

Schemi Risorse Operative

Programmazione Lineare \rightarrow Sia funzione obiettivo che vincoli del problema sono lineari (n variabili, m vincoli)



Forme canonica del problema:

$$\min c^T x$$

$$Ax \leq b$$

Problemi di P2 \rightarrow Diversi Tipi:

- Problemi di allocazione ottima di risorse
- Problemi di mixelazione
- Problemi di Transporto

• Problemi di allocazione ottima di risorse

Definiamo un insieme di risorse R \rightarrow Ogni risorsa disponibile in una quantità finita b_i

Dato un insieme di prodotti? \rightarrow A ogni prodotto è associato un profitto c_j

Definiamo la matrice dei Tassi di assorbitamento \rightarrow R_i sulle righe, P_j sulle colonne

Il generico elemento a_{ij} indice le quantità di risorse necessarie per produrre una unità di prodotto j \rightarrow Vogliamo massimizzare il profitto \rightarrow $\max c^T x$

$$\left[\begin{array}{l} c \rightarrow \text{Vettore dei profitti} \\ b \rightarrow \text{Vettore delle quantità.} \end{array} \right]$$

$$\left\{ \begin{array}{l} Ax \leq b \\ x \geq 0 \end{array} \right.$$

• Problemi di miscelezione



Definiamo un insieme di sostanze da miscelare \Rightarrow

→ Di ciascuna sostanza si conosce il costo unitario $c \in \mathbb{C}$



Definiamo un insieme di componenti utili

G

→ Ogni componente G_i deve essere presente nella miscela finita



Dati questi due insiemi è possibile definire una matrice di Componenti-Sostanze

↓
Tante righe quante sono le componenti e Tante colonne quante sono le sostanze

→ Il generico elemento indica la quantità di componente i presente in una unità di sostanza j



min $c^T x$

$$\begin{cases} Ax \leq b \\ x \geq 0 \end{cases}$$

Problemi di Trasporto \rightarrow Un prodotto generato in m Origini di origine deve essere trasportato a n destinazioni di destinazione

Per ogni nodo di origine si ha una disponibilità massima di merce

B_i

Per ogni stazione di destinazione si ha una richiesta D_j

Definiamo una matrice dei costi di trasporto

\rightarrow I coefficienti rappresentano il costo di trasporto di una unità di merce dall'origine i alla destinazione j

Vogliamo minimizzare il costo di trasporto

Originale sulle righe, destinazione sulle colonne

Problemi di Trasporto \rightarrow Rappresentabile come grafo bipartito

$\left\{ \begin{array}{l} m \text{ nodi di origini} \\ n \text{ nodi di destinazione} \end{array} \right. \rightarrow \begin{array}{l} \text{Disponibilità } B_i \\ \text{Domanda } D_j \end{array}$

Formulazione:

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

$$\left\{ \begin{array}{l} \sum_{j=1}^n x_{ij} \leq B_i \quad \forall i \\ \sum_{i=1}^m x_{ij} \geq D_j \quad \forall j \end{array} \right.$$

con $x_{ij} \geq 0$

Nota:

Se $\sum B_i < \sum D_j$ il problema è detto inaccessibile

Disponibilità & Richiesta

Problemi di P2 con funzione obiettivo non lineare



$$\max \min \{l_1(x), l_2(x), \dots, l_q(x)\}$$

→ Si ricorda che le funzioni $e_i(x)$ sono lineari



Tale modello è detto max-min



È sempre possibile trasformare

un modello del genere in uno equivalente lineare

→ Introduciamo la variabile

$$y = \min \{e_1(x), e_2(x), \dots, e_q(x)\}$$



La funzione obiettivo diventa

$$\max \{y\}$$

è dunque necessario che la variabile y , i vincoli: $y \leq l_i(x) \quad \forall i = 1, \dots, q$

Altro problema simile → min-abs



La funzione obiettivo è del tipo

$$\min \{|e(x)|\}$$



Risoluzione basata sulla definizione

$$|e(x)| = \max \{e(x), -e(x)\} \rightarrow \text{Si ottiene un modello max-min}$$



$$y = \max \{e(x), -e(x)\} \rightarrow \text{nuova f.o.}$$

$$y = e(x)$$

$$y = -e(x)$$

} → vincoli da introdurre

Schemi Ricerca Operativa

Rappresentazione grafica di problemi lineari

Particolarmente utile quando il vettore delle variabili di decisione x è bidimensionale

→ Risolvibili nel piano cartesiano mediante l'equazione lineare

$$ax_1 + bx_2 = c$$

Ogni vincolo individua una regione del piano

→ Dati n vincoli si identificano n regioni del piano le cui intersezioni individuano la regione di ammissibilità del problema

Caso bidimensionale

Molti vincoli potrebbero essere ridondanti

Funzione obiettivo → Rotta nel piano xy

È possibile dimostrare che la soluzione ottima di un problema di P.R. si trova sulla frontiera del dominio di ammissibilità

→ La soluzione ottima T_{opt} è ottima se si trova sui vertici della regione di ammissibilità

Nel caso bidimensionale è facile da mostrare e quindi conviene tracciare le curve di livello e puntate

Possibili costruzioni: dalla funzione obiettivo

1) Regione delle soluzioni ammesse → Curva di livello perfettamente sovrapposta alla frontiera → Valutiamo i vertici del poliedro

2) Problema inadmissible → Non è possibile individuare alcun dominio di ammissibilità → Vincoli in disaccordo

È possibile anche imbattersi in un dominio di ammissibilità illimitato



Non è sempre possibile individuare un vertice ottimo

In conclusione: In un problema di LP si possono presentare tre possibili scenari

- 1) Regione di ammissibilità vuota \rightarrow Non si ha soluzione
- 2) Regione di ammissibilità illimitata \rightarrow È possibile che non esista la soluzione ottima
- 3) Regione di ammissibilità limitata \rightarrow Il problema unica soluzione è l'ottima si trova in corrispondenza dei vertici

Schemi Ricerca Operativa

Simplesso \rightarrow Primo algoritmo per la risoluzione
di problemi di LP

Lo è il più usato

Si come un poliedro ha sempre un numero
finito di vertici, allora un dominio di ammissibilità
accoglie un numero finito di soluzioni ottime

Poliedro come dominio di un
problema con n variabili decisionali
identifica nell'iperspazio n vertici

Algoritmo del Simplex \rightarrow Identifica i vertici del poliedro in
maniera intelligente

È un algoritmo a direzione
ammisibile

Valuta la funzione obiettivo nei
punti corrispondenti ai vertici e
ne estrae il migliore \rightarrow Appoggio a forza bruta

Genera una successione
di punti attraverso la riduzione

$$x^{n+1} = x^n + \partial_n d^n$$

Effettua questa valutazione
in maniera intelligente

Determina un vertice iniziale
 v e valuta se sia una soluzione
ottima. Se v non è una
soluzione ottima \rightarrow viene determinato
un nuovo vertice in maniera
intelligente.

Soluzione ottima di un problema di LP
se esiste, si trova in corrispondenza
di un vertice del dominio di ammissibilità

Sono particolari soluzioni
dette basiche

\curvearrowleft
Fino a convergenza

Lo ottiene a partire dalle
trasformazioni dei vincoli
in vincoli di uguaglianza

Simplesso \rightarrow Primo passo: Trasformare il problema originario in una

forma standard

\hookrightarrow Tutti vincoli di uguaglianza

Po^r poter effettuare la Trasformazione
è necessario effettuare l'uso di
variabili auxiliari dette Slack

- Un vincolo di Tipo \leq si trasforma
in un vincolo di tipo $=$ aggiungendo
le variabili slack.
- Un vincolo di tipo \geq si trasforma
in un vincolo di tipo $=$ sottraendo
le variabili slack.

