Università degli studi di
Milano-Bicocca

Decision Models

Final Project

Santa's uncertain Bags

Authors:
Dario Carolla - 807547 - d.carolla@campus.unimib.it
Matteo Licciardello - 799368 - m.licciardello@campus.unimib.it

July 15, 2019

Abstract

La massimizzazione della quantità di merce da trasportare rappresenta un grosso problema per numerose società in tutto il mondo e persino Babbo Natale non è esente da tale problema: al fine di consegnare i regali, necessita di riempire i propri sacchi il più possibile. Per aiutare Babbo Natale è stato affrontato un problema di massimizzazione in cui sono presenti molteplici contenitori soggetti a delle limitazioni, quali la capacità massima ed il numero massimo di elementi inseribili per contenitore. Il problema è stato analizzato e sono state individuate diverse soluzioni sia per mezzo di un algoritmo greedy che tramite l'ausilio un algoritmo genetico.

1 Introduzione

Il presente progetto è ispirato ad una competizione postata sul sito Kaggle denominata *Santa's uncertain bags*. Il problema proposto consiste nell'aiutare Babbo Natale nel riempire i sacchi contenenti i regali in seguito ad un problema verificatosi: dopo aver preparato tutti i regali, Babbo Natale, ha smarrito i pesi di questi ultimi. Fortunatamente, sono state calcolate delle distribuzioni di probabilità per approssimare il peso di ogni tipologia di giocattolo. I regali devono essere posti all'interno di mille sacchi i quali, però, sono soggetti a delle limitazioni di peso e di quantità di doni contenibili: in ogni sacco possono essere inseriti al più nove regali ed il peso complessivo di ognuno di essi non può superare cinquanta libbre. La capacità complessiva dei contenitori risulta calcolabile semplicemente moltiplicando la capacità dei singoli sacchi per il numero di essi.

$$NumeroSacchi \cdot Capacità = 1000 \cdot 50 = 50000$$

Per verificare di che tipo di problema si tratti, è stato necessario effettuare la sommatoria dei pesi di tutti i regali, che risulta essere 50099.57. Il risultato ottenuto è maggiore della capacità complessiva dei sacchi, dunque, il problema consiste nella massimizzazione del peso totale dei contenitori; in caso contrario, lo scopo sarebbe stato minimizzare il numero complessivo dei sacchi utilizzati.

Il problema della massimizzazione del peso contenuto all'interno di un sacco è conosciuto come ***bin-packing problem***. Lo scopo di questo problema è, dato il peso di ogni elemento e la capacità massima dei sacchi, assegnare ogni elemento ad un contenitore in modo che il peso totale degli

articoli in ogni contenitore non superi la capacità massima e il numero di contenitori utilizzati sia il minimo [1]. Il problema considerato, dunque, è una generalizzazione del bin-packing problem, in quanto, come accennato precedentemente, lo scopo non è minimizzare il numero di sacchi, ma massimizzare i pesi per tutti quelli disponibili.

2 Dataset

I dati utilizzati sono stati offerti dalla competizione di Kaggle. È stato fornito un file *CSV* denominato *gift.csv* all'interno del quale sono presenti tutti i regali che Babbo Natale deve consegnare. I nomi dei regali sono strutturati nel seguente modo:

NomeRegalo_NumeroRegalo

Oltre ai nomi dei regali, sono state fornite le distribuzioni di probabilità dei loro pesi:

- horse = $\max(0, N(5,\ 2))$
- ball = $max(0, 1 + N(1,\ 0.3))$
- bike = $max(0, N(20,\ 10))$
- train = $max(0, N(10,\ 5))$
- coal = $47 * B(0.5,\ 0.5)$
- book = $X^2(2)$
- doll = $\Gamma(5,\ 1)$
- block = $triangle(5, 10, 20)$
- gloves =$3.0 + U(0,\ 1),\ if\ U(0,\ 1) < 0.3\ else\ Ricalcola$

