è una soluzione ammissibile: STOP.
- e. Altrimenti Poni  $i = i + 1$ . Va al passo b.

Si osservi che il passo [b.] possiede un certo grado di indeterminatezza. In particolare, potrebbero esistere più elementi che minimizzano il costo della nuova soluzione parziale. In questo caso, è necessaria una regola per "dirimire i pareggi" (in inglese "tie breaking rule"). A volte è sufficiente scegliere a caso (*random choice*); altre volte è più opportuno definire una nuova funzione di costo. Vedremo di seguito esempi delle due alternative.

Per concludere questa sezione, si osservi che ogni qual volta si voglia applicare l'algoritmo greedy a un qualunque problema di ottimizzazione combinatoria, bisognerà innanzitutto definire:

- l'insieme di base
- la famiglia delle soluzioni
- la famiglia delle soluzioni parziali
- la funzione obiettivo

Poichè il nostro esempio prevede la soluzione in cascata di un problema di partizionamento di grafi e di un problema di commesso viaggiatore, considereremo appunto le applicazioni dell'algoritmo greedy a tali problemi. Consideriamo innanzitutto l'applicazione dell'algoritmo greedy al problema del TSP.

### 8.3.1 Applicazione dell'algoritmo greedy generico al problema del Commesso Viaggiatore.

Consideriamo il problema del Commesso Viaggiatore (TSP). Come visto nel paragrafo 8.1.2, una soluzione ammissibile per il TSP è un insieme di archi che definisce un ciclo hamiltoniano. Si consideri ad esempio il grafo in Fig. 8.6. L'insieme delle soluzioni è rappresentato da tutti i possibili cicli hamiltoniani del grafo. Un possibile ciclo hamiltoniano è, ad esempio  $S_1 = \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 1)\}$ , e il suo costo (somma dei costi degli archi) è pari a 24. Un altro ciclo hamiltoniano è  $S_2 = \{(1, 3), (3, 2), (2, 4), (4, 5), (5, 1)\}$  di costo pari a 32. E' facile vedere che i cicli hamiltoniani in un grafo completo sono in corrispondenza ai possibili ordinamenti circolari dei nodi del grafo. Ad esempio, la soluzione  $S_1$  è in corrispondenza dell'ordinamento  $o_1 = (1, 2, 3, 4, 5)$ , mentre  $S_2$  è in corrispondenza a  $o_2 = (1, 3, 2, 4, 5)$  Quindi, se i nodi sono 5, abbiamo  $4! = 24$  cicli hamiltoniani distinti.

**Osservazione 8.3.2** *Il numero di cicli hamiltoniani in un grafo completo con  $n$  nodi è pari a  $(n - 1)!$ .*

La precedente osservazione permette di rilevare che il numero di cicli hamiltoniani di un grafo completo cresce *esponenzialmente* col numero di nodi. In altri termini, quando il numero di nodi è elevato, l'esplorazione completa dell'insieme delle soluzioni diventa impraticabile ed è cruciale sviluppare delle buone euristiche per la soluzione del problema.

Vediamo come l'algoritmo greedy si applica al TSP. Innanzitutto dobbiamo rispondere alla domanda: chi sono le soluzioni parziali del problema del Commesso Viaggiatore? Per vedere ciò consideriamo innanzitutto le proprietà di cui deve godere un insieme di archi  $s$  perchè esso sia un ciclo hamiltoniano.

**Osservazione 8.3.3** *Dato un grafo  $G = (V, A)$  e un insieme  $S \in A$ ,  $S$  è l'insieme di archi di un ciclo hamiltoniano se e solo se:*

1. in ogni nodo di  $G$  incidono esattamente due archi di  $S$
2.  $S$  non contiene cicli di cardinalità inferiore a  $|V|$ .

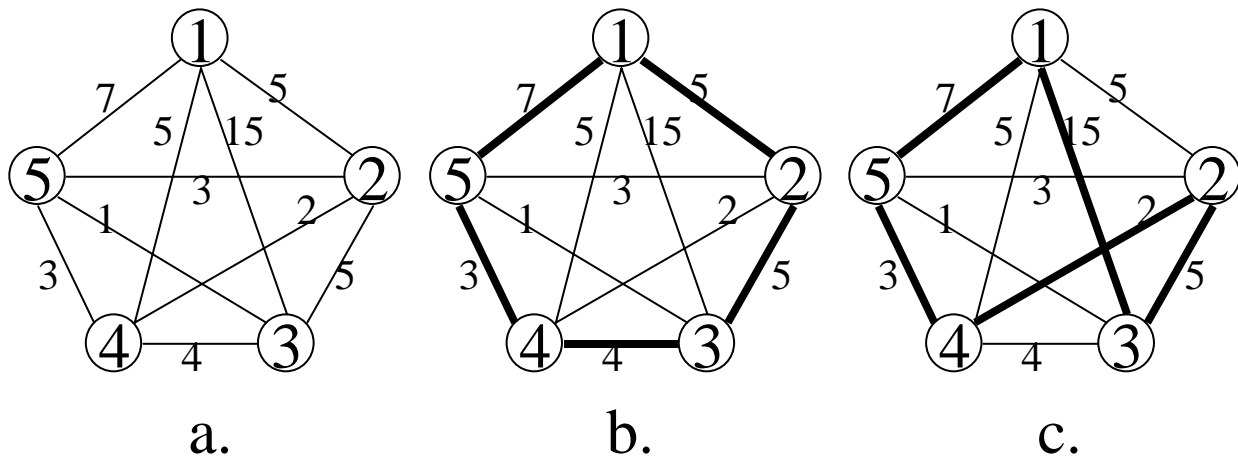


Figura 8.6: Esempi di cicli hamiltoniani

Per capire la natura della condizione 2., si osservi la figura 8.7. In grassetto è evidenziato un insieme  $S$  di archi che soddisfa la condizione 1. ma non la condizione 2. (l'insieme contiene infatti cicli di cardinalità inferiore a 6). In effetti, l'insieme  $S$  non rappresenta in questo caso un ciclo hamiltoniano.

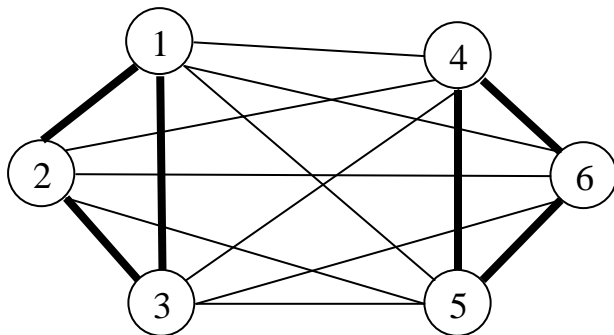


Figura 8.7: Esempi di cicli hamiltoniani

E' facile a questo punto vedere che  $T$  è una soluzione parziale per il TSP se e solo se

1. in ogni nodo di  $G$  incidono al più due archi di  $T$ .
2.  $T$  non contiene cicli di cardinalità inferiore a  $|V|$ .

Un altro elemento da determinare è la funzione di costo. Si è detto che il costo di un ciclo hamiltoniano è pari alla somma dei costi (o lunghezze) degli archi che lo compongono. L'immediata estensione è la seguente: il costo di un sottoinsieme di archi  $T$  è pari alla somma dei costi degli archi nel sottoinsieme, e cioè  $w(T) = \sum_{e \in T} w_e$ .

Siamo ora in grado di applicare l'algoritmo greedy al grafo di Fig. 8.6.

