Modelli e algoritmi per il supporto alle decisioni

kanopo

2022

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione ai modelli

Considerando un sistema reale(esistente o in fase di progetto), non è possibili agire direttamente su di esso perchè troppo costoso(tempo e danaro).
Di solito si sfruttano rappresentazioni e modelli del sistema.

1.1 Componenti dei modelli

- dati
- variabili decisionali
- vincoli
- obiettivo

1.1.1 Dati

Tutte le grandezze di un sistema reale **che non sono in nostro diretto controllo**.
Esempio della casa: Il terrono sul quale la casa deve essere edificata

1.1.2 Variabili decisionali

Tutte le granddezze del sistema reale su cui abbiamo un controllo diretto.
A diverse configurazioni di variabili decisionali corrispondono diversi output.
Esempio della casa: Grandezza delle stanze o colore pareti.

1.1.3 Vincoli

Solitamente bisogna rispettare alcuni vincolo, che inpongono limiti sui valori che possono assumere le variabili.
Esempio della casa: Metratura minima della cucina, norme che inpongo certi criteri sulla sicurezza.

1.1.4 Obiettivo

Nel assegnare valori rispettando i vincoli, siamo guidati dall'**obiettivo**.
Vogliamo i scegliere i valori per arrivare all'obiettivo nella maniera più ottimale possibile.
Esempio della casa: Spendere il meno possibilie.

1.2 Procedura di risoluzione

Dopo aver definito il modello, lo si deve risolvere.
La procedura di risoluzione sceglie i valori da assegnare alel variabili rispettando i vincoli e in modo da ottimizare il modello rispetto all'obiettivo.

1.3 Validazione del modello

Una volta ottenuta una configurazione ottimale, si esegue la validazione del modello, il modello è una rappresentazione del sistema reale quindi bisogna chiedersi se la rappresentazione è fedele oppure se si sono tralasciati alcuni aspetti.
Esempio della casa: Ci siamo dimenticati del vincolo che serve almeno una finestra per ogni stanza, ecc.
La valutazione della risoluzione permette di trovare errori e di correggere il tiro.

1.4 Tipi di modelli

Tre macro gruppi:

- modelli a scala: rappresentazione su scala ridotta
- modelli matematici: sistema tradotto in matematichese
- modelli di simulazione: sistema tradotto in informatichese

1.4.1 Modelli matematici

Vantaggio di poter usare tutti gli strumenti matematici nella risoluzione.
Attraverso questi è possibile algoritmi per la risoluzione.
Lo svantaggio dei modelli matematici è la limitata capacità espressiva.

1.4.2 Modelli di simulazione

Come vantaggi e svantaggi sono complementari ai modelli matematici, le procedure di risoluzione sono meno efficaci nel trovare la soluzione migliore ma hanno una capacità di espressione molto maggiore.

1.5 Tipi di algoritmi

Gli algoritmi trattati in questo corso sono:

- **Algoritmi costruttivi**: si costruisce la soluzione ottimale partendo da soluzioni sub incomplete.
- **Algoritmi di raffinamento locale**: si raffinano algoritmi che possono già essere accettati ma sono sub ottimali.
- **Algoritmi di enumerazione**: si enumerano le possibili soluzioni per trovare quella migliore.

1.5.1 Algoritmi costruttivi

Per gli algoritmi di costruione c'è una sub classificazione:

- **Algoritmi costruttivi senza revisione di decisioni passate**: i pezzi aggiunti non vengono più tolti
- **Algoritmi costruttivi con revisione di decisioni passate**: i pezzi aggiunti possono essere tolti

1.5.2 Algoritmi di raffinamento locale

Qui introcuciamo il concetto di "vicinanza", in genere per soluzione vicina, si intende una soluzione che differisce solo in parte da quella attuale.
Di solito si vuole che la soluzione vicina sia migliore della soluzione attuale rispetto all'obiettivo prefissato.

1.5.3 Algoritmi enumerativi

Si dividono in:

- **Algoritmi di enumerazione completi**: dove si valuta ogni singola soluzione ammissibile.
- **Algoritmi di enumerazione implicita**: dove attraverso la risoluzione di sottoproblemi, si enumerano implicitamente interi sottoinsiemi della regione ammissibile.