Fronterie \rightarrow Costituita dall'area variabile

slack $y_i = 0$

Un punto interno al dominio
avrà variabili slack diverse
da 0

\rightarrow y_i rappresenta la distanza tra
un punto interno al dominio di
ammissibilità e la sua frontiera

Soluzione basica ammissibile \rightarrow Dato un sistema di m equazioni in n variabili

con $n > m$, una soluzione basica ammissibile è
definita come una soluzione che presenta $n-m$ variabili
nulle e m variabili non negative.

Degenera nel caso in cui presenti:

$n-m$ variabili nulle e m variabili
non negative con $m > m$

Numero di soluzioni basiche ammissibili \rightarrow Pari al numero di combinazioni di m e n

$$\frac{n!}{m!(n-m)!}$$

Simplesso \rightarrow A ogni iterazione si sposta di un vertice in vertice \rightarrow Tra i vertici adiacenti risulta sempre una variazione di una singola variabile basica

Esiste una corrispondenza biunivoca tra i vertici del dominio di ammissibilità e le soluzioni basiche ammissibili del sistema di equazioni del problema in forma standard.

Se il vertice non appartiene agli assi sarà una delle variabili slack a diventare non basica \rightarrow Ad annullarsi.

Simplesso \rightarrow Vincoli di Tipo \leq

Caso più semplice \rightarrow Avremo tutte le variabili slack positive

e troveremo una soluzione basica ammissibile è banale

Trasformazione di tutti i vincoli di tipo \leq in vincoli di tipo $=$ e aggiungiamo le variabili slack

Coincide con l'origine degli assi

Classica risoluzione del simplex

Simplesso \rightarrow Vincoli di tipo \geq e $=$

Più complicato \rightarrow Il problema in forma canonica non contiene la matrice d'identità

Non possiamo considerare l'origine come una soluzione del problema

Le variabili slack sono di segno negativo

Introduciamo altre variabili auxiliarie h

Dette variabili artificiali

Simplesso con vincoli di Tipo \leq e $=$



Il sistema sarà costituito da:

- n variabili originali
- p variabili slack > 0 (per vincoli di Tipo \leq)
- q variabili slack $= 0$ (per vincoli di Tipo \geq)
- r variabili artificiali > 0 (per vincoli di Tipo \geq e $=$)

Risoluzione del Simplex con vincoli di Tipo \leq e $=$



Due possibilità:

1) Big M

2) Dae fusi

Big M → Modifica la funzione obiettivo

Introduce variabili b_i con segno negativo (se la f.o. è a massimizzare)
o con segno positivo (se la f.o. è a minimizzare)

Favorisce l'annullamento delle variabili artificiali e quindi
la loro uscita dalla base

Potrebbero non essere eliminate
Tutte le variabili artificiali nella
base

↳ Variabili b_i moltiplicate per un coefficiente
n diverso

↳ Problema inconsistente

Big M → Difficile da implementare → Problemi di round off e di overflow

Due fasi → Forchiamo sempre di determinare
una prima soluzione basica ammessa
e di rimuovere le variabili artificiali
dalla soluzione del problema

Introduce una nuova funzione obiettivo auxiliare w

$$w = \sum_i b_i$$

Dove sempre essere minimizzata

Se il minimo della funzione è più
di 0 si è raggiunto l'annullamento
di tutte le variabili artificiali

Simplesso → Soluzione basica ammessa degenere

Soluzione degenere quando
almeno una delle variabili
in base assunse valore zero

Soluzione che ha più di n-m variabili
uguali a zero \Rightarrow meno di m variabili positive

Degenerezza quando uno o più termini
noti b_i sono nulli

Soluzioni degeneri possono
portare a ciclazione

L'algoritmo non riesce a
convergere a una soluzione
ottima

Il risultato del pivoting sarà il passaggio
a una nuova soluzione basica in cui
le variabili anticamente entrate in base con
valore zero

Ciclizzazione \rightarrow l'algoritmo "ripone" una soluzione già
generata in precedenza

Necessario stabilire delle regole di
anticiclizzazione

Regole di Bland \rightarrow tra le variabili non in base con coefficiente
di costo negativo (o positivo, dipende dal
problema) si sceglie la variabile con indice
di colonna più basso \rightarrow la picca da un rozzo dx

Stessa logica per la variabile uscente

Schemi Ricerca Operativa

Programmazione Lineare Intera



Consideriamo un insieme di variabili e una funzione obiettivo lineare

Vogliamo calcolare una soluzione → Insieme delle soluzioni è un sottoinsieme di $\mathbb{S}^n_0 \subset \mathbb{Z}^n$



Sottoinsieme di vettori le cui componenti sono valori interi
↳ \mathbb{Z}^n

Programmazione Lineare Intera



Risolvibile con metodo di Branch and Bound

→ Componenti fondamentali di Branch and Bound



Branching and Bounding

Bounding → Determina un valore limite, il Bound, di funzione obiettivo

Utile per il raffrasamento continuo

- Upper Bound per un problema a massimizzazione
- Lower Bound per un problema a minimizzazione

Elimina i viali di integrazione

Branching → Partizione in sotto problemi e genera nuovi sottoproblemi più semplici

↳ Partizione binaria o multivalore per effetto ed eccesso

Raffrasamento continuo → \mathbb{I} insieme di ommissibilità del problema intero e I_r il valore delle sue soluzioni ottime. Sia R l'insieme di ommissibilità del suo raffrasamento e sia I_r il valore delle sue soluzioni ottime. L'insieme \mathbb{I} è certamente contenuto in R

Individuare i bound del problema intero

Branch and Bound



Algoritmo esatto \rightarrow Se esiste, trova l'ottimo

Bound migliori permettono di ridurre le dimensioni dell'albero



Tra più individui sono richieste più risorse



\rightarrow Necessario un trade off tra impiego di risorse (buoni bound) e grandezza dell'albero

Ricerca di Bound



Due strategie:

1) Depth first \rightarrow Si scava in profondità l'albero delle decisioni e poi si risale

2) Best first \rightarrow Si espone l'albero in ampiezza esaminando per primo il nodo con bound migliore



- Problema a massimizzazione \rightarrow UB maggiore

- Problema a minimizzazione \rightarrow LB minore

Schemi Ricette Operativa

Problemi di ottimizzazione definiti su grafo

- Problema del cammino minimo
- Problema del massimo flusso

Cammino minimo

↓
Modellabile come
problema di progettazione
lineare

→ Dato un grafo orientato $G = (V, A)$ con
 V insieme dei vertici e A insieme degli archi
si vuole trovare il percorso di costo minimo
da un nodo sorgente $s \in V$ a un nodo
destinazione $d \in V$

↓
Ciascun arco (i, j) è caratterizzato da
un costo c_{ij}

Shortest Path da uno a uno

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

$$\sum_{j \in \delta^+(i)} x_{ij} - \sum_{j \in \delta^-(i)} x_{ji} = \begin{cases} 1 & \text{se } i = s \\ 0 & \text{altrimenti} \\ -1 & \text{se } i = d \end{cases}$$

$$\text{con } x_{ij} = \begin{cases} 1 & \text{se } (i, j) \in P \\ 0 & \text{altrimenti} \end{cases} \quad P \text{ percorso che viene costituito}$$

$\delta^+(i)$ e $\delta^-(i)$ indicano le stelle degli archi uscenti ed entranti
da i rispettivamente

Shortest Path da uno a Tutti \rightarrow Ogni nodo deve essere Toccatto con il percorso di costo minimo

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

n numero Totale di destinazioni

$$\sum_{j \in S(i)} x_{ij} - \sum_{j \in S(i)} x_{ji} = \begin{cases} n-1 & \text{se } i=s \\ -1 & \text{se } i \in V \setminus \{s\} \end{cases}$$

x_{ij} \rightarrow Indice la quantità di "pacchi" che
può passare per quel determinato arco

Il modello valido solo se il grafo è
acchiaro

Se fosse acchiaro, al fine di avere un
modello corretto e risolvibile, sarebbe
necessario che non ci siano cicli di
costo Totale negativo

\rightarrow Un qualsiasi algoritmo andrebbe
in loop

\downarrow
Continuo miglioramento
della soluzione

Problemi di consumo minimo

\downarrow
Approssimazione di due tipi:

- Label Setting \rightarrow A ogni iterazione rendono un'etichetta permanente

\hookrightarrow Applicabili se il grafo è acchiaro / non presenta archi di costo negativo

- Label Correcting \rightarrow Sfuggono etichette temporanee \rightarrow vengono cancellati solo
all'ultima iterazione

\downarrow
Applicabili anche in presenza di archi
negativi:

\rightarrow Il costo di eventuali cicli
non deve essere negativo

Problemi di massimo flusso



Dato un grafo orientato G , ogni arco è caratterizzato da una capacità massima u_{ij} di flusso che può scorre in quel dato arco

arco



Si vuole determinare la massima quantità di flusso che è possibile inviare dalla sorgente s alla destinazione d attraverso il grafo

$$G = (V, A)$$

max { } ↓

$$\sum_{j \in \delta^+(i)} x_{ij} - \sum_{j \in \delta(i)} x_{ji} = \begin{cases} f & \text{se } i = s \\ 0 & \text{affluenzi} \\ -f & \text{se } i = d \end{cases}$$

→ Problema facilmente estendibile al caso con più sorgenti e più destinazioni



Aggiunta di due nodi fittizi

Dove le variabili decisionali x_{ij} indicano la quantità di flusso che attraversa l'arco (i,j) appartenente ad A



Tale flusso è maggiore di zero e minore o uguale delle capacità massime dell'arco u_{ij}

Problema di Matching → Il modello causa un problema
di massimo flusso

↓
Consiste nel determinare un accoppiamento
con un numero massimo di archi su un
grafo bipartito

↓
Dal grafo bipartito, si aggiungono
due nodi fonte di sorgente e destinazione

Taglio di un grafo → Concetto legato al massimo
flusso

↓
Partizione dell'insieme V dei nodi del
grafo orientato in due sottoinsiemi $\rightarrow V = V_s \cup V_d$, $V_s \cap V_d = \emptyset$

↓
Il primo contiene il nodo di partenza, \rightarrow Sorgente $\rightarrow s \in V_s$
Il secondo contiene il nodo di arrivo \rightarrow Destinazione $\rightarrow d \in V_d$

Dai due sottoinsiemi, consideriamo V_s

↓
Definiamo gli archi diretti e inversi:

- Archi diretti: $A_{sd}^+ = \{(i,j) \in A : i \in V_s, j \in V_d\}$
↳ Archi diretti da V_s a V_d

- Archi inversi: $A_{sd}^- = \{(i,j) \in A : i \in V_d, j \in V_s\}$
↳ Archi diretti da V_d a V_s

Capacità del Tuglio

Definito come la somma delle capacità degli archi diretti del Tuglio $\rightarrow u(s, v_d) = \sum_{(i,j) \in A_{sd}^+} u_{ij}$

Valore del flusso sul Tuglio

Definito come la differenza fra la somma dei flussi sugli archi diretti e la somma dei flussi sugli archi inversi $\rightarrow x(s, v_d) = \sum_{(i,j) \in P_{sd}^+} x_{ij} - \sum_{(i,j) \in P_{sd}^-} x_{ji}$

Notiamo che: $x(s, v_d) \leq u(s, v_d)$

Possibili Tugli \rightarrow Sono moltiplici

Interessanti in modo particolare quelli di capacità minima

Indicando deboli del massimo flusso - minimo Tuglio

Sia $f(s, t)$ il valore del massimo flusso e $u(s, v_d)$ il valore del minimo Tuglio - Sommando i vincoli di bilancio dei nodi appartenenti all'insieme V_s otteniamo che il massimo flusso è pari al valore del flusso sul Tuglio

$$f(s, t) = \sum_{(i,j) \in P_{st}^+} x_{ij} - \sum_{(i,j) \in P_{st}^-} x_{ji}$$

Value del flusso \rightarrow non superiore alle capacità del Taglio



$$f(s, t) \leq u(v_s, v_t)$$

Theorem Jate del massimo flusso



Il flusso massimo da s ad è uguale alle capacità del minimo Taglio

Dobbiamo introdurre il grafo residuo \rightarrow Dato un flusso ammmissibile x , il grafo residuo

G_x è un grafo con gli stessi nodi di G , mentre gli archi e le loro capacità sono definite come:

$$\begin{cases} (i, j) \in A \quad x_{ij} < u_{ij} \Rightarrow (i, j) \in A' \quad r_{ij} = u_{ij} - x_{ij} \\ (i, j) \in A \quad x_{ij} > 0 \Rightarrow (j, i) \in A' \quad r_{ij} = x_{ij} \end{cases}$$

r_{ij} è definita come la capacità residua

Ciclo aumentante



Dato il grafo residuo G_x , un ciclo aumentante è un ciclo da s ad sul grafo residuo

Sia δ il valore minimo delle capacità degli archi di un ciclo aumentante $P(s, d)$, il value del flusso ammmissibile può essere aumentato ponendo:

$$\left\{ \begin{array}{lll} x_{ij} = x_{ij} + \delta & \text{se } (i, j) \in P & (\text{archi concordi}) \\ x_{ij} = x_{ij} - \delta & \text{se } (i, j) \in P & (\text{archi discordi}) \\ x_{ij} = x_{ij} & \text{se } (i, j) \notin P & \end{array} \right.$$

Differenza tra il massimo flusso e il flusso attualmente presente su quel dato arco

Teatrma del cammino aumentante



Il flusso ammmissible x su G è ottimo
se e solo se nel grafo residuo G_x non
esiste alcun cammino aumentante da s a d



Se esistesse un percorso P aumentante
che collega s a d , il flusso da s a d

$$f' = f + \delta$$



Il flusso x non era ottimo

Teatrma forte del massimo flusso - minimo Taglio



Stabilisce che il massimo flusso da s a d è
uguale alla capacità del taglio minimo



$$f(s, d) = \min \{ u(v_s, v_d) \}$$

Algoritmo di Ford-Fulkerson



Efficiente per la risoluzione di problemi
di massimo flusso



Potrebbe però essere lento → Pota sempre a convergenza



Introduciamo l'algoritmo Edmonds-Karp



Per tutti i possibili cammini aumentanti
determina quello con il minimo numero di archi

Schemi Ricerca Operativa

Metodi Euristicici → cercano di determinare una buona soluzione in tempi ragionevoli

Non è detto sia ottima

Esempio di problema dove si devono sfruttare le heuristiche è il TSP

Un commesso deve visitare tutte le città e ritornare al punto di partenza scegliendo il percorso più breve

→ Direttamente difficili all'aumentare del numero delle città

Definiamo il ciclo hamiltoniano

Dato un grafo $G(V, E)$, un ciclo hamiltoniano è un ciclo che attraversa tutti i nodi del grafo una e una sola volta

Associando un peso d_{uv} a ogni arco del grafo, a ogni ciclo hamiltoniano

si può associare un costo totale dato dalla somma dei pesi degli archi che lo compongono

Risolvere il TSP significa individuare il ciclo di costo minimo

$$\min \left\{ \sum_{(i,j) \in E} d_{ij} x_{ij} \right\}$$

ciclo hamiltoniano

Vincoli per TSP



- In ogni nodo j deve entrare un arco $\rightarrow \sum_{(i,j) \in E} x_{ij} = 1 \quad j \in V$

- In ogni nodo j deve uscire un arco $\rightarrow \sum_{(i,j) \in E} x_{ji} = 1 \quad j \in V$



Massimo due archi devono
incidente sul nodo j

\rightarrow tali vincoli sono detti vincoli di
assegnamento

Un ciclo hamiltoniano non
deve presentare sottogiri

- Il numero di archi che hanno origine e
destinazione sui nodi deve essere minore
uguale alla cardinalità dell'insieme dei
nodi meno 1

$$\sum_{i \in S, j \in S, (i,j) \in E} x_{ij} \leq |S| - 1 \quad \forall S \subset V: 2 \leq |S| \leq n$$