*Inizializzazione.*

- a.  $T_0 = \emptyset$  ( $w(T_0) = 0$ ).  $i = 1$ .

*Iterazione 1.*

- b. Poichè ogni arco del grafo può essere aggiunto a  $T_0$  soddisfacendo le condizioni di soluzione parziale, scegliamo l'arco che costa di meno, cioè l'arco  $(3, 5)$  di costo  $w_{3,5} = 1$ .
- c. Poni  $T_1 = T_0 \cup \{(3, 5)\}$  ( $w(T_1) = 1$ ).
- d. Poichè  $T_1$  non è un ciclo hamiltoniano non possiamo fermarci.
- e. Poni  $i = 2$ .

*Iterazione 2.*

- b. Anche in questo caso ogni arco (in  $A - \{(3, 5)\}$ ) può essere aggiunto a  $T_1$  soddisfacendo le condizioni di soluzione parziale; scegliamo l'arco che costa di meno, cioè l'arco  $(2, 4)$  di costo  $w_{2,4} = 2$ .
- c. Poni  $T_2 = T_1 \cup \{(2, 4)\} = \{(3, 5), (2, 4)\}$  ( $w(T_2) = 3$ ).
- d.  $T_2$  non è un ciclo hamiltoniano.
- e. Poni  $i = 3$ .

*Iterazione 3.*

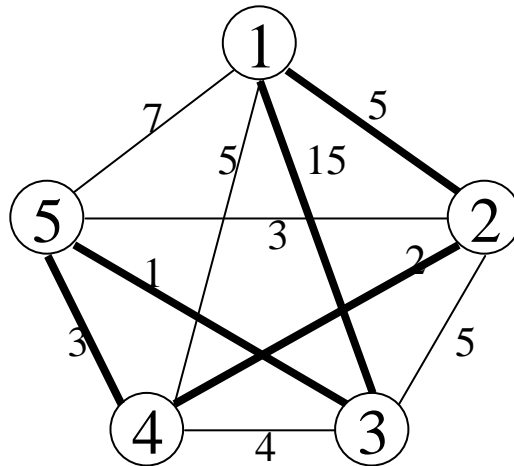
- b. Tutti gli archi residui possono essere aggiunti e quindi scegliamo l'arco meno costoso in  $A - T_2$ : l'arco  $(2, 5)$  di costo  $w_{2,5} = 3$ . Si osservi che anche l'arco  $(4, 5)$  ha costo  $w_{4,5} = 3$ . La scelta fra due archi equivalenti può essere fatta casualmente. Tuttavia, in altri casi si può decidere un criterio di *risoluzione dei pareggi* ("tie breaking rule"), legato ovviamente al tipo di problema.
- c. Poni  $T_3 = T_2 \cup \{(2, 5)\} = \{(3, 5), (2, 4), (2, 5)\}$  ( $w(T_3) = 6$ ).
- d.  $T_3$  non è un ciclo hamiltoniano.
- e. Poni  $i = 4$ .

*Iterazione 4.*

- b. Non tutti gli archi in  $A - T_3$  possono essere aggiunti a  $T_3$  soddisfacendo le condizioni di soluzione parziale; infatti, se aggiungiamo l'arco  $(3, 4)$  si crea il ciclo su quattro nodi  $(2, 3), (3, 4), (4, 5), (5, 2)$ , violando così la condizione (2) di soluzione parziale. Se aggiungiamo l'arco  $(1, 2)$ , invece, avremo tre archi incidenti nel nodo 2. Ragionamenti analoghi valgono per gli archi  $(1, 5), (2, 3), (4, 5)$ . Quindi, gli unici archi che aggiunti non violano le condizioni per le soluzioni parziali, sono l'arco  $(1, 4)$  e l'arco  $(1, 3)$ . Fra i due, scegliamo  $(1, 4)$  che ha costo minimo.
- c. Poni  $T_4 = T_3 \cup \{(1, 4)\} = \{(3, 5), (2, 4), (2, 5), (1, 4)\}$  ( $w(T_4) = 11$ ).
- d.  $T_4$  non è un ciclo hamiltoniano.
- e. Poni  $i = 5$ .

*Iterazione 5.*

- b. L'unico arco selezionabile è l'arco  $(1, 3)$ .
- c. Poni  $T_5 = T_4 \cup \{(1, 3)\} = \{(3, 5), (2, 4), (2, 5), (1, 4), (1, 3)\}$  ( $w(T_5) = 26$ ).
- d.  $T_5$  è un ciclo hamiltoniano: STOP.



### 8.3.2 Applicazione dell'algoritmo greedy generico al problema del Partizionamento di Grafi (clustering).

Si è visto come il problema di clustering consiste nel partizionare i nodi di un grafo  $G = (V, A)$  in  $k$  classi  $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$  in modo da minimizzare la somma dei costi degli archi appartenenti alla stessa classe. Una soluzione  $S$  (detta  $k$ -partizione) può essere rappresentata come un insieme di coppie  $(v, c)$ , ove  $v \in V$  è un nodo del grafo e  $c \in \{C_1, \dots, C_k\}$  è un possibile cluster. Quindi, l'insieme di base  $B$  è l'insieme di tutte le coppie (nodo, cluster) e cioè  $B = \{(v, c) : v \in V, c \in \mathcal{C}\}$ . Una soluzione  $S \subseteq B$  è quindi un sottoinsieme di  $B$  con la proprietà che, per ogni nodo  $v \in V$  esiste una e una sola coppia  $(u, c) \in S$  tale che  $u = v$ . Una conseguenza di ciò è che  $S$  conterrà esattamente  $|V|$  coppie (cioè  $|S| = |V|$ ). Inoltre, per la nostra applicazione (ogni classe corrisponde a una vettura), le classi hanno una cardinalità massima prefissata. Le soluzioni parziali sono tutti i sottoinsiemi  $T$  di  $B$  tali che, per ogni nodo  $v \in V$ , esisterà al più una coppia  $(u, c) \in T$  tale che  $u = v$ . In altri termini, le soluzioni parziali sono  $k$ -partizioni parziali, cioè  $k$ -partizioni di un sottoinsieme dei nodi di  $G$ . Il costo di una soluzione parziale (più in generale, di un sottoinsieme di  $B$ ) è calcolato ancora una volta come la somma dei costi degli archi appartenenti a una stessa classe.

Supponiamo adesso di avere 5 persone in attesa e due taxi disponibili. Inoltre, supponiamo che ogni taxi possa ospitare al massimo 3 persone. Supponiamo infine che il grafo dei clienti sia quello di Figura 8.6. Vediamo adesso come sia possibile calcolare una 2-partizione (partizione in due classi, bi-partizione) del grafo in Figura 8.6.

Poichè  $V = \{1, \dots, 5\}$  e  $\mathcal{C} = \{C_1, C_2\}$ , l'insieme di base sarà  $B = \{(1, C_1), (1, C_2), (2, C_1), (2, C_2), (3, C_1), (3, C_2), (4, C_1), (4, C_2), (5, C_1), (5, C_2)\}$ . E' importante osservare che in questo caso i pareggi vengono risolti individuando una funzione di costo ad hoc per motivi che verranno discussi di seguito. Infine, poichè scegliere una coppia di  $B$  corrisponde ad assegnare un nodo a un cluster, di seguito adotteremo una terminologia (semplificata) e a fianco della locuzione "scegliere la coppia  $(v, C_j)$ ", diremo anche "assegnare il nodo  $v$  al cluster  $C_j$ , e via di seguito.

*Inizializzazione.*

- a.  $T_0 = \emptyset$  ( $w(T_0) = 0$ ).  $i = 1$ .