Per ciascun regalo sono stati calcolati i pesi impostando un seme globale per rendere replicabile la creazione dei dati a partire dalle distribuzioni; successivamente, tutti i valori ottenuti sono stati aggiunti ad una nuova colonna *weight* del dataset. Per eseguire questa operazione è stato utilizzato il *pattern matching* per riconoscere i nomi delle varie tipologie di regali ed applicare la

giusta distribuzione. Dopo aver generato il dataset, non è stato necessario effettuare ulteriori operazioni di pulizia sui dati in quanto non erano presenti particolari anomalie. Il dataset risulta composto da 2 colonne e 7166 osservazioni. Di seguito è stato rappresentato un estratto del dataset finale contenente un esempio per ogni tipologia di regalo [Tab.1].

GiftId	**Weight**
horse_2	4.7810
ball_58	1.9298
bike_103	17.4134
train_883	11.0597
coal_116	26.9737
book_34	1.2422
doll_402	5.5358
block_77	10.2937
gloves_27	3.2447

Table 1: Estratto dataset

I dati sono stati analizzati attraverso una rappresentazione grafica. È stato possibile osservare, in particolar modo, la distribuzione dei pesi per le varie tipologie di giocattolo [Fig.1].

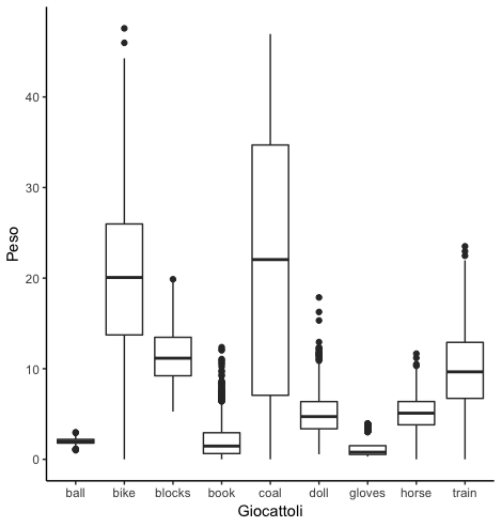

Figure 1: Boxplot distribuzione dei pesi per tipologia di regalo

Dal presente grafico si è potuto evincere che tutti i giocattoli hanno un peso inferiore a cinquanta, dunque, tutti possono essere inseriti all'interno dei sacchi. Inoltre, numerosi oggetti hanno un peso molto vicino allo zero.

3 Approccio metodologico

La formulazione matematica che permette il conseguimento della soluzione ottima per il problema proposto è la seguente:

$$Max \sum_{i=1}^{n}\sum_{j=1}^{m} w_i \cdot x_{ij} \tag{1}$$

$$s.t. \quad \sum_{i=1}^{n} w_i \cdot x_{ij} \leq 50 \quad \forall j = 1, ..., m \tag{2}$$

$$\sum_{j=1}^{m} x_{ij} \leq 1 \quad \forall j = 1, ..., m \tag{3}$$

$$x_{ij} \in [0, 1] \quad \forall j = 1, ..., m; \ \forall j = 1, ..., m$$

dove x_{ij} rappresenta una variabile decisionale binaria che assume valore 1 nel caso in cui il regalo i sia stato assegnato al sacco j, 0 altrimenti; w_i costituisce la variabile peso associata all'oggetto i; n si riferisce al numero totale dei regali presenti nel dataset (=7166); m rappresenta il numero totale di sacchi utilizzabili (=1000). La funzione obiettivo, quindi, consiste nel massimizzare il peso complessivo degli oggetti i inseriti nei sacchi j, rispettando i vincoli associati. Il vincolo (2) rappresenta la capacità del sacco j ovvero 50, mentre il vincolo (3) impone che ogni regalo possa essere assegnato ad un solo sacco.