Per TSP si potrebbero potenzialmente
avere 2^n vincoli



Non sono tutti sempre necessari



Algoritmo a gerarchie di vinchi:

\rightarrow il circuito degli archi deve essere fatto
per ogni sottinsieme di V



2^n possibili sottinsiemi

- 1) Si riformula il problema \rightarrow Eliminazione dei vincoli e di risolvere
- 2) Si reintroducono i vincoli di interruzione
- 3) Check se la soluzione è ottima \rightarrow Non contiene sottocicli
- 4) Individuazione dei sottocicli
- 5) Aggiunta dei vincoli per l'eliminazione dei sottocicli
- 6) Si risolve il nuovo problema e si torna al punto 3

\rightarrow Worst case: aggiunta di 2^n vinchi

Problemi di ottimizzazione combinatoria



Introduzione:

- Ground set \rightarrow Insieme finito: $B = \{b_1, \dots, b_n\}$
- Subset system \rightarrow Famiglia di sottoinsiemi B : $\Sigma = \{S_1, \dots, S_m\}$
- $w: \Sigma \rightarrow \mathbb{R} \rightarrow$ Fungione obiettivo



Un problema di ottimizzazione combinatoria
si presenta nella forma:

$$\min w(s)$$

Il problema del cammino minimo
è un problema di ottimizzazione
combinatoria

→ Il ground set è l'insieme degli archi,
il subset system è l'insieme dei percorsi da
 s a d

Clustering \rightarrow Problema di ottimizzazione combinatoria



Dato un grafo non orientato $G = (V, E)$ e sia associato
a ogni arco un peso $w_{u,v}$, si vuole trovare le
partizioni di V in V_i classi (cluster) che minimizzino
la somma dei pesi degli archi incidenti in nodi appartenenti
alla stessa classe



Dato una partizione $\pi = \{C_1, C_2, \dots, C_n\}$
si ha che:

$$\bigcup_i C_i = V, \quad C_i \cap C_j = \emptyset \quad \forall i, j : i \neq j$$

Clustering \rightarrow Costo di un singolo cluster:

$$w(C_i) = \sum_{u,v \in C_i} w_{u,v}$$

Costo della partizione

$$w(\pi) = \sum_{i=1}^n w(C_i)$$

Consiste nel trovare le C_n partizioni a costo minimo

Problema di ottimizzazione combinatoria

- Ground set \rightarrow insieme delle coppie (v, i)

- v nodo appartenente a \mathcal{V}

- i indice di cluster dove viene inserito il nodo v

$$\mathcal{B} = \{(v, i) : v \in \mathcal{V}, i: 1, \dots, n\} \quad \mathcal{B} = \mathcal{V} \times \{1, \dots, n\}$$

- Subset system $\rightarrow \mathcal{S} = \{S_1, \dots, S_m\} \quad m \approx \mathcal{V}^n$

Sottoinsieme di \mathcal{B}

$$S_j \subset \mathcal{B}: \forall v \in \mathcal{V} \quad \exists i \in \{1, \dots, n\}; (v, i) \in S_j$$

Euristici

→ Due principali categorie di algoritmi →

- Costitutivi → Costituiscono gradualmente la soluzione
- Migliorativi

Posto I una soluzione di un dato problema ?,

EOB(I) è il valore della soluzione fornita dall'euristica e OPT(I) il valore della soluzione ottima

Partono da una soluzione ammessa e tentano di modificarla
Piccolo a migliore

Si può stabilire un errore percentuale relativo

$$\text{gap} = \frac{|OPT(I) - EOB(I)|}{|OPT(I)|} \cdot 100$$

Possiamo classificare gli algoritmi euristici secondo l'errore

- Algoritmi a errore massimo garantito
- Algoritmi con stima dell'errore

$$gap \leq \epsilon$$

Tentano una soluzione ammessa e una stima della distanza della soluzione fornita da quella ottima

Algoritmi greedy → Algoritmi che a ogni iterazione aggiungono

un pezzo alla soluzione

→ Soluzioni parziali

A ogni iterazione viene scelta la soluzione di costo inferiore

→ Si gioca sulla soluzione complessiva

Una soluzione parziale è un sottoinsieme T dell'insieme di base B che può essere

ricondotto a una soluzione ammessa appartenente all'insieme S secondo

aggiungendo altri elementi di B

→ Un algoritmo greedy costituisce una sequenza di soluzioni parziali fino ad arrivare a una soluzione ammessa

Algoritmi greedy \rightarrow Tie Breaking Rule

Regole di decisione che
mette in gioco quando ci si imbatta in "pugni" \rightarrow Più elementi che coinvolgono
il costo della nuova soluzione
pugnale

Utili nel caso di problemi di 2-clustering

L'insieme delle soluzioni pugnali T viene definito
come insieme vuoto

A ogni iterazione l'insieme viene
costituito aggiungendo delle coppie
nodo - cluster. \rightarrow In questo specifico caso
2 cluster

Algoritmi miglioratori:

\rightarrow Restituiscono un ottimo locale

Si basano su concetti di intorno
di una soluzione ammissibile corrente

Non per forza è anche
ottimo globale

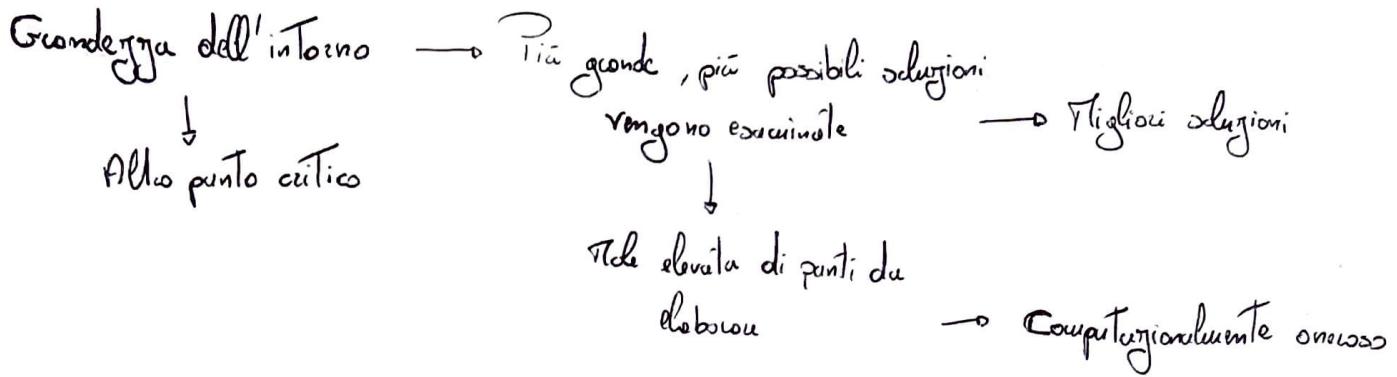
A ogni iterazione si inserisce
un nodo in uno dei due
cluster

Da una soluzione corrente $H_0 \in S$
costruiscono un intorno di H_0

Il punto di
partenza dell'algoritmo
è fondamentale

Si ipotizza H_1 tale intorno
e si calcola una soluzione migliore H_1 \rightarrow Se H_1 coincide con
 H_0 , l'algoritmo termina

Influisce sulla convergenza



Euristiche di ricerca locale



Due definizioni:

- Flusso → Operazione che a partire dalla soluzione corrente consente di generare altre soluzioni ammissibili per il problema di partenza con ulteriori caratteristiche simili alla soluzione di partenza
- Intorno di una soluzione t^* → Insieme costituito da tutte le soluzioni ottenibili applicando una determinata mossa alla soluzione t^*

Local Search per TSP



Flusso del 2-scuambio

Dato una soluzione ammissibile per il TSP, se rimuoviamo due archi non adiacenti la soluzione parziale può essere completata in due modi distinti