(Equivalentemente: crea 2 classi vuote  $C_1, C_2$ ).

*Iterazione 1.*

- b. Poichè ogni coppia può essere aggiunta a  $T_0$  senza incrementare il costo di  $T_0$  (ovvero, ogni nodo può essere assegnato a una qualunque classe a costo nullo), ho bisogno di una regola per decidere quale nodo assegnare a quale classe. Nell'esempio del TSP si era optato per una scelta di tipo

”random”. In questo esempio si preferisce una regola che tenga in conto della particolare struttura del problema. Una considerazione euristica è la seguente: conviene innanzitutto ”sistemare” in qualche classe della partizione quei nodi su cui incidono gli archi più costosi in modo da poter scegliere per essi classi abbastanza (se non del tutto) vuote. Quindi la regola di tie breaking sarà: scegli il nodo per cui la somma degli archi incidenti è massima. Di seguito, indicheremo con  $g(v)$  la somma dei costi degli archi incidenti nel nodo  $v$ . E’ facile verificare che  $g(1) = 32$ ,  $g(2) = 17$ ,  $g(3) = 25$ ,  $g(4) = 14$ ,  $g(5) = 14$ . Scegliamo quindi il nodo 1. Poichè le coppie  $(1, 1)$  e  $(1, 2)$  sono equivalenti dal punto di vista del costo delle partizioni corrispondenti (e cioè è indifferente assegnare il nodo 1 alla classe  $C_1$  o alla classe  $C_2$ , possiamo scegliere in modo casuale. Scegliamo quindi la coppia  $(1, C_1)$  (assegnamo cioè il nodo 1 alla classe  $C_1$ ).

- c. Poni  $T_1 = T_0 \cup \{(1, C_1)\}$  ( $w(T_1) = 0$ ).

*Equivalentemente*  $C_1 = \{1\}$ ,  $C_2 = \emptyset$ .

- d. Poichè  $T_1$  non è una partizione (completa) dei nodi di  $G$ , si continua.

- e. Poni  $i = 2$ .

*Iterazione 2.*

- b. Ogni coppia in  $B$  che non contiene il nodo 1 può essere aggiunta a  $T_1$  soddisfacendo le condizioni di soluzione parziale. Tuttavia, ogni coppia del tipo  $(v, C_1)$ , che corrisponde ad assegnare il nodo  $v$  alla classe  $C_1$ , produce un incremento di costo pari al costo dell’arco  $(1, v)$  (infatti il nodo 1 appartiene alla classe 1); al contrario, ogni nodo può essere assegnato alla classe 2 senza incrementi di costo (infatti la classe  $C_2$  è ancora vuota). Infine, poichè ogni coppia di tipo  $(v, C_2)$  produce un incremento di costo nullo, dobbiamo usare ancora una volta la regola di tie breaking e scegliere quindi il nodo di costo massimo, ovvero il nodo 3 ( $g(3) = 25$ ). Scegliamo quindi la coppia  $(3, 2)$ , assegnando in questo modo il nodo 3 alla classe 2.

- c. Poni  $T_2 = T_1 \cup \{(3, C_2)\} = \{(1, C_1), (3, C_2)\}$  ( $w(T_2) = 0$ ).

$C_1 = \{1\}$ ,  $C_2 = \{3\}$ .

- d.  $T_2$  non è una partizione di  $V$ .

- e. Poni  $i = 3$ .

*Iterazione 3.*

- b. In questo caso, ogni coppia aggiungibile a  $T_2$ , e cioè tutte quelle coppie che non contengono il nodo 1 e il nodo 3, producono un incremento di costo positivo. Ad esempio, se il nodo 2 viene assegnato alla classe 1, l’incremento di costo è pari al costo dell’arco  $(1, 2)$ , ovvero 5. E’ facile vedere che il minimo incremento di costo si ottiene scegliendo la coppia  $(5, C_2)$ , ovvero assegnando il nodo 5 alla classe  $C_2$ . Non ci sono pareggi.

- c. Poni  $T_3 = T_2 \cup \{(5, C_2)\} = \{(1, C_1), (3, C_2), (5, C_2)\}$  ( $w(T_3) = 5$ ).

$C_1 = \{1\}$ ,  $C_2 = \{3, 5\}$ .

- d.  $T_3$  non è una partizione.

- e. Poni  $i = 4$ .

*Iterazione 4.*

- b. Tutte le coppie contenenti i nodi 1, 3 e 5 non possono essere aggiunte. L’incremento di costo associato alla coppia  $(2, C_1)$  è pari a 5, per la coppia  $(2, C_2)$  è pari a  $3 + 5 = 8$ , per la coppia  $(4, C_1)$  è pari a 5, per la coppia  $(4, C_2)$  è pari a  $3 + 4 = 7$ . Fra la coppia  $(2, C_1)$  e la coppia  $(4, C_1)$  si sceglie  $(2, C_1)$  perchè  $g(2) = 17 > g(4) = 14$ .

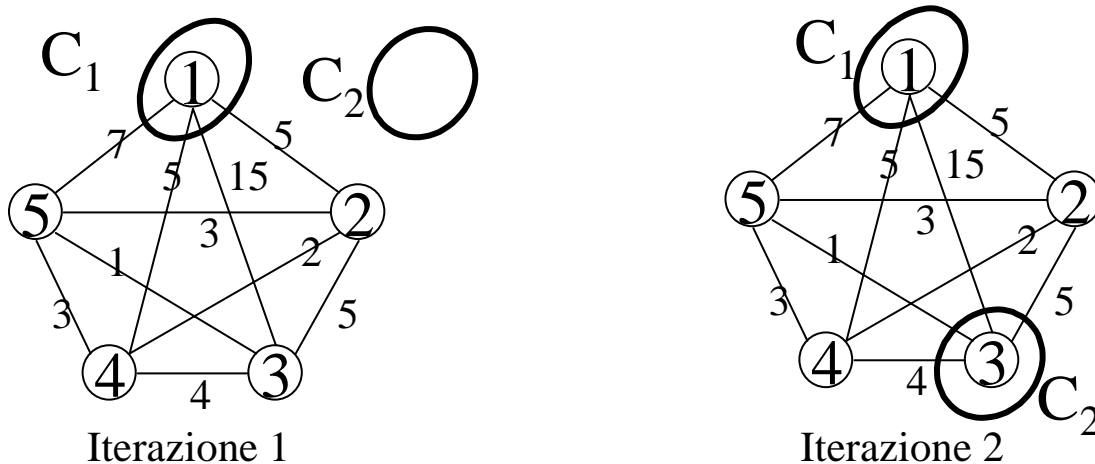


Figura 8.9: Evoluzione delle classi I

c. Poni  $T_4 = T_3 \cup \{(2, C_1)\} = \{(1, C_1), (3, C_2), (5, C_2), (2, C_1)\}$  ( $w(T_3) = 10$ ).

$C_1 = \{1, 2\}$ ,  $C_2 = \{3, 5\}$ .

d.  $T_4$  non è una partizione.

e. Poni  $i = 5$ .

*Iterazione 5.*

b. Le uniche coppie selezionabili sono  $(4, C_1)$  e  $(4, C_2)$ . L'incremento di costo corrispondente a  $(4, C_1)$  è pari a 7 ed è uguale all'incremento prodotto dalla coppia  $(4, C_2)$ . Possiamo scegliere indifferentemente.

c. Poni  $T_4 = T_3 \cup \{(2, C_1)\} = \{(1, C_1), (3, C_2), (5, C_2), (2, C_1), (4, C_2)\}$  ( $w(T_5) = 17$ )

$C_1 = \{1, 2\}$ ,  $C_2 = \{3, 4, 5\}$ .

d.  $T_5$  è una partizione: STOP.

