Il problema dello Zaino e problem di Ottimizzazione su Grafi

Riferimento: Capitolo 12 del testo di Python consigliato
“Knapsack and Graph Optimization Problems”

Problemi di Ottimizzazione

In generale un **problema di ottimizzazione** ha due parti:

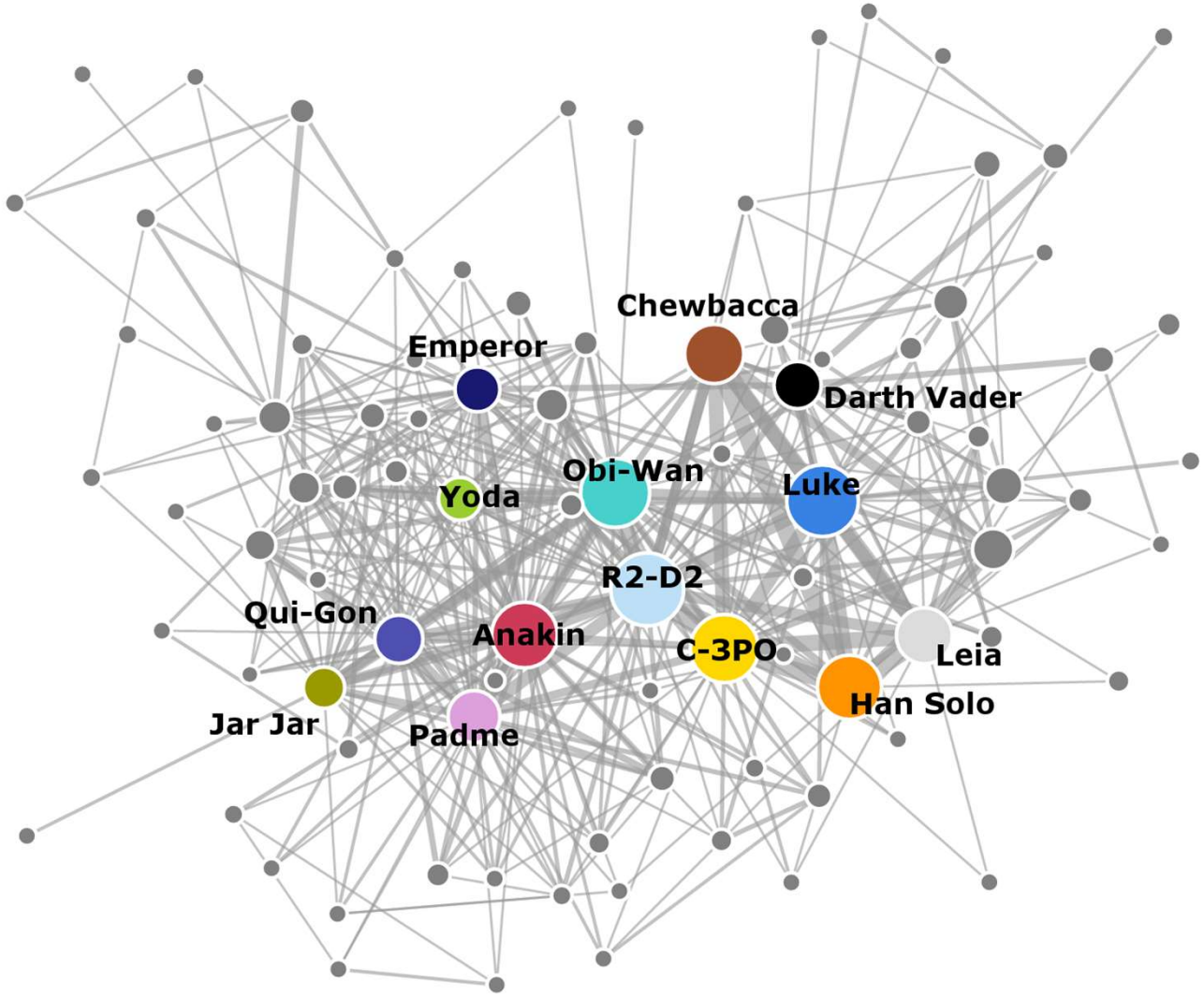
1. Una **funzione obiettivo** che deve essere massimizzata o minimizzata
2. Un **insieme di vincoli** (eventualmente vuoto) che devono essere rispettati.

Si distinguono i problemi di ottimizzazioni “continui” in cui le variabili sono numeri reali, dai problemi “discreti” in cui le variabili possono essere solo numeri interi. Nel caso in cui le variabili possono assumere solo i valori 0 o 1, si parla di problemi di **ottimizzazione combinatoria**.

Take away message

1. Molti problemi reali possono essere formulati in modo relativamente semplice come **problemi di ottimzzazione** che possono poi essere risolti con un calcolatore
2. Ridurre un nuovo problema ad un problema conosciuto permette l'utilizzo di soluzioni esistenti
3. Il **problema dello zaino** e i **problemi su grafi** sono esempi classici di problemi «core», che appaiono spesso come sottoproblemi
4. **Enumerazione esaustiva** è sempre un opzione che vale la pena valutare, ma spesso non è praticabile
5. Gli **algoritmi euristici** (golosi) offrono spesso un buon compromesso tra qualità della soluzione e velocità di calcolo

Star Wars Social Network



Recommended reading: http://evelinag.com/blog/2015/12-15-star-wars-social-network/

Grafi e Digrafi

Un **grafo non orientato** è una tripla **(V, E, Ψ)** dove **V** e **E** sono insiemi finiti e

$$\Psi : \mathrm{E} \rightarrow \{X \subseteq V : |X| = 2\}$$

Gli elementi di V sono chiamati **vertici**, gli elemeti di E **lati**.