2 Introduzione ai grafi

2.1 Grafi

Un grafo G è costituito da una coppia di insiemi (V, A) dove:

- V è l'insieme dei **nodi**
- A è l'insieme degli **archi**

Il grafo può essere **orientato** o **non orientato**, si capisce dalle frecce presenti sulgi archi.
I grafi sono oggeti matematiche con i quali è possibile rappresentare in maniera astratta le relazioni fra le entità(nodi).
Se la relazione è simmetrica(cioè i è in relazione con j se e solo se j è in relazione con i) gli archi usati saranno privi di orientamento.
Esempio:

- la relazione "è padre di" non è simmetrica
- la relzione "è fratello di" è simmetrica

Esempio:
Avendo un grafico del tipo:

$$V = \{a, b, c, d, e\}$$

Se si parla di un grafo orientato, l'arco (a, b) è diverso dall'arco (b, a).
Se parliamo di grafi non orientati l'ordine di comparizione delle lettere dell'arco non ha nessun impatto.
Definizione:
Dato un grafo **orientato** $G = (V, A)$ e un arco (i, j):

- diremo che i è **predecessore** del nodo j.
- diremo che j è **successore** del nodo i.

Dato un grafo **non orientato** $G = (V, A)$ e un arco (i, j):

- diremo che j e i sono **adiacenti**.

2.2 Liste di adiacenza

Ogni nodo del grafo ha una lista dei suoi successori(grafo orientato) o dei suoi adiacenti(grafo non orientamento).

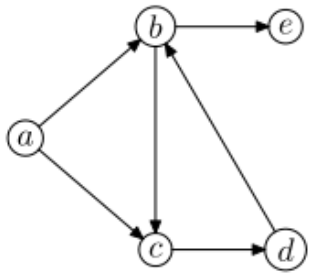

Figura 1: Esempio di grafo orientato

Le liste di adiacenza verrranno fuori a seconda dei tipi di grafo:

2.2.1 Grafo orientato

$$a : (b, c) \quad b : (c, e) \quad c : (d) \quad d : (b) \quad e : (\emptyset)$$

2.2.2 Grafo non orientato

$$a : (b, c) \quad b : (a, c, d, e) \quad c : (a, b, d) \quad d : (c, b) \quad e : (b)$$

2.3 Matrice di incidenza nodo-arco

Matrice con tante righe quanti sono i nodi e tante colonne quanti sono gli archi.

2.3.1 Grafo orientato

	(a,b)	(a,c)	(b,c)	(b,e)	(c,d)	(d,b)
a	1	1	0	0	0	0
b	-1	0	1	1	0	-1
c	0	-1	-1	0	1	0
d	0	0	0	0	-1	1
e	0	0	0	-1	0	0

2.3.2 Grafo non orientato

	(a,b)	(a,c)	(b,c)	(b,e)	(c,d)	(d,b)
a	1	1	0	0	0	0
b	1	0	1	1	0	1
c	0	1	1	0	1	0
d	0	0	0	0	1	1
e	0	0	0	1	0	0

2.4 Archi adiacenti e cammini

Due archi con un nodo in comune sono detti **adiacenti**.
Il cammino è il il percorso per arrivare da un nodo ad un'altro, e la lunghezza del cammino è il numero di archi.
Un cammino è detto **semplice** se nessun arco è percorso più di una volta, **elementare** se nessun nodo viene toccato più di una volta.

2.5 Cicli

Se il primo e ultimo nodo di un cammino sono uguali, il cammino si chiama ciclo.

2.5.1 Cammini e cicli orientati

Se gli archi sono percorsi con il loro orientamento(siamo per forza in un grafo orientato), il ciclo si chiama oreintato, altrimento non orientato.

2.6 Circuiti Hamiltoniani

Un circuito Hamiltoniano è un ciclo elementare che (orientato ne lcaso di un grafo orientato) che tocca tutti i nodi del grafo.

2.7 Componenti connesse

Dato un grafo, se posso andare da i a j, si dice che j è accessibile da i.
Un grafo si dice **connesso** se tutti i nodi sono accessibili tra di loro(il grafo ha un'unica componente connessa).

2.8 Grafo completo

Un grafo si dice completo se essite un'arco che congiunge ogni coppia di nodi distinti el grafo.
Se il grafo è orientato, devono essere presenti sia gli archi (i,j) che gli archi (j,i).