↓
Se lo produce un nuovo ciclo

→ Sia $H_0 = \{(u_1, u_2), \dots, (u_{q-1}, u_q), (u_q, u_1)\}$ un generico ciclo hamiltoniano. La mossa di 2-scuambio non fa altro che scegliere una coppia di archi non adiacenti (u_i, u_{i+1}) e (u_j, u_{j+1}) , rimuovere la coppia di archi dal ciclo e aggiungere due nuovi archi (u_i, u_j) e (u_{i+1}, u_{j+1})

Local Search per TSP



- 1) Si sceglie un ciclo hamiltoniano $H_0 \in \bar{T}$ e si impone $i=1$
 - Alle i -esime iterazione:
- 2) Consideriamo H_i il più corto ciclo hamiltoniano ottenibile a partire da H_{i-1} con il 2 -scambio
- 3) Se $w(H_i) \geq w(H_{i-1})$ l'algoritmo termina $\rightarrow H_{i-1}$ è il migliore ciclo trovato fino a questo punto
- 4) Si pone $i = i+1$ e si torna al passo b

Euristiche di Local Search



È migliorabile secondo due possibilità:

- Applicare l'algoritmo a partire da diverse condizioni iniziali \rightarrow Applicazione Multistart
- Aumentare la dimensione degli intorni

Multistart

→ Soluzioni iniziali scelte casualmente o costituite in maniera guidata

Se il problema presenta numerosi punti di minimo locale diventa difficile raggiungere il picco

Aumentare la dimensione degli intorni



Raggiore il Tempo di esecuzione \rightarrow Per TSP con una 1-opt la complessità sarà dell'ordine di $O(n^2)$

Tiende off TSP accuratezza e tempiistiche

→ Politiche di miglioramento

- First Improvement \rightarrow Scegli appena migliore

- Best Improvement \rightarrow Piu' andranno Tutti e poi migliora

Euristiche migliorativa Tabu Search

→ Punti migliori:

- Diversificazione → Fluoresce in nuove regioni
- Intensificazione → Attributi meno eslettivi

Per superare un minimo locale
può scegliere la migliore soluzione dell'intorno
anche se corrisponde a un
valore peggiore di funzione obiettivo

→ Potrebbe andare in "loop" riselzionando
il punto migliore

Entra in gioco lo stimatore

Funzione che permette una
rapida stima delle qualità
della soluzione

→ È necessario conservare le soluzioni passate
in modo da non riselzionarle

Liste Tabu

Impedirentale come code (FIFO)
di lunghezza L_T

Venne evitata la non visita
di intere regioni

Vengono memorizzati solo delle
caratteristiche di rilievo

→ attributi

→ T_L è un parametruo
critico

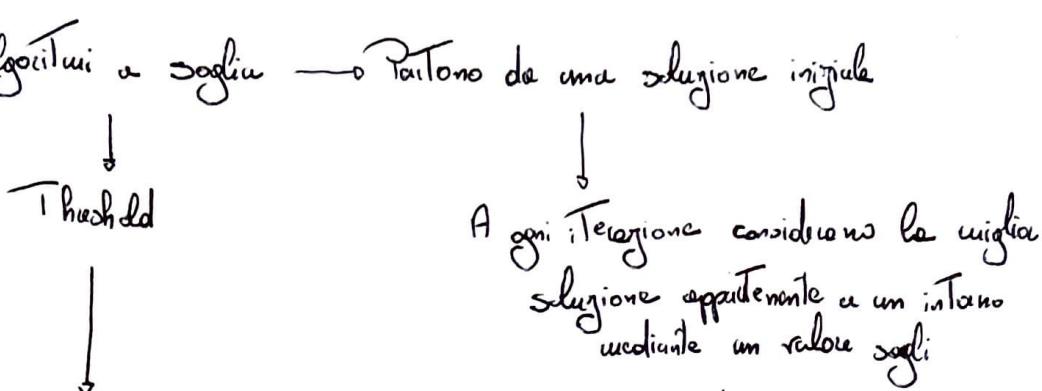
Scelto o staticamente oppure
dinamico corrente

Riass oppure variabile

Condizione d'aspettazione

Condizione che quando si
verifica permette l'esecuzione
di uno nuovo Tabu

Algoritmo a soglia \rightarrow Partono da una soluzione iniziale



Genera una soluzione $x' \in \mathcal{X}(x)$
e valuta:

$$f(x') - f(x) < T_n$$

L'algoritmo procede fino a che non si soddisfa un criterio di arresto

La soglia T_n viene modificata in caso d'opera

→ All'inizio è elevata

Serve per spostarsi nel dominio di possibilità

$$\lim_{n \rightarrow \infty} T_n = 0$$

Varietàzione effettuata in maniera probabilistica \rightarrow Euristiche Simulated Annealing

Probabilità di accettazione di soluzione

$$\rightarrow P(x') = \begin{cases} 1 & f(x') \leq f(x) \\ e^{-\frac{f(x') - f(x)}{T_n}} & f(x') > f(x) \end{cases}$$

Euristiche Simulated Annealing

Algoritmo naturale \rightarrow Processo di formazione di un solido

Soglia T_n diminuita iterazione per iterazione

Per ogni valore di T_n devono essere effettuate T_n transizioni

In base a quale ne vengono accettate si modifica il valore T_n

- 1) Si individua una soluzione iniziale s
- 2) Si definisce l'operazione che permette di individuare casualmente una nuova soluzione s'
- 3) Si calcola la probabilità di accettazione $P(x')$

Euristic Simulated Annealing



Probabilità di accettazione

- All'inizio $\rightarrow e^{-\frac{\Delta E_{\text{tot}}}{T_n}} \approx 1$
 - Alla fine $\rightarrow e^{-\frac{\Delta E_{\text{tot}}}{T_n}} \approx 0$
- } \rightarrow Variazione ottimale di T_n



$$\overline{T}_{\text{totale}} \propto k_B \rightarrow \overline{T}_{n+1} = \alpha \overline{T}_n$$



Devono essere scelti anche i valori k_B

Algoritmi genetici \rightarrow Si basano sulla selezione naturale



Stessi meccanismi

Venne codificato il problema e dunque stabilite le variabili di decisione (geni). Si rappresenta una soluzione del problema indicando i possibili valori di ciascuna variabile (allele).



Bisogna generare un insieme di possibili soluzioni che formi la popolazione iniziale



A ogni individuo viene assegnato un certo valore di fitness (funzione obiettivo).

\rightarrow Si rivelano le coppie a maggiore valore di fitness (soluzione)

Algoritmi genetici

↓
Dopo le selezioni → Generazione di nuove
selezioni

↓
Sostituzione degli
elementi della popolazione

Selezione → Vengono scelti degli elementi della
generazione corrente

→ Simulazione Flottante

Simulazione Flottante:

- Si calcola il valore cumulativo di fitness
per ogni individuo:

$$\bar{f}_i = \frac{\sum_{j=1}^i f_j}{\bar{f}}, \quad \bar{F} = \sum_{i=1}^n \bar{f}_i$$