3.1 Algoritmi greedy

Il primo metodo implementato rappresenta una soluzione efficiente al problema proposto, ma sicuramente non ottima. Questo algoritmo sviluppato appartiene alla tipologia degli algoritmi ***greedy***, ovvero algoritmi utilizzati per risolvere problemi in cui, per giungere ad un risultato, deve essere eseguita una sequenza di passi o scelte per ottenere la soluzione ottimale. Gli algoritmi greedy sono un metodo con cui si determina una soluzione basata

sulla scelta ottima a livello locale [2]. In altre parole, questa tipologia di algoritmi, si basa sulla strategia dell'ingordo, ovvero, quella di compiere, ad ogni passo, la scelta migliore nell'immediato piuttosto che adottare una strategia a lungo termine.

Formalmente, dato un insieme OPT di soluzioni ottime locali, un insieme di elementi X rappresentanti la soluzione finale ed un possibile elemento della soluzione $\{x\}$ si verificano i casi descritti in (4):

$$\begin{cases} \mathrm{X} := \mathrm{X} \cup \{x\} \text{ se } x \in OPT \\ \mathrm{X} := \mathrm{X} \; se \; \{\mathrm{x}\} \notin OPT \end{cases} \tag{4}$$

Tale modalità di funzionamento risulta particolarmente efficiente a livello computazionale ed è sicuramente la via più rapida per l'implementazione di un algoritmo, non viene garantita in alcun modo la convergenza verso la soluzione ottima globale, ma può fornire ugualmente una buona approssimazione.

Il primo algoritmo implementato, è una variante dell'algoritmo *First-Fit*, il quale viene spesso utilizzato per risolvere il problema del *bin-packing problem*: il fine del First-Fit è minimizzare il numero di contenitori utilizzati per accogliere gli oggetti considerati. Il codice è stato adattato al problema nel seguente modo:

```
for (tutti i regali i = 1, 2, . . . , n) do
    for (tutti i sacchi j = 1, 2, . . . , m) do
        if (oggetto i contenibile nel sacco j AND
            nel sacco j ci sono meno di 9 oggetti)
            inserisce oggetto i nel sacco j.
        end if
    end for
end for
```

L'algoritmo First-Fit affronta il problema del riempimento dei sacchi provando ad inserire i regali, nell'ordine in cui gli vengono posti, nel primo contenitore in cui essi possano rientrare. La procedura appena descritta ottiene buoni risultati con un'ottima efficienza, tuttavia, questo algoritmo fa parte degli algoritmi greedy, dunque, la resa è fortemente dipendente dall'ordine con cui vengono forniti in input gli elementi. Tendenzialmente,

per ottenere un buon risultato, è necessario disporre i regali in ordine decrescente rispetto al peso. Pertanto, si è deciso di modificare l'algoritmo applicando questo ordinamento che permette di migliorare l'ottimo trovato in precedenza, ma continua a non restituire l'ottimo globale.

3.2 Genetic Algorithm

Per ottenere la sequenza ottima da fornire in input al First-Fit adattato è stato utilizzato il *genetic algorithm*. La programmazione genetica è un approccio di problem-solving che si basa sul principio darwiniano della riproduzione e della sopravvivenza dei più adatti ed utilizza operazioni genetiche naturali come il crossover e la mutazione [3]. L'idea alla base degli algoritmi genetici è quindi quella di selezionare le soluzioni migliori e di ricombinarle fra loro, in maniera tale che esse possano convergere verso un punto di ottimo.

Per poter comprendere al meglio il funzionamento degli algoritmi genetici è doveroso chiarire l'utilizzo della terminologia utilizzata:

- Cromosoma: rappresenta una singola soluzione al problema considerato;
- Popolazione: rappresenta l'insieme di soluzioni relative al problema considerato;
- Gene: è una parte di un cromosoma, dunque, rappresenta un sottoinsieme della singola soluzione;
- Fitness: funzione appositamente progettata per valutare una soluzione;
- Crossover: meccanismo utilizzato per generare una nuova soluzione mescolando delle soluzioni esistenti;
- Mutazione: meccanismo utilizzato per simulare un'alterazione casuale di una soluzione.