Un **grafo orientato o digrafo** è una tripla **(N, A, Ψ)** dove **N** e **A** sono insiemi finiti e

$$\Psi : \mathrm{A} \rightarrow \{(v, w) \in N \times N : v \neq w\}$$

Gli elementi di N sono chiamati **nodi**, gli elemeti di A **archi**.

I lati, o archi, *e, f* con e \= f e $\Psi(e) = \Psi(f)$ sono chiamati lati (archi) paralleli. Grafi senza archi paralleli sono chiamati grafi semplici.

Esempi di problemi di ottimizzazione su grafi

1. **Cammino Minimo**. Dati due nodi $v, w \in N$ trovare la sequenza più corta di lati in modo tale che il nodo sorgente del primo arco sia *v*, la destinazione dell'ultimo lato sia *w* e per tutti gli altri lati in sequenza la destionazione di un lato e la sorgente del successivo.

2. **Cammino Minimo Pesato**. In questo caso si considera una funzione che assegna ad ogni lato un costo (peso) e si vuole trovare il cammino di peso minimo, ovvero la sequenza la cui somma dei costi associati ai lati sia minima.

Esempi di problemi di ottimizzazione su grafi

3. **Cricca Massima**. Una cricca (clique) è un sottoinsieme di vertici $K \subseteq V$ tale per cui esiste un lato per ogni coppia di vertici in *K*. La cricca massima è la cricca di cardinalità massima in un grafo G=(V, W).

4. **Taglio Minimo**. Dati due sottoinsiemi di vertici $\mathrm{A}, \mathrm{B} \subseteq V$ in un grafo, un "**taglio**" è l'insieme di lati la cui rimozione elimina tutti i cammini da ogni vertice di A ad ogni vertice di B. Il taglio minimo è il sottoinsieme di lati più piccolo che sconnette A da B.

Rappresentazione di un grafo 1/3

Un **grafo non orientato** può essere rappresentato con una matrice con una riga per ogni vertice e una colonna per ogni lato. La **matrice di incidenza** di un grafo non orientato *G=(V,E)* è:

$$A = (a_{ve}), \text{con v} \in \text{V}, \text{e} \in \text{E}$$

dove $(a_{ve} = 1)\ se\ e \in V, (a_{ve} = 0)\ altrimenti.$

La **matrice di incidenza** di un grafo orientato *G=(N,A)* è:

$$A = (a_{na}), \text{con n} \in \text{N}, \text{a} \in \text{A}$$

dove $(a_{na} = 1)\ se\ a = (i,j)\ e\ n = i, (a_{na} = -1)\ se\ a = (i,j)\ e\ n = j, (a_{na} = 0)\ altrimenti.$

Lo spazio richiesto per memorizzare la matrice è *O(nm)*, con *n* numero di vertici, *m* numero di lati.

Rappresentazione di un grafo 2/3

Un **grafo non orientato** può essere rappresentato con una matrice con una riga e una colonna per ogni vertice. La **matrice di incidenza (quadrata)** di un grafo non orientato *G=(V,E)* è:

$$A = (a_{vw}), \text{con v, w} \in \text{V}$$

dove $(a_{vw} = 1)\ se\ (v, w) \in E$, $(a_{vw} = 0)$ altrimenti.

Lo spazio richiesto per memorizzare la matrice in questo casto è *O(n^2)*.

Rappresentazione di un grafo 3/3

Un **grafo non orientato** può essere rappresentato con una lista di adiacenza, in cui per ogni nodo si ha la lista di archi incidenti in quel nodo.

Quale struttura dati utilizzare per memorizzare nel calcolatore un grafo non è scontato e dipende dal tipo di problema.

List Graph 1/2

```
class Vertex(object):
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return self.name

class Edge(object):
    def __init__(self, v, w):
        self.v = v
        self.w = w

    def __str__(self):
        return ''
```

ESERCIZIO: creare una sottoclasse di Edge chiamata WeightedEdge
che memorizzi anche il costo del lato n self.v + '->' + self.w

List Graph 1/2

```
class ListGraph(object):
    def __init__(self):
        self.nodes = []
        self.edges = {}

    def addVertex(self, node):
        if node in self.nodes:
            raise ValueError('Nodo già inserito')
        else:
            self.nodes.append(node)
            self.edges[node.name] = []

    def addEdge(self, edge):
        if edge.v not in self.nodes
           or edge.w not in self.nodes:
            raise ValueError('Uno dei due estremi..')
        self.edges[edge.v].append(edge)
        self.edges[edge.w].append(edge)

    def neighbourOf(self, node):
        return self.edges[node]
```

Esercizio: Distanza in una rete sociale

Una **rete sociale** è composta da individui e di relazioni (e.g. di amicizie) tra individui. Questa rete può essere rappresentata da un **grafo**, in cui *ogni nodo rappresenta un individuo e ogni lato rappresenta una relazione*.

Se la relazione è **simmetrica** (e.g., facebook), i lati sono non orientati, se la relazione è **asimmetrica** (e.g., twitter) gli archi sono orientati.

ESERCIZIO: Data la struttura dati delle slide precedenti, scrivere un algoritmo che permetta di trovare il CAMMINO MINIMO tra un dato nodo sorgente e un dato nodo destinazione. Si consiglia di pensare ad una funzione ricorsiva che permetta di risolvere il problema.

Discutere la complessità dell'algoritmo trovato.

Esempi di reti

Stanford Large Network Dataset Collection
Link: https://snap.stanford.edu/data/

Amazon networks : nodes represent products and edges link commonly co-purchased products (nota: pensare agli esercizi sulla predizione di valutazioni)