→ Per problemi a minimizzazione si
considera $1/\bar{f}_i$

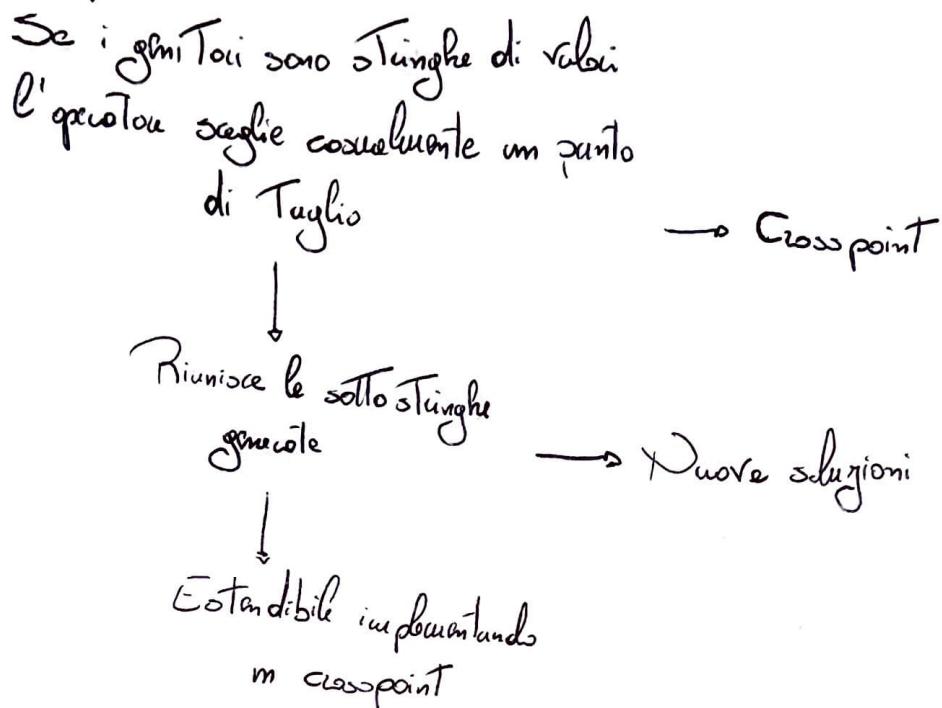
- Generati due valori casuali a e b compresi
tra 0 e 1, selezioniamo
due elementi della popolazione
se T tali che:

$$\bar{f}_{S-1} \leq a \leq \bar{f}_S \quad ; \quad \bar{F}_{T-1} \leq b \leq \bar{F}_T$$

Generazione di nuove
selezioni → Nuova popolazione

↓
Effettuata gejze a
operatori →
- Crossover
- Mutazione

Crossover → Accoppia due soluzioni generando altre due che presentano un patrimonio genetico diverso da quello dei genitori



Mutazione → Prevede di modificare casualmente il
valore di una variabile decisionale
all'interno di una stringa

Operazione necessaria → Evita il fenomeno di convergenza prematura

Dove essere effettuata
in maniera casuale
↓
Generazione completamente
casuale

↓
Individui troppo simili in
poche generazioni

Si fissa una probabilità
di mutazione per ogni gene → Simulazione Monte Carlo per valutare
la mutazione

Filtro → Operatore di mutazione particolare

↓
Transforma una soluzione inadmissibile
in una ammessa

Necessario quando uno degli altri operatori
genera una soluzione non
ammessibile

Inversione → Operatore che dà una stringa a scatti
coincidente due crosspoint, inverte l'ordine
degli ultimi parenti nelle sottostringhe
individuate

↓
Se vengono generate
stringhe non ammesse → Filtro

Si accettano → Tecniche di penalità
nella funzione obiettivo

La popolazione deve evolvere → Si devono sostituire

gli individui a
basso fitness

→ Sintesi genetica Montecarlo

} → Due possibilità
di selezione

L'algoritmo converge quando gli elementi

delle popolazioni sono tutti più
o meno simili tra loro

→ Crossover non lavora più

Schemi Ricette Operativa

Problemi di Localizzazione

Uno dei principali strumenti per la
Pianificazione Territoriale di
reti di servizio

→ Vale a dire definisce le localizzazioni di
centri di servizio che devono
soddisfare una domanda disposta
sul Territorio

Localizzazione di due Tipi:

- Puntuale → Si cerca un punto nel geofisico in cui
localizzare uno o più centri di
servizio

- Non puntuale → Si localizza sugli archi → Estensione

Tipi nei progetti di reti.

Altre caratteristiche:

- Discritti → Grafici

- Continui → Si localizza in qualsiasi punto della regione

Distanza fra chanti e centri di servizio



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

$k=1$ → Distanza Lincea

$k=2$ → Distanza euclidea

k compreso tra 1 e 2 per
realità territoriali urbane

Problemi di localizzazione



Bisogna definire i costi:

- Fissi → Costo di localizzazione

- Variabili → Costi di accesso al servizio da parte degli utenti finali

Funzione obiettivo → Possono essere diverse:

- Minimizzare i costi
- Maximizzare la copertura
- Minimizzare il massimo dei costi variabili

Modello min max

Vinchi → Copertura del servizio, capacità dei centri etc.

Simple Plant Location

→ Vale minimizzare i costi Totali

↓
Problema definito su di un grafo → Localizzazione discreta
 $G = (V, E)$

$$I \subseteq V, J \subseteq V$$

↓
Clienti ↓
Localizzazioni

Simple Plant Location

$$\rightarrow \min \left\{ \sum_{i,j} c_{ij} x_{ij} + \sum_j d_{ij} y_{ij} \right\}$$

$$\sum_j x_{ij} = 1 \quad \forall i \rightarrow \text{La domanda di ogni cliente deve essere soddisfatta}$$

$$\begin{cases} x_{ij} \leq y_j & \rightarrow \text{Un cliente può essere servito solo se il centro } j \text{ è aperto} \\ x_{ij} \geq 0 \end{cases}$$

Simple Plant Location



Al passo iniziale l'algoritmo sceglie di aprire il centro di servizio cui corrisponde il minimo valore delle scuvia. Tra il costo di apertura e i costi di affluenza al centro di servizio



Alla fine dell'iterazione viene aperto il centro di servizio v_0 cui corrisponde il massimo valore del saving:

$$S_{v_0} = \sum \max(\min\{c_{i,j} - c_{v_0}, 0\}) - f_{v_0}$$

Simple Plant Location



l'algoritmo si arresta quando non vengono più identificati saving positivi.

Se si considera una mappa di sostegno di impianto si ottiene un algoritmo di ricerca locale

Capacitated

Capacitated Plant Location



Nodi clienti caratterizzati da un valore di domanda d_i e i potenziali centri di servizio da un valore di capacità m_j

Stessi vincoli di SP2 con piccole differenze:

$$\sum_j x_{ij} = d_i \quad \forall i ; \quad \sum_j x_{ij} \leq m_j y_j \quad \forall j$$

P-mediana \rightarrow Problema che conosciute nella individuazione

di p nodi nei quali localizzare
i centri di servizio allo scopo di
minimizzare la somma dei costi
di offerta

$$F.o. \rightarrow \text{min} \sum_{i,j} c_{ij} x_{ij}$$

$$\begin{aligned} \text{Vinci} &\rightarrow \left\{ \begin{array}{l} \sum_j y_j = p \\ \text{Gli altri del SP} \end{array} \right. \end{aligned}$$

P-mediana \rightarrow Risolvibile con algoritmo Teitz-Batt

Algoritmo migliorativo basato
sul concetto di saving

Alla generica iterazione, per ogni coppia
 $i \in S$ e $j \in S^c$ viene calcolato il
saving:

$$s_{ij} = z(S) - z(S - \{i\} \cup \{j\})$$

Venne effettuata la selezione cui
corrisponde il saving minore

P-centro \rightarrow Cenzi eugeneticisti

Vede se favorire il meno possibile
gli utenti più svantaggiati

Minimizza il massimo delle
distanze: $\text{cui}-\text{max}$

Se tutti i saving sono non
negativi, l'algoritmo si
arresta

$$\text{cui} z \quad z \geq \sum_{i,j} c_{ij} x_{ij}$$

Altii vinci uguali al p-mediana

Schemi Ricerca Operativa

Clustering → Ruppazze insiem di oggetti con cui
ci dobbiamo, prima o poi, relazionare

Dato un insieme di oggetti di cardinalità elevata
ogni oggetto può essere classificato sulla base del
valore assunto da "grandezze e rappresentato
da un punto in uno spazio n-dimensionale

