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

Il problema dello zaino

	Valore	Peso
Orologio	175	10
Quadro	90	9
Radio	20	4
Vaso	50	2
Libro	10	1
Computer	200	20

Dato uno zaino che riesce a portare al massimo 20 unità di peso, quali oggetti prendo per massimizzare la somma dei valori degli oggetti selezionati?

Algoritmo greedy (goloso)

Dato uno zaino che riesce a portare al massimo 20 unità di peso, quali oggetti prendo per massimizzare la somma dei valori degli oggetti selezionati?

1. Scarto tutti gli oggetti di peso superiore alla capacità massima (**preprocessing**)
2. **Ordina** gli oggetti per un dato criterio
3. **Seleziona** gli oggetti uno alla volta sino a quando viene rispettato il vincolo di zaino (vincolo di capacità)
4. Restituisci il valore della soluzione e l'insieme di oggetti selezionati

Algoritmo greedy (goloso)

Dato uno zaino che riesce a portare al massimo 20 unità di peso, quali oggetti prendo per massimizzare la somma dei valori degli oggetti selezionati?

1. Scarto tutti gli oggetti di peso superiore alla capacità massima (**preprocessing**)
2. **Ordina** gli oggetti per un dato criterio
3. **Seleziona** gli oggetti uno alla volta sino a quando viene rispettato il vincolo di zaino (vincolo di capacità)
4. Restituisci il valore della soluzione e l'insieme di oggetti selezionati

Algoritmo greedy (goloso)

```
B = 20 # Capacità massima
Ls = [(175,10),(90,9),(20,4),(50,2)(10,1),(200,20)]
Ls = filter(lambda x: x[1]<=B, Ls)     # Passo 1
Ls = sorted(Ls, key=lambda x: x[1]) # Passo 2

fobj, weight, sol = 0, 0, []          # Passo 3
for item in Ls:
  if weight + item[1] <= B:
    fobj = fobj+ item[0]
    weight = weight + item[1]
    sol.append(item)

print(value, sol)                     # Passo 4
```

Il problema dello zaino

	Valore	Peso	Valore/Peso
Orologio	175	10	17.5
Quadro	90	9	10
Radio	20	4	5
Vaso	50	2	25
Libro	10	1	10
Computer	200	20	10

Dato uno zaino che riesce a portare al massimo 20 unità di peso, quali oggetti prendo per massimizzare la somma dei valori degli oggetti selezionati?

Algoritmo greedy (goloso)

```
def KnapsackGreedy(Items, B, OrderFun):

  Ls = filter(lambda x: x > B, Items) # Passo 1

  Ls = sorted(Ls, key=OrderFun)       # Passo 2

  fobj, weight, sol = 0, 0, []        # Passo 3
  for item in Ls:
    if weight + item[1] <= B:
      fobj = fobj+ item[0]
      weight = weight + item[1]
      sol.append(item)

  return (value, sol)                 # Passo 4
```

Soluzione esatta

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

Algoritmo greedy (goloso)

```
def KnapsackGreedy(Items, B, OrderFun):

  Ls = filter(lambda x: x[1]<=B, Items) # Passo 1

  Ls = sorted(Ls, key=OrderFun)         # Passo 2

  fobj, weight, sol = 0, 0, []          # Passo 3
  for item in Ls:
    if weight + item[1] <= B:
      fobj = fobj+ item[0]
      weight = weight + item[1]
      sol.append(item)

  return (fobj, sol)                    # Passo 4
```