Un'ultima osservazione riguarda la forma con cui vengono generalmente rappresentati gli algoritmi greedy. Infatti, raramente si fa esplicito riferimento all'insieme di base (o alle soluzioni ammissibili come sottoinsiemi dell'insieme di base). Normalmente gli algoritmi vengono presentati in una forma molto più vicina alla descrizione "naturale" del problema. Come visto nell'esempio del problema di clustering, ogni coppia dell'insieme di base rappresenta in realtà l'assegnazione di un nodo a un cluster. Una forma alternativa e più leggibile dell'algoritmo greedy generico applicato al problema di clustering è la seguente.

**Problema.** Sia dato un grafo  $G = (V, A)$ , con costi  $c_a \in \mathbb{R}$  associati a ciascun arco  $a \in A$ . Trovare la partizione dell'insieme dei nodi  $V$  in  $k$  classi  $C_1, \dots, C_k$  che minimizza il costo  $\sum_{i=1}^k \sum_{u \in C_i, v \in C_i} c_{uv}$ .

#### Algoritmo Greedy particolarizzato per il problema di clustering

Note: l'insieme  $W$  utilizzato nell'algoritmo rappresenta l'insieme dei nodi già assegnati (nelle iterazioni precedenti a quella corrente) a qualche cluster.

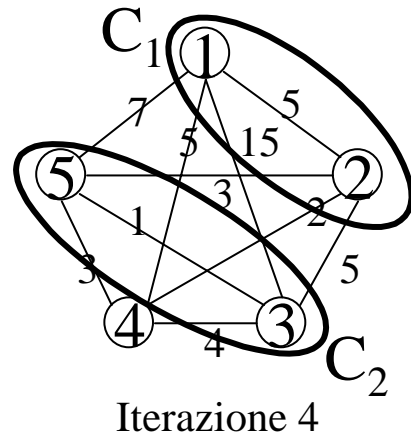
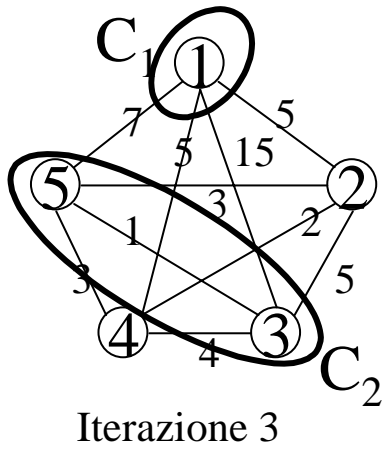


Figura 8.10: Evoluzione delle classi II

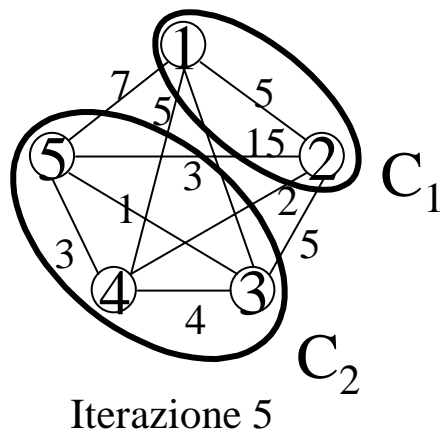


Figura 8.11: Evoluzione delle classi III

- a. *Inizializzazione.* Poni  $C_i = \emptyset$  per  $i = 1, \dots, k$ . Poni  $W = \emptyset$ . Poni  $i = 1$ .
- b. *Iterazione  $i$ -esima.* Scegli un nodo  $u \in V - W$  e assegna  $u$  a un cluster  $C_j$  in modo da minimizzare l'incremento di costo della partizione parziale, e cioè la quantità  $\sum_{v \in C_j} c_{uv}$ . In caso di pareggi, scegli il nodo  $u$  che massimizza  $g(u)$ .
- c. Poni  $C_j = C_j \cup \{u\}$ .
- d. *Terminazione* Se  $W = V$  ogni nodo è stato assegnato a un cluster: STOP.
- e. Altrimenti Poni  $i = i + 1$ . Va al passo b.

E' in questa forma che verranno di seguito descritti gli algoritmi di ricerca locale per l'ottimizzazione combinatoria.

### 8.3.3 Una diversa forma del generico algoritmo greedy.

Gli esempi di applicazione dell'algoritmo greedy sopra elencati hanno una specifica struttura delle soluzioni e delle soluzioni parziali. In particolare, tutte le soluzioni parziali generate dal greedy - tranne l'ultima - non sono soluzioni ammissibili per il problema. Inoltre la funzione obiettivo peggiora a ogni iterazione, ma noi siamo comunque obbligati a continuare i passi finché la sottosoluzione diventa una soluzione del problema.

Per altri problemi di ottimizzazione, come il problema dell'accoppiamento, le soluzioni parziali sono anche soluzioni del problema originario. Infatti, se  $M = \{e_1, \dots, e_q\}$  è un matching, allora ogni sottoinsieme di  $M$  è ancora un matching (e cioè una soluzione ammissibile). In questo caso si tende in generale a estendere le soluzioni parziali perchè tipicamente la funzione obiettivo migliora all'aumentare del numero di elementi contenuti nella soluzione. Per problemi di questo tipo, il generico algoritmo greedy (per problemi di minimizzazione) viene riscritto nel modo seguente:

#### Algoritmo Greedy II

- a. *Inizializzazione.* Poni  $T_0 = \emptyset$ . Poni  $i = 1$ .
- b. *Iterazione  $i$ -esima.* Scegli  $e \in B - T_{i-1}$  tale che  $T_{i-1} \cup \{e\}$  sia una soluzione parziale e  $w(T_{i-1} \cup \{e\})$  sia minimo.
- c. Se per ogni  $e \in B - T_{i-1}$ ,  $T_{i-1} \cup \{e\}$  non è una soluzione ammissibile: STOP.  $T_{i-1}$  è la soluzione greedy.
- d. Altrimenti se  $w(T_{i-1}) < w(T_{i-1} \cup \{e\})$ : STOP.  $T_{i-1}$  è la soluzione greedy.
- e. Altrimenti Poni  $T_i = T_{i-1} \cup \{e\}$ . Poni  $i = i + 1$ . Vai al passo b.

Il passo [c.] serve ad assicurare che l'algoritmo termini qualora ogni elemento residuo non può essere aggiunto alla soluzione parziale corrente senza violare il vincolo di essere soluzione parziale. Il passo [d.] assicura invece che se la prossima soluzione parziale è peggiore di quella attuale essa non venga generata e l'algoritmo termini. Si osservi infine che in questa forma l'algoritmo greedy può essere applicato anche a problemi in cui le sottosoluzioni via via generate non sono soluzioni ammissibili. Si consideri ad esempio il problema del Commesso Viaggiatore. A ogni iterazione generiamo sottosoluzioni che non sono cicli hamiltoniani finché, all'iterazione  $n$ -esima, genereremo il nostro ciclo hamiltoniano. Secondo lo schema qui sopra riportato, all'iterazione  $n + 1$  l'algoritmo terminerà in quanto l'aggiunta di un qualunque arco a un ciclo hamiltoniano non produce una sottosoluzione: quindi sarà verificato il test di terminazione al passo [c.].