Cluster → Sottoinsieme di oggetti simili

↓
Caratterizzati da valori
vicini delle grandezze in
gioco

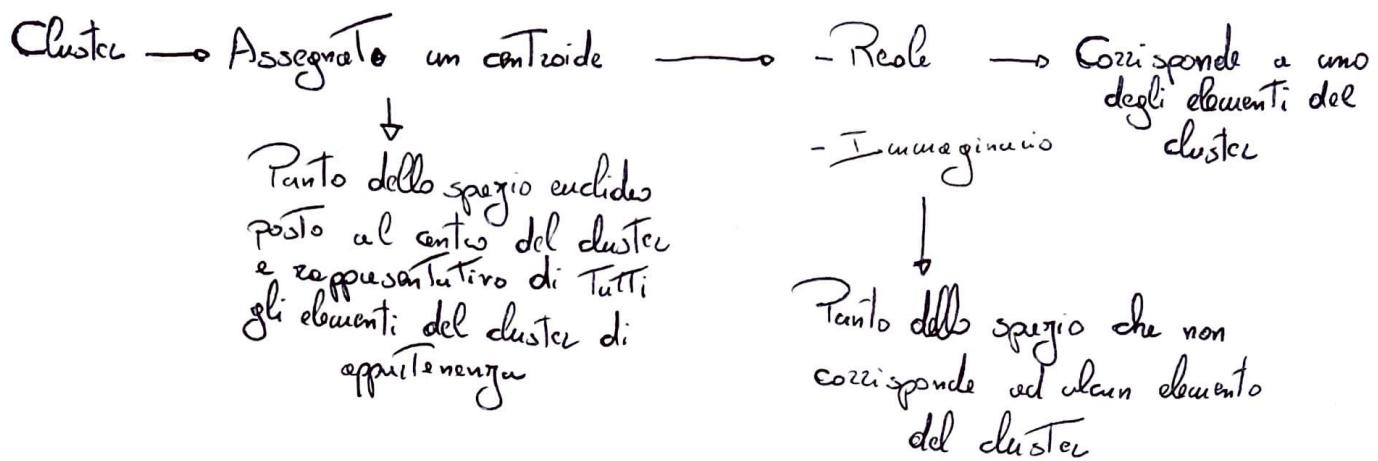
Clustering → Rappresentazione del problema mediante
un grafo $G(V, E)$

↓
Definiamo la distanza e la
dissimilitudine.

↓
Dati due punti $u = (u_1, u_2, \dots, u_n)$
e $v = (v_1, v_2, \dots, v_n)$, la loro
distanza $d(u, v) = \sqrt{\sum (u_i - v_i)^2}$
è una misura della dissimilitudine
dei due oggetti

- L'insieme dei nodi V è costituito dai punti dello spazio euclideo rappresentativi dei singoli oggetti
- L'insieme degli archi E è costituito da tutte le coppie di nodi $(u, v) \in V$
- A ogni arco $(u, v) \in V$ è associato un costo pari alla distanza euclidea fra i due nodi $d(u, v)$

↓
Si vuole trovare la partizione di V in K classi (cluster), con K prefissato, che minimizzi la somma dei pesi degli archi incidenti in nodi appartenenti a una stessa classe



Clustering con centroidi:

real



Inadattabile come problema di β -mediane in cui:

→ Algoritmo di Teitz - Bals

- Ciascuna mediana corrisponde a un centroide
- Il cluster rappresentato da un centroide è costituito da tutti nodi appartenenti al centroide



$$\min \sum d_{ij} x_{ij}$$

Minimizza la somma delle distanze di ogni nodo dal suo centroide

$$\left\{ \begin{array}{l} \sum_j y_{ij} = 1 \\ \sum_i x_{ij} = 1 \\ 0 \leq x_{ij} \leq y_{ij} \\ y_{ij} \in \{0,1\} \end{array} \right.$$

Fissa il numero di centroidi (clusters)

Ogni elemento deve appartenere a un e un solo centroide (cluster)

Un elemento i può appartenere a j se e solo se j è centroide

K -means → Algoritmo basato sul clustering



1) Seleziona K centroidi (un po' a caso)

2) Assegna a ogni elemento il centroide più vicino : $c = \arg \min \{ \text{dist}(c_i, x) \}$

3) Aggiorna la posizione dei centroidi : $c_i = \frac{1}{|R_i|} \sum x_i$ → Iduo degli elementi associati al cluster

4) Ripeti fino a convergenza → I centroidi si spostano di poco

Schemi Ricerca Operativa

Vehicle Routing Problem \rightarrow Problema di ottimizzazione combinatoria

Si vuole utilizzare una flotta di veicoli con capacità limitata per consegnare merci in determinati punti

Rappresentabile su un grafo orientato e completo $G = (V, E)$

\rightarrow TSP + Clustering

Un nodo di V è il nodo deposito, tutti gli altri sono nodi di vendita

\rightarrow La flotta parte dal nodo deposito, viaggia per i punti vendita e torna al nodo deposito

Gli archi rappresentano le distanze (costi) tra i nodi d_{ij}

Tutti i veicoli della flotta sono controllati da una pista (capacità) limitata q_i . Tutti i nodi vendita, invece, sono controllati da una domanda di merce d_i .

Vehicle Routing Problem

Risolubili con clusterizzazione \rightarrow

A ogni cluster viene assegnato un veicolo che servirà quello specifico insieme scegliendo il miglior percorso possibile

$$C = \{C_1, \dots, C_m\}$$

n veicoli

Tutti i veicoli v_h deve soddisfare la richiesta del cluster C_h

$$\sum_{i \in C_h} d_i \leq q_h$$

Tutti i veicoli tornano al deposito dopo aver consegnato la merce a disposizione (q_i)

\rightarrow Nodo deposito presente in tutti i cluster

Vehicle Routing Problem

↓
Risoluzione esatta:

Posto le famiglie di cluster: C , sia $L(C_h)$

la lunghezza del minimo ciclo hamiltoniano su

C_h . Il valore della funzione obiettivo

sarà:

$$L(C) = \sum_{h=1}^m L(C_h)$$



↳ Significa risolvere in realtà il problema di TSP

Modellazione: → Non lineare: funzione obiettivo non lineare

- $y_n = (y_{1n}, y_{2n}, \dots, y_{mn})$ → Variabili binarie alte se l' i -esimo nodo
è associato al n -esimo cluster

Vettori d'incidenza

- d_n → Puntate del n -esimo nodo

- q_n → Puntata del n -esimo veicolo

Funzione obiettivo: $\min \left\{ \sum_{n=1}^m f(y_n) \right\}$ con $f(y_n) = L(C_n)$

Vincoli:

$$\left\{ \begin{array}{l} \sum_{n=1}^m y_{1n} = m \\ \sum_{n=1}^m y_{in} = 1 \quad \forall i \\ \sum_{i=1}^m d_i y_{in} \leq q_n \quad \forall n \end{array} \right.$$

→ Ogni clustre deve contenere il nodo deposito: somma delle variabili di incidenza del deposito per il numero totale di cluster
→ A ogni clustre deve corrispondere un solo veicolo
→ Ogni veicolo non deve eccedere la propria portata massima

Vehicle Routing Problem

↓
Il modello:

- Funzione obiettivo: $\min \left\{ \sum_{v=1}^m \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ijv} \right\}$

dove: $x_{ijv} = \begin{cases} 1 & \text{se l'arco } (i,j) \text{ è percorso dal veicolo } v \\ 0 & \text{altrimenti} \end{cases}$