2.9 Sottografi

È un sottoinsieme degli archi dell'insieme di partenza
Esempio:
Grafo di partenza:

$$G = (V, A)$$

Se $A' \subseteq A$,

$$G' = (V, A')$$

viene detto **grafo parziale** di G.
Se il grafo è viene creato fornendo un sottoinsieme dei nodi(al posto che archi):

$$A'' = A(V'')$$

Quindi il grafo sarà:

$$G'' = (V'', A'')$$

e si denota con: **sottografo indotto** dai vertici.

2.10 Matching

Dato un grafo $G = (V, A)$ non orientato, chiamiamo matching un sottoinsieme $M \subseteq A$ dell'insieme di archi con la proprietà che ogni nodo $i \in V$, esiste al massimo una arco M avente come estremo il nodo i.

2.11 Grafi bipartiti

Un grafo $G = (V, A)$ si dice **bipartito** se l'insieme dei vertici(nodi o V) può essere partizionato in due sottoinsiemi V_1 e v_2 dove:

- $V_1 \cup V_2 = V$
- $V_1 \cap V_2 = \emptyset$

in modo che un vertice non compaia in entrambi i sottoinsiemi.
Se ogni coppia $i \in V_1, j \in V_2$ si ha che $(i, j) \in A$, allora il grafo si chiama **bipartito completo**.

2.11.1 Riconoscimento grafi bipartiti

1. Pongo $W = V, C_1 = C_2 = \emptyset$
2. Seleziono $i \in W$ e pongo $T_1 = \{i\}, c_1 = T_1$
3. Pongo $T_2 = \{k \in V \backslash C_2 : \exists i \in T_1$ tale che $(i, k) \in A$ oppure $(k, i) \in A\}$,
   - Pongo $C_2 = C_2 \cup T_2$
4. Pongo $T_1 = \{k \in V \backslash C_1 : \exists i \in T_1$ tale che $(i, k) \in A$ oppure $(k, i) \in A\}$,
   - Pongo $C_1 = C_1 \cup T_1$
5. Se $C_1 \cap C_2 \neq \emptyset$ allroa il grafo non è bipartito, altrimenti vado al passo 6.
6. Pongo $W = W \backslash (C_1 \cup C_2)$
   - se $W = \emptyset$ e $T_1 = \emptyset$, il grafo è bipartito con $V_1 = C_1$ e $V_2 = C_2$
   - se $T_1 = \emptyset$, ritorno al passo 2
   - Se $T_1 \neq \emptyset$, ritorno al passo 3

2.12 Alberi

Sia dato un grafo $G = (V, A)$ con cardinalità(numero di elementi): $card(V) = n$.
Si dice che G è un'**albero** se soddisfa le seguenti condizioni(equivalenti tra loro):

- G è privo di cicli e connesso
- G è privo di cicli e $card(A) = n - 1$
- G è connesso e $card(A) = n - 1$
- Esiste un cammino elementare che congiunge tutti i nodi

2.13 Alberi di supporto

Sia definito un grafo generico $G = (V, A)$.
Si definisce **albero di supporto** o **spanning tree** di G un grafo parziale $T = (V, A_T)$ dove $A_T \subseteq A$ che è un albero.
Un'albero di supporto deve contenere tutti i nodi, quindi si fa un sempling degli archi.

3 Minimum Spannign Tree

Nei problemi **MSP**(Minimum Spannign Tree o albero di supporto minimo), dato un grafo non orientato $G = (V, E)$ con pesi $W_{ij} \forall (i,j) \in E$, si vuole determinare tra tutti gli alberi di supporto, quello con peso complessivo $\sum_{(i,j)\in E_T} w_{ij}$ minimo.

3.1 Algoritmo greedy per MSP

1. **Inizializzazione**: Si ordinano tutti gli $m = |E|$ archi del grafo in ordine crescente rispetto al peso degli archi
   - Pongo $E_T = \emptyset$
   - pongo $k = 1$
2. Se $|E_T| = |V| - 1$ mi fermo e restituisco l'albero $T = (V, E_T)$ come soluzione, altrimenti continuo.
3. Se e_t(penso sia uno degli archi delli'insieme di archi E) non forma cicli con gli archi di E_T:
   - pongo $E_T = E_T \cup \{e_k\}$
   - altrimenti lascio invariato E_T
4. pongo $k = k + 1$ e torno al passo 2

3.1.1 Correttezza dell'algoritmo

Per dimostrare che l'algoritmo greedy non è il migliore si fa una dimostrazione per assurdo, cioè una volta terminato l'algoritmo greedy, si aggiunge un nuovo arco(formando cosi un ciclo), si leva un'arco per evitare il ciclo e si ricalcolano i pesi.
Se non vengono trovati percorsi migliori, greedy ha vinto, altrimenti si ottiene un nuovo MSP con peso minore.