E' facile vedere che per adattare il precedente algoritmo a problemi di massimizzazione è sufficiente cambiare il passo [b.] scegliendo l'elemento  $e$  tale che  $w(T_{i-1} \cup \{e\})$  sia massimo e invertendo il senso della disequazione nel test al passo [d.] che va' riscritto come:

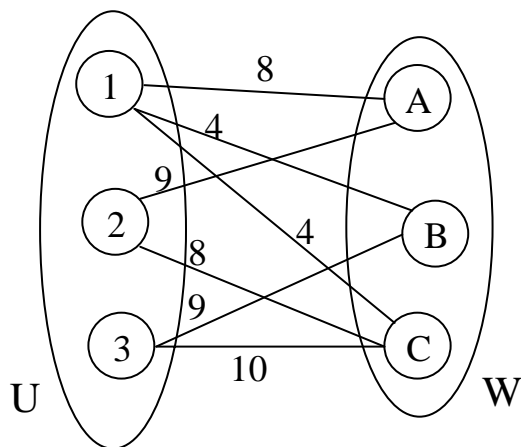


Figura 8.12: Un problema di accoppiamento massimo

[d.] Altrimenti se  $w(T_{i-1}) > w(T_{i-1} \cup \{e\})$ : STOP.  $T_{i-1}$  è la soluzione greedy.

Si consideri l'esempio di Fig. 8.12, ove si voglia calcolare l'accoppiamento di peso massimo.

*Inizializzazione.*

a.  $T_0 = \emptyset$  ( $w(T_0) = 0$ ).  $i = 1$ .

*Iterazione 1.*

b. Poichè ogni arco può essere aggiunto a  $T_0$  mantenendo la proprietà di essere una soluzione parziale, scegliamo l'arco di peso massimo  $(3, C)$ .

c. L'arco esiste, il test di terminazione non è soddisfatto.

d.  $10 = w(T_0 \cup \{(3, C)\}) > w(T_0)$  e il secondo test di terminazione non è soddisfatto (si ricordi che si tratta di un problema di massimizzazione).

e. Poni  $T_1 = T_0 \cup \{(3, C)\}$  ( $w(T_1) = 10$ ). Poni  $i = 2$ .

*Iterazione 2.*

b. Gli archi  $(1, C)$  e  $(2, C)$  non possono essere aggiunti a  $T_1$  in quanto incidenti in  $C$  (esiste già un arco in  $T_1$  incidente in  $C$ ). Analogamente, l'arco  $(3, B)$  non può essere aggiunto perchè incidente in 3. Fra gli archi residui, quello di peso massimo è l'arco  $(2, A)$ , di peso 9.

c. L'arco esiste, il test di terminazione non è soddisfatto.

d.  $18 = w(T_1 \cup \{(2, A)\}) > w(T_1)$  e il secondo test di terminazione non è soddisfatto.

e. Poni  $T_2 = T_1 \cup \{(2, A)\}$  ( $w(T_2) = 19$ ). Poni  $i = 3$ .

*Iterazione 3.*

b. L'unico arco che può essere aggiunto a  $T_2$  è l'arco  $(1, B)$ , di peso 4.

- c. L'arco esiste, il test di terminazione non è soddisfatto.
- d.  $23 = w(T_2 \cup \{(1, B)\}) > w(T_2)$  e il secondo test di terminazione non è soddisfatto.
- e. Poniamo  $T_3 = T_2 \cup \{(1, B)\}$  ( $w(T_3) = 23$ ). Poniamo  $i = 4$ .

*Iterazione 4.*

- b. Nessun arco può essere aggiunto a  $T_3$ .
- c. L'arco non esiste, il test di terminazione è soddisfatto.  $T_3$  è la soluzione greedy.

La soluzione trovata dall'algoritmo greedy per il grafo di Fig. 8.12 è mostrata in Fig. 8.13.

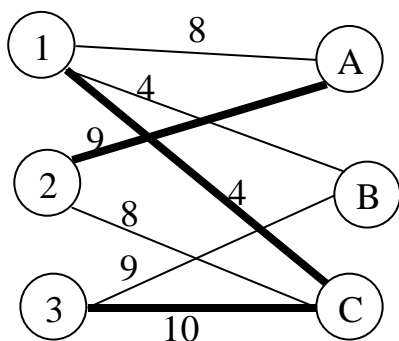


Figura 8.13: Matching massimo

## 8.4 Ricerca Locale

Si è osservato come uno degli elementi caratterizzanti dell'algoritmo greedy sia la natura irrevocabile della scelta greedy. Tuttavia, assai spesso apportando "piccole" modifiche alla soluzione greedy si possono avere miglioramenti nel valore della funzione obiettivo. Si consideri l'esempio del ciclo hamiltoniano di Fig. 8.14.a, che è quello prodotto dall'algoritmo greedy (si confronti con la Fig. 8.6). Il costo del ciclo è pari a 26. Per comodità gli archi non appartenenti al ciclo non sono stati rappresentati nella figura. Se rimuoviamo gli archi (1,3) e (4,5) otteniamo la soluzione parziale di Fig. 8.14.b. Esistono due soli modi di completare questo insieme di archi per renderlo un ciclo hamiltoniano. Il primo è re-inserire gli archi (1,3) e (4,5), riottenendo così il ciclo di partenza. Il secondo è invece scegliere gli archi (3,4) e (1,5), ottenendo così il ciclo hamiltoniano di Fig. 8.14.c.

Si osservi che questo ciclo ha costo 19, cioè un costo inferiore al ciclo di partenza. Quindi, con una piccola modifica della soluzione greedy (cambiando solo due archi) si è ottenuto un sostanziale miglioramento della funzione obiettivo. Proviamo ora, sempre partendo dal grafo di Fig.8.14 a sostituire la coppia (1,2) e (4,5). In questo caso, l'unico modo per ricostruire un ciclo hamiltoniano diverso da quello di partenza è aggiungere gli archi (1,4), (2,5) (si veda la Fig. 8.15).