Il procedimento utilizzato dall'algoritmo genetico per generare nuove popolazioni si compone dei seguenti passi [Fig.2]:

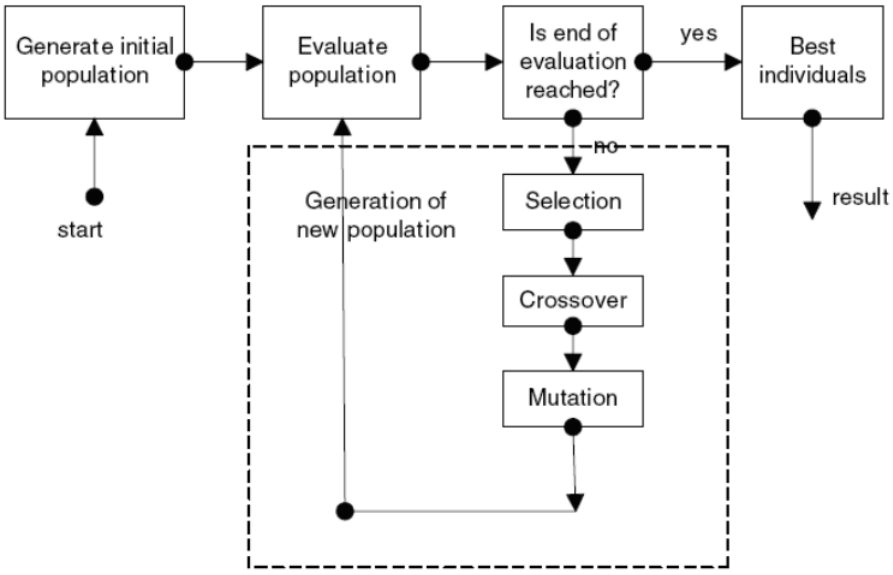

Figure 2: Flusso algoritmo genetico

1. Definizione di un primo insieme casuale di possibili soluzioni al problema in esame;

2. Valutazione di ogni soluzione, associando a ciascuna un valore proveniente dalla funzione di fitness in modo da poterle ordinare;

3. Definizione di un nuovo gruppo di soluzioni modificando opportunamente le soluzioni con valore di fitness elevato, in modo da favorire lo sviluppo a scapito di quelle peggiori;

4. Se il numero di iterazioni stabilite è stato raggiunto, o la qualità della migliore soluzione disponibile è superiore agli standard richiesti, si può terminare l'elaborazione, altrimenti si ritorna al passo 2 per definire un nuovo gruppo di soluzioni.

L'algoritmo genetico è stato implementato utilizzando un'apposita libreria di R, denominata *GA* [4], nel seguente modo:

```
GA <- ga(type = "permutation",
         fitness = firstFit_ga,
         weight = gift$weight,
         lower = 1,
         upper = 7166,
         popSize = 100,
         maxiter = 50,
         run = 20,
         pmutation = 0.2,
         keepBest = TRUE,
         monitor = TRUE)
```

La funzione di fitness fornita in input all'algoritmo genetico è la First-Fit precedentemente definita, la quale, come anticipato, gode di ottima efficienza, ma è limitata dall'ordine in cui vengono forniti i dati. L'algoritmo genetico permettere di superare tale difetto generando un elevato numero di combinazioni convergenti verso la soluzione ottima.

3.3 Multi Knapsack Problem

Come anticipato nella sezione introduttiva [Cap.1], il problema affrontato è una variante del *bin-packing problem*, il quale, a sua volta, è un sottoproblema del ***Knapsack Problem***. Si tratta di un problema di ottimizzazione combinatoria posto nel seguente modo: siano dati uno zaino di una determinata capacità ed N oggetti, ognuno dei quali caratterizzato da un peso ed un valore. Il problema richiede di massimizzare il valore degli oggetti posti nello zaino senza eccedere la capacità di quest'ultimo. Il knapsack, nella sua variante multipla, è caratterizzato dalla presenza di più zaini nei quali poter trasportare gli oggetti.