- Vincoli

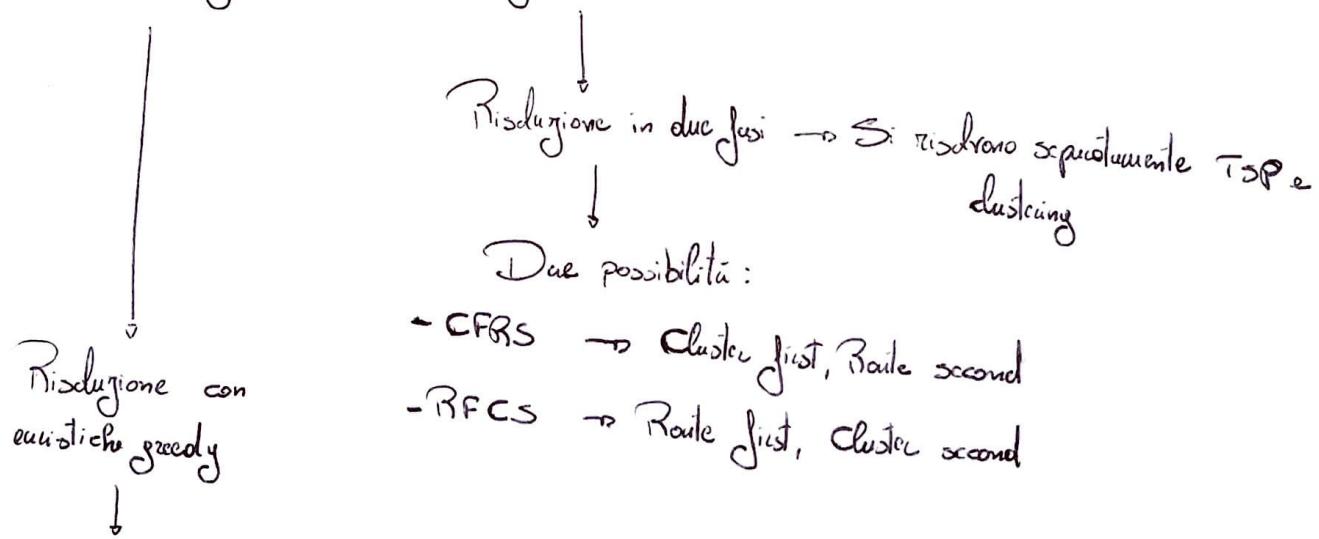
$$\left\{ \begin{array}{l} \sum_i x_{ijv} = y_{jv} \quad \forall i \\ \sum_j x_{ijv} = y_{iv} \quad \forall j \\ \sum_{i,j} x_{ijv} \leq 1 \quad \forall v \end{array} \right.$$

Alto uscente dove esce attraverso del veicolo se v è servito da v

Alto entrante: se il nodo appartiene al cluster dove esiste un arco entrante nel nodo

Quindi se l'assegna di cicli se il sottoinsieme S di V non contiene il nodo deposito: $S \subseteq V - \{v_0\}$

Vehicle Routing Problem → Risoluzione con euristiche



RFCS: Un solo veicolo può con capacità Q .
 Si determina il cammino hamiltoniano e ci informa quando si è esaurita la disponibilità.
 Si torna al deposito e parte un nuovo veicolo

Soluzione 2-opt

Euristiche CFRS



Rappresentiamo i nodi nel piano utilizzando coordinate polari: θ e δ



All'arrivo di θ si percorre il piano \rightarrow Quando si incontrano dei punti vendita si aggiungono al \star dataset fino a saturare la capacità del veicolo i :

Il dataset viene chiuso e gli si associa una spiegazione δ .

Si continua utilizzando il veicolo $i+1$



Dopo si determina il ciclo hamiltoniano del driver individuale

Queste due heuristiche non sono però quelle migliori



Più utilizzata e famosa è basata sull'algoritmo di Christofides e Wright

\rightarrow Basata sul saving e di tipo greedy



La soluzione a ogni iterazione è fatta da un certo numero di tour.

A partire da questa soluzione si cerca di unire più percorsi in uno solo.

La soluzione corrente, a ogni passo, è data da una famiglia $T = \{T_1, \dots, T_q\}$ di cicli chiusi, escluso il deposito. Ogni nodo appartiene a un ciclo, il deposito appartiene a ogni ciclo e le somme delle domande dei cicli siano non superiori alla capacità del veicolo servente.

Vehicle Routing Problem

↓
Algoritmo di Clarke and Wright

↓
Prima soluzione: ogni veicolo serve un nodo e poi
Torna al deposito

↓
Dati due tour T_h e T_v si prova
a fonderli

→ Reitkolo fino a convergenza

Deve essere ammisiabile

$$\sum_{j \in T_h - \{v_0\}} d_j \leq Q$$

Si crea un nuovo ciclo T'
eliminando due degli archi
che prima chiudevano i
due cicli

↓
Tutti saving
negativi:

Se ne inizia un nuovo che
faccia da parte T_h e i due veicoli
cicli costituendone uno nuovo

Viene valutato il saving: costo del nuovo veicolo - delle somme dei precedenti:

$$s_{h,v} = c_{v_h(h), v_i} + c_{v_i, v_p(v)} - c_{v_u(h), v_p(v)} > 0$$

↓
Fusione fattibile

Altro vincolo importante

→ Cardinalità delle flotte

↓
Il numero di tour non deve superare il
numero di veicoli disponibili

In cosa succede → Effettuano delle unioni che peggiorano la soluzione

Vehicle Routing Problem \rightarrow Tabu Search



Algoritmo di Hertz, Gendreau e Laporte

Sfutta il canotto di Pre-Soluzione \rightarrow famiglie di sottosoluzioni

$C = \{C_1, \dots, C_c\}$ con le proprietà
che ogni punto rendita sia
assegnato a un solo sottosoluzioni
 C_i .

Costuisce una sequenza di pre-soluzioni
 C_1, C_2, \dots, C_n ciascuna nell'intorno
della soluzione precedente

L'intorno viene costituito scegliendo
un punto di rendita r_i appartenente
all'insieme C_A e C_B si inserisce in un
nuovo cluster C_n .

L'insieme del quale viene prelevato
il punto rendita è uno dei qualsiasi
cluster delle pre-soluzione corrente

Se la somma delle domande
di punti rendita appartenenti a C_i
è minore delle capacità Q_i si
parla di pre-soluzione ammessa,
in caso contrario si parla di
Pre-soluzione inammissibile

\rightarrow Tale cluster C_n si inserisce
in una Tabu list

Possibili mosse



- Prelevare un nodo dal cluster C_A e
assegnato a un altro cluster
- Cominciare la costruzione di un altro
cluster a partire dal nodo deposito
e che un nodo rendita prelevato da
un altro cluster C_B

\rightarrow Il reinserimento di r_i in C_B
resta Tabu per le successive θ
mosse.

\rightarrow Si evita di lavorare scapie
su soluzioni con
lo stesso numero di
cluster

Algoritmo HGL

↓
Ogni nuova soluzione generata dall'algoritmo viene ottimizzata applicando l'heuristica 2-opt a ciascun insieme $C_i \in G$.

↓
Siamo valutando il problema classico

↓
Pre-Soluzioni e non Soluzioni

↓
A ogni pre-soluzione viene assegnata una doppia funzione obiettivo

- ↓
- Prima: classica del vehicle routing
- Seconda: Time conto delle penalizzazioni dovute alla pre-soluzione non ammessa

Prima funzione obiettivo : $J_1 = \sum_{j=1}^q L(c_j)$

↓
Somme su Tutti i cluster delle lunghezza del cicl hamiltoniano

Seconda funzione obiettivo : $J_2 = J_1 + \alpha \sum_{i=1}^n \max \{ 0, (\sum d_i) - Q \}$

↓
Time in conto il caso in cui la domanda sia maggiore delle postule

Algoritmo HGL

Per ispezionare un insieme di nodi si pone
un nodo da un generico cluster per inserirne in uno
degli esistenti oppure creare uno nuovo

→ Poi si applica l'opt

Infinite soluzioni e quindi
impossibile gestire

Per risolvere questo problema, una
volta scelto il nodo v_i non si sostituisce
in tutti i cluster bensì solo in quelli
che contengono nodi vicini al nodo v_i
selezionato

Bisogna valutare in che
posizione inserire il nodo nel
percorso del veicolo

Immediatamente prima e immediatamente dopo
al nodo v_i caso più vicino in C_n

C_n contiene lo stesso circuito hamiltoniano
a meno di una piccola modifica

Selezionare le soluzioni dell'intorno → Viene valutata solo f_2

Stabilità delle nuove soluzioni, le coppie
nodo-cluster diventano stabili

Costante di penalizzazione α → Inizializzare $\alpha = f$

- Aumentare se vi sono solo soluz. inammissibili
- Diminuire se vi sono solo soluz. ammissibili