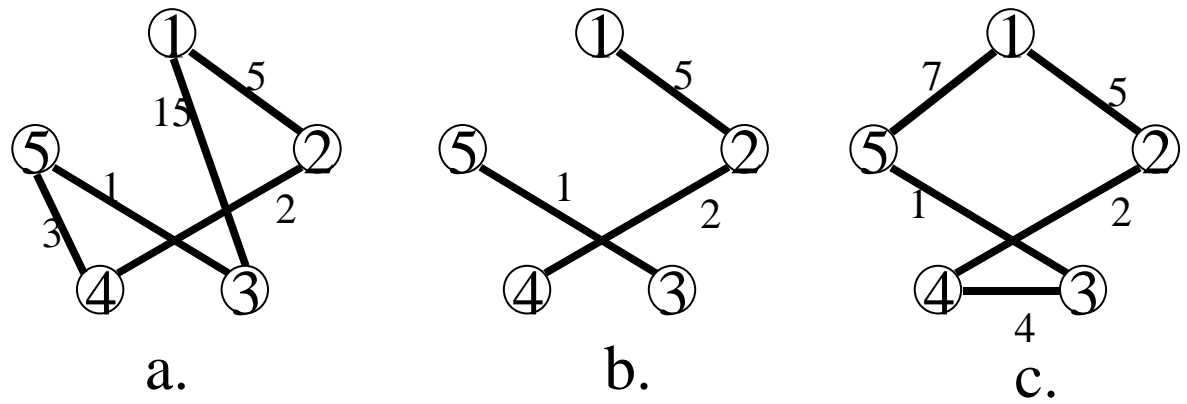


Figura 8.14: Trasformazione di un ciclo hamiltoniano

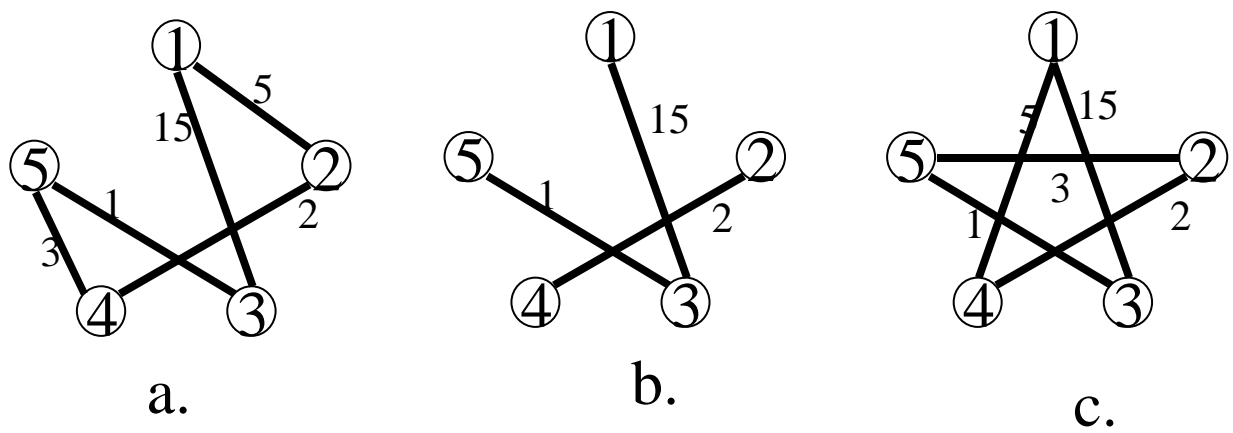


Figura 8.15: Un'altra possibile trasformazione della soluzione greedy

Questo nuovo ciclo tuttavia ha lo stesso costo di quello di partenza. Cerchiamo di generalizzare ciò che è stato fatto. Innanzitutto osserviamo che, se rimuoviamo dal ciclo di partenza due archi adiacenti (ovvero incidenti in uno stesso nodo), esiste un solo modo per completare la soluzione parziale così ottenuta consistente nel re-inserire gli archi appena eliminati. Al contrario, se rimuoviamo due archi non adiacenti (cioè due archi che non hanno nodi in comune), la soluzione parziale può essere completata in due modi distinti, uno dei quali produce un nuovo ciclo. In generale, se indichiamo con  $H_0 = \{(u_1, u_2), (u_2, u_3), \dots, (u_{q-1}, u_q), (u_q, u_1)\}$  un generico ciclo hamiltoniano, allora il meccanismo di generazione di un nuovo ciclo hamiltoniano può essere descritto come segue (si veda la Fig. ??):

1. Scegli una coppia di archi non adiacenti  $(u_i, u_{i+1})$  e  $(u_j, u_{j+1})$ .
2. Rimuovi la coppia di archi dal ciclo
3. Aggiungi i due nuovi archi  $(u_i, u_j)$  e  $(u_{i+1}, u_{j+1})$ .

Poichè l'operazione coinvolge lo scambio di due archi con altri due archi, verrà chiamata *2-scambio* (in inglese "2-exchange").

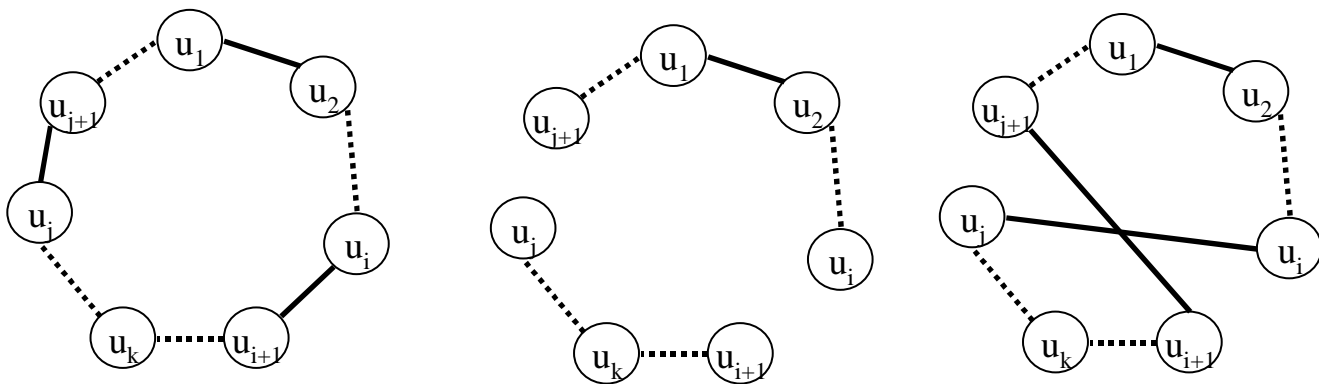


Figura 8.16: Scambio di archi generalizzato

Negli esempi delle figure 8.14 e 8.15 ci siamo limitati a provare la sostituzione di due coppie di archi del ciclo originario  $H_0$ . Un'ovvia generalizzazione consiste nel provare tutti i possibili 2-scambi (in corrispondenza a tutte le possibili coppie di archi non adiacenti di  $H_0$ ) e scegliere alla fine il 2-scambio che produce il ciclo hamiltoniano di costo minimo. Formalmente, possiamo definire la seguente procedura:

1. Per ogni coppia di archi non adiacenti di  $H_0$  calcola la lunghezza del ciclo hamiltoniano corrispondente
2. Scegli il migliore dei cicli così prodotti.

Si osservi che il numero di cicli hamiltoniani esaminati non è molto grande. Infatti, il ciclo iniziale  $H_0$  contiene esattamente  $n$  archi (ove  $n = |V|$ ). Quante sono le coppie di archi non adiacenti? Per ogni arco  $(u, v)$ , ci sono esattamente due archi adiacenti (un arco incidente in  $u$  e un arco incidente in  $v$ ). Quindi, il numero di archi non adiacenti è  $n - 3$ . Quindi, per ognuno degli  $n$  archi del ciclo, esistono esattamente  $n - 3$  2-scambi distinti. Il numero di cicli generati per 2-scambio a partire da  $H_0$  è quindi minore o uguale a  $n(n - 3)$  (esattamente è pari a  $n(n - 3)/2$ ).

Ora, se indichiamo con  $H_1$  il migliore dei cicli hamiltoniani ottenibili per 2-scambio da  $H_0$  saranno possibili due casi:

- a.  $w(H_1) < w(H_0)$ , cioè il ciclo trovato è migliore di quello di partenza.
- b.  $w(H_1) \geq w(H_0)$ , cioè il ciclo trovato non è migliore di quello di partenza.

Supponiamo adesso di trovarci nel caso a., e cioè  $H_1$  è migliore di  $H_0$ . Nulla ci impedisce di ricominciare da capo, nella speranza di migliorare ulteriormente il ciclo hamiltoniano. Quindi possiamo riapplicare ad  $H_1$  la procedura di ricerca del miglior 2-scambio, eventualmente identificando un nuovo ciclo  $H_2$  migliore di  $H_1$  (e quindi di  $H_0$ ); infine, la procedura può essere re-iterata finchè si riesce a identificare un ciclo migliore del precedente.

Siamo in grado di descrivere adesso una delle euristiche più efficienti per la ricerca di un ciclo hamiltoniano di costo (lunghezza) minimo, la cosiddetta euristica 2-*Opt*. Di seguito, denotiamo con  $\mathcal{T}$  la famiglia dei cicli hamiltoniani di  $G$ .