3.1.2 Complessità dell'algoritmo

Solitamente l'operazione all'interno di un grafo è quella di ordinamento secondo l'ordine crescente degli archi.
Se siamo in presenza di grafi densi(numero di archi $O(|V|^2)$), l'operazione di ordinamento richiede $O(|E|\log(|E|))$ operazioni.
L'algoritmo greedy ha complessità: $O(|E|\log(|E|))$.

3.1.3 Foresta di supporto

Chiamiamo foresta di support di un grafo, un grafo parziale privo di cicli, in particolare chiamiamo albero si supporto una foresta che ha solo una componente connessa.

3.2 Algoritmo MST-1

1. **Inizializzo**:
   (a) scelgo un nodo $v_1 \in V$
   (b) pongo $U = \{v_1\}$
   (c) $E_T = \emptyset$
   (d) $c(v)$ conterra il nodo in U più vicino a v
2. se $U = V$ mi fermo
3. seleziono $\bar{v}$ con peso minore($\bar{v}$ è il nodo di $V \backslash U$ più vicino a U)
4. pongo $U \cup \bar{v}$ e $E_T = E_T \cup \{(\bar{v}, c(\bar{v}))\}$
5. $\forall v \in V \backslash U$ se il peso di $v\bar{v}$ è minore del peso di $vc(\bar{v})$, pongo $c(v) = \bar{v}$ e torno al passo 2

3.2.1 Complessità dell'algoritmo

Complessità: $O(|V|^2)$
Molto buono per grafi densi.

3.3 Algoritmo MST-2

- **Inizializzazione** Poni $E_T = \emptyset$ e sia $\mathcal{C} = \{S_1, \ldots, S_n\}$ con $S_i = \{v_i\}$ per ogni $i = 1, \ldots, n$ ($\mathcal{C}$ nel corso dell'algoritmo conterrà la collezione di componenti connesse di (V, E_T) ed essendo inizialmente E_T vuoto, viene inizializzata con n componenti, una per ciascun nodo del grafo). Poni

  $$componente[v_j] = S_j \quad j = 1, \ldots, n.$$

  ($componente[v]$ restituisce la componente connessa a cui appartiene il nodo v durante l'algoritmo).
- **1** Se $| \mathcal{C} | = 1$, allora STOP.
- **2** Per ogni $S_i \in \mathcal{C}$, poni $min[i] = \infty$.
- **3** Per ogni $(u, v) \in E$, siano $S_i = componente[u]$ e $S_j = componente[v]$. Se $i \neq j$, allora:
  - $w_{uv} < min[i] \Rightarrow shortest[i] = (u, v)$, $min[i] = w_{uv}$;
  - $w_{uv} < min[j] \Rightarrow shortest[j] = (u, v)$, $min[j] = w_{uv}$;

  (si noti che $shortest[i]$ indica l'arco a peso minimo tra quelli con un solo nodo in S_i).
- **4** Per tutti gli $S_i \in \mathcal{C}$ poni

  $$E_T = E_T \cup \{shortest[i]\}.$$
- **5** Poni $\mathcal{C}$ pari all'insieme di componenti connesse di (V, E_T) e aggiorna i valori $componente[v]$ per ogni $v \in V$. Torna al Passo 1.

3.3.1 Complessità

Complessità: $O(|E| \log(|V|))$

4 Shortest Path

Nei problemi di cammino a costo minimo, dato un grafo orientato $G = (V, A)$ che ha un costo per ogni arco, si vuole individuare un percorso con il mino costo possibile.

4.1 Algoritmi per Shortest Path

usiamo due algoritmi:

- **Dijksta**: valido solo se i costi dei percorsi sono maggiori o uguali a 0 per ogni percorso.
- **Floyd-Warshall**: valido anche er distanze negative

Per i prossimi algoritmi si supporrà che:

- il peso dei percorsi non appartenenti all'insieme degli archi sia $+\infty$
- la cardinalità dei vertici è n

4.2 Algoritmo di Dijkstra