### Algoritmo 2-opt per il TSP

#### *Inizializzazione*

- a. Scegli un ciclo hamiltoniano iniziale  $H_0 \in \mathcal{T}$  (ad esempio mediante l'euristica greedy). Poni  $i = 1$ .

#### *Iterazione i-esima*

- b. Sia  $H_i$  il più corto ciclo hamiltoniano ottenibile da  $H_{i-1}$  per 2-scambio.
- c. Se  $w(H_i) \geq w(H_{i-1})$  allora STOP.  $H_{i-1}$  è il miglior ciclo trovato fino a questo punto.
- d. Altrimenti poni  $i = i + 1$  a va al passo b.

#### *Fine iterazione i-esima*

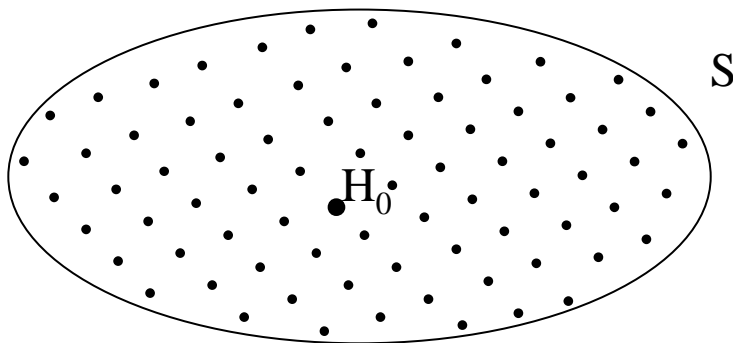


Figura 8.17: Insieme delle soluzioni ammissibili

Quanto abbiamo visto finora applicato al caso del TSP può essere generalizzato a ogni problema di ottimizzazione combinatoria. Esaminiamo più nel dettaglio ciò che è stato fatto. Siamo partiti da un ciclo hamiltoniano iniziale  $H_0$  (*soluzione iniziale*) e abbiamo esaminato un certo numero di cicli hamiltoniani "non troppo diversi" da  $H_0$ . In particolare, abbiamo esaminato tutti quei cicli che hanno esattamente  $n - 2$  archi in comune con  $H_0$ . Chiaramente, questo è solo un piccolo sottoinsieme dell'insieme di tutti i cicli hamiltoniani del grafo (che, come si è detto, ha dimensione  $n!$ ). I cicli hamiltoniani appartenenti a questo sottoinsieme hanno la caratteristica di trovarsi, in qualche senso, "vicino" a  $H_0$ . Graficamente,

possiamo rappresentare tutti i cicli hamiltoniani come dei punti in uno spazio di soluzioni  $S$  (vedi Fig. 8.17).

Il ciclo  $H_0$  sarà un punto appartenente a questo sottoinsieme. L'insieme di tutti i cicli "vicini" ad  $H_0$  è evidenziato in figura 8.18 da un ovale centrato in  $H_0$  e contenuto in  $S$ . Questo insieme è detto in generale *l'intorno* di  $H_0$ , ed è indicato come  $N(H_0)$ . In questo caso stiamo parlando di un particolare intorno, quello definito dal 2-scambio. Nulla ci impedisce di definire intorni più grandi (ad esempio, l'insieme di tutti i cicli hamiltoniani ottenibili sostituendo 3 archi) o più piccoli. In genere, è bene che la dimensione dell'intorno non cresca troppo per poter effettuare efficientemente la visita (e cioè la generazione) dei cicli hamiltoniani in esso contenuti, il calcolo della funzione obiettivo e quindi la scelta della migliore soluzione appartenente all'intorno.

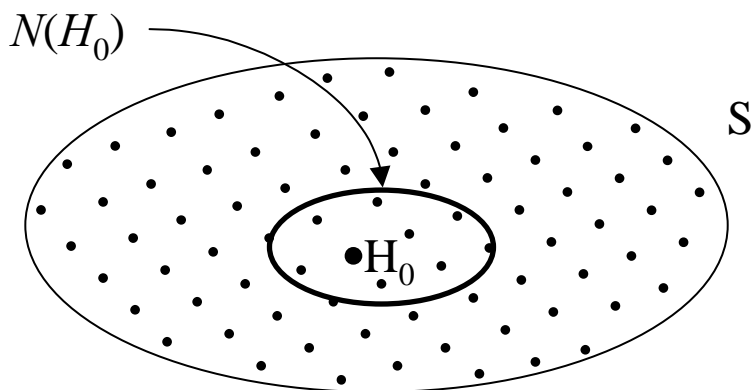


Figura 8.18: Intorno di una soluzione

Continuando con questo esempio, è semplice interpretare l'euristica 2-opt come traiettoria di punti all'interno di questo spazio di soluzioni. Infatti, il metodo può essere interpretato come segue: parto da  $H_0$ , visito il suo intorno, mi sposto nella migliore soluzione trovata  $H_1$ , visito l'intorno di  $H_1$ , mi sposto nella migliore soluzione trovata  $H_2$ , etc. La traiettoria termina quando l'ultima soluzione visitata è migliore (non peggiore) di tutte quelle nel suo intorno.

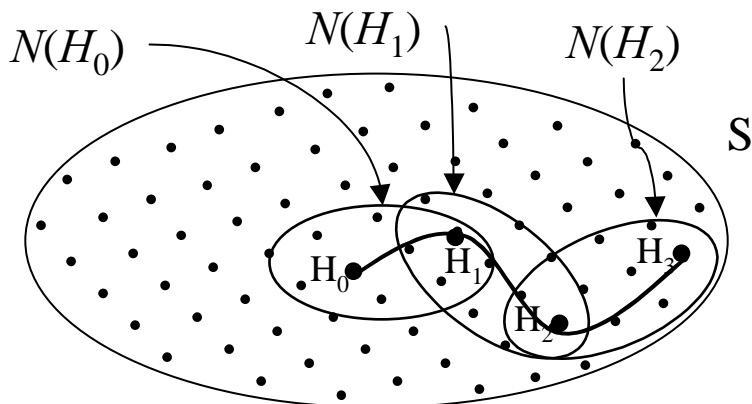


Figura 8.19: Traiettoria nello spazio delle soluzioni

La Figura 8.19 rappresenta appunto sia la traiettoria di soluzioni che la sequenza di intorni visitata.

Si capisce a questo punto il significato del termine "Ricerca Locale". La ricerca (nello spazio delle soluzioni) avviene localmente, negli intorno delle soluzioni sulla traiettoria seguita. L'ultima soluzione è detta *minimo locale*, a sottolineare il fatto che questa soluzione non è l'ottimo globale (su tutto l'insieme delle soluzioni) ma solo l'ottimo locale, relativo alla famiglia di intorno definita.

### 8.4.1 Algoritmo generico di Ricerca Locale

L'euristica 2-Opt per il TSP è un esempio di euristica di *Ricerca Locale*. Il primo passo per costruire un'euristica consiste nel definire un opportuno intorno delle soluzioni. Chiamando con  $S \in \mathcal{S}$  una soluzione ammissibile per il problema di ottimizzazione, si tratta di stabilire come costruire a partire da  $S$  un insieme di soluzioni  $N(S)$ . La definizione dell'intorno avviene tipicamente mediante la specificazione di una particolare operazione, detta *mossa*, da applicare alla soluzione corrente per ottenere le nuove soluzioni. Per esempio, nell'euristica 2-Opt per il Commesso Viaggiatore, la mossa era il cosiddetto 2-scambio (sostituzione di due archi non adiacenti con due nuovi archi). Consideriamo come nuovo esempio il problema di  $k$ -partizionamento. Per semplificare la trattazione, considereremo il caso senza vincoli di cardinalità sulle partizioni. Le soluzioni del problema di  $k$ -clustering sono dunque tutte le  $k$ -partizioni dell'insieme di nodi  $V$  del grafo. Quindi, una soluzione è una specifica  $k$ -partizione, che possiamo indicare come  $S = \{C_1, \dots, C_k\}$ . Un modo per definire l'intorno di una partizione è il seguente: l'intorno  $N(S)$  di una partizione  $S$  è l'insieme di tutte le partizioni ottenibili da  $S$  spostando uno e un solo nodo dalla sua classe di appartenenza a una nuova classe. La mossa in questo caso consiste in:

- scegli un nodo  $v_j \in V$ . Sia  $C_l$  la classe di appartenenza di  $v_j$ .
- genera una nuova partizione  $S' = \{C'_1, C'_2, \dots, C'_l, \dots, C'_q\}$  nel seguente modo:
  - Scegli un indice  $r$  con  $r \in \{1, \dots, q\}$  e  $r \neq l$ .
  - Poni  $C'_i = C_i$  per  $i = 1, \dots, q$  e  $i \neq r, i \neq l$ .
  - Poni  $C'_l = C_l - \{v_j\}$ ,  $C'_r = C_r \cup \{v_j\}$ .

Se consideriamo ancora l'esempio di Fig. 8.11, la soluzione prodotta dall'algoritmo greedy è:  $S_0 = \{C_1 = \{1, 2\}, C_2 = \{3, 4, 5\}\}$ . L'intorno di  $S_0$  è formato da cinque soluzioni  $N(S_0) = \{S_1, S_2, S_3, S_4, S_5\}$ , ove  $S_1 = \{C_1 = \{1, 2, 3\}, C_2 = \{4, 5\}\}$ ,  $S_2 = \{C_1 = \{1, 2, 4\}, C_2 = \{3, 5\}\}$ ,  $S_3 = \{C_1 = \{1, 2, 5\}, C_2 = \{3, 4\}\}$ ,  $S_4 = \{C_1 = \{2\}, C_2 = \{1, 3, 4, 5\}\}$ ,  $S_5 = \{C_1 = \{1\}, C_2 = \{2, 3, 4, 5\}\}$ . È facile anche verificare che  $w(S_0) = 13$ ,  $w(S_1) = 18$ ,  $w(S_2) = 13$ ,  $w(S_3) = 21$ ,  $w(S_4) = 35$ ,  $w(S_5) = 16$ .

Introdotta la definizione di intorno di una soluzione, siamo finalmente in grado di descrivere formalmente il generico algoritmo di ricerca locale.

#### Algoritmo di Ricerca Locale

##### Inizializzazione

- a. Scegli una soluzione iniziale  $S_0 \in \mathcal{S}$  (ad esempio mediante l'euristica greedy). Poni  $i = 1$ .

##### Iterazione $i$ -esima

- b. Sia  $S_i \in N(S_{i-1})$  la migliore soluzione nell'intorno di  $S_{i-1}$ .
- c. Se  $w(S_i) \geq w(S_{i-1})$  allora STOP.  $S_{i-1}$  è il minimo locale.
- d. Altrimenti poni  $i = i + 1$  e va al passo b.

##### Fine iterazione $i$ -esima

Ovviamente è sempre possibile interrompere la ricerca locale prima di aver raggiunto un ottimo locale, ad esempio dopo aver eseguito un prefissato massimo numero di iterazioni.

## 8.5 Estensione del modello al caso Roma Centro - Fiumicino Areoporto

Finora abbiamo trattato il caso del servizio da Fiumicino Areoporto al centro città. In questo contesto le vetture, una volta lasciati i passeggeri alle loro destinazioni, tornano all'areoporto per prelevare nuovi clienti. Tuttavia nulla impedisce loro di trasportare passeggeri da Roma centro all'areoporto di Fiumicino effettuando il servizio anche nel viaggio di ritorno. In effetti, un servizio del genere è allo studio. Il modo più semplice per affrontare il problema consiste nel considerare le due tratte (Fiumicino-Roma Centro e Roma Centro - Fiumicino) come due problemi distinti.

Il caso Roma Centro - Fiumicino presenta qualche elemento di complessità addizionale. Infatti, in questo caso il servizio viene effettuato su prenotazione e i clienti vanno raccolti rispettando il più possibile gli orari stabiliti. In ogni caso, ancora una volta di stratta di un problema di clustering (assegnazione dei clienti alle vetture) e di un problema di Comesso Viaggiatore (scelta del percorso ottimo) risolti in sequenza. Quindi, alla fine del processo di ottimizzazione avremo un certo numero di cicli hamiltoniani  $H = \{H_1, H_2, \dots, H_q\}$  ciascuno rappresentante i clienti da prelevare e la sequenza da seguire per ciascun taxi. A ogni ciclo avremo inoltre associato un tempo d'inizio raccolta  $t_i$  per  $i = 1, \dots, q$ , ovvero il primo cliente di ogni ciclo  $H_i$  deve essere prelevato al tempo  $t_i$ . Si tratta ora di assegnare ciascun ciclo a un taxi proveniente dall'areoporto e che abbia già concluso il suo giro di consegne. Quale vettura conviene assegnare al ciclo  $H_j$  (per  $j = 1, \dots, q$ )? Sicuramente un taxi che abbia finito le consegne in tempo per prelevare il primo cliente al tempo  $t_j$  (ma non troppo prima, altrimenti il taxi dovrà aspettare a lungo). Inoltre, ci converrà assegnare un taxi che consegnerà il suo ultimo passeggero in un nodo del grafo di Roma abbastanza vicino al nodo corrispondente al primo cliente del ciclo di ritorno. Allora, se chiamiamo con  $A = \{A_1, A_2, \dots, A_r\}$  l'insieme dei cicli hamiltoniani che corrispondono a taxi provenienti dall'areoporto di Fiumicino che possono essere assegnati ai cicli di ritorno  $\{H_1, H_2, \dots, H_q\}$ , si tratterà innanzitutto di definire un insieme di pesi  $p_{ij}$  che rappresentino il vantaggio di assegnare il taxi che serve  $A_i$  al percorso di ritorno  $H_j$ . Ad esempio, se  $d_{ij}$  è la distanza fra l'ultimo passeggero di  $A_i$  e il primo passeggero di  $H_j$ , potremmo porre  $p_{ij} = -d_{ij}$ . Potremmo aggiungere nel peso anche un fattore relativo al tempo di attesa - minori tempi di attesa per il conducente sono da preferire). A questo punto vogliamo scegliere come assegnare ogni vettura che arriva dall'areoporto a un ciclo verso l'areoporto. Costruiamo quindi un grafo bipartito  $G = (V, E)$  ove l'insieme dei nodi sarà  $V = H \cup A$ , ed esiste un arco fra il nodo  $H_j \in H$  e il nodo  $A_i \in A$  se e solo se l'ultimo cliente di  $A_i$  viene lasciato in tempo utile per prelevare il primo cliente di  $H_j$ . Inoltre, associamo a ogni arco  $(i, j) \in E$  il peso  $p_{ij}$  che rappresenta il vantaggio di assegnare il taxi del ciclo  $A_j$  al ciclo di ritorno  $H_i$ . È facile convincersi che cercare l'assegnamento migliore corrisponde a risolvere un problema di accoppiamento massimo nel grafo  $G$ .