Il Knapsack Problem è risolvibile tramite tecniche di programmazione dinamica, la quale consiste nella suddivisione del problema in sottoproblemi e nell'utilizzo di sottostrutture ottimali [5].

L'utilizzo di tale approccio è stato considerato ai fini della risoluzione del problema, tuttavia la programmazione dinamica necessita dell'utilizzo di numeri interi o, almeno, numeri con cifre decimali uguali tra loro. I dati a disposizione non posseggono queste caratteristiche. L'unico modo per utilizzare

la programmazione dinamica sarebbe stato applicare un'approssimazione sui dati a disposizione ma, trattandosi di numeri generati da distribuzioni statistiche, si è preferito non attuare un'ulteriore approssimazione.

4 Risultati e valutazioni

Tutti gli algoritmi presentati basano il proprio risultato sui pesi dei regali presenti nel dataset e considerano i vincoli imposti dal problema. Il risultato finale è costituito dal peso totale dei regali che ciascun algoritmo ha posizionato nei sacchi a disposizione. I risultati ottenuti sono descritti nella tabella [Tab. 2].

Algoritmo	**Peso totale**
First-Fit	43127 lb
First-Fit ordinato	47527 lb
Genetic Algorithm	49507 lb

Table 2: Risultati finali

5 Discussione dei risultati

I tre algoritmi implementati ottengono buoni risultati. Nel caso peggiore, ovvero per il First-Fit non ordinato, viene raggiunto un peso totale di 43127 libbre. Dal grafico [Fig. 3] risulta evidente come il riempimento non sia costante e questo accade perché gli elementi più pesanti, che sono più difficili da allocare, possono essere selezionati quando lo spazio disponibile non è più sufficiente.

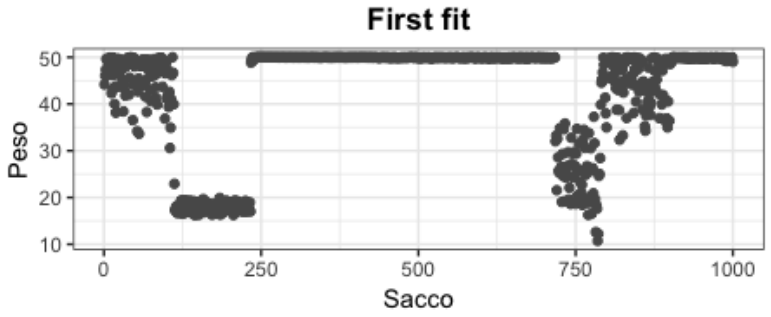

Figure 3: Peso contenuto per sacco

Infatti, in seguito ad un semplice ordinamento dei dati, senza cambiare l'algoritmo di base, il risultato iniziale viene migliorato fino a 47527 libbre ed il riempimento dei sacchi risulta più costante [Fig. 4].

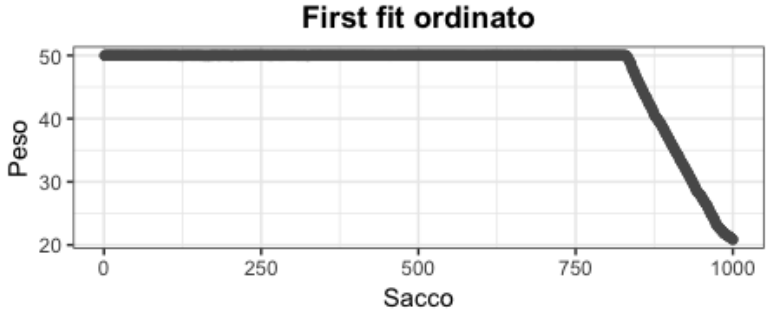

Figure 4: Peso contenuto per sacco

Infine, il caso migliore si ottiene applicando l'algoritmo genetico, il quale, testando le possibili combinazioni, converge verso un peso ulteriormente migliorato come mostrato in figura [Fig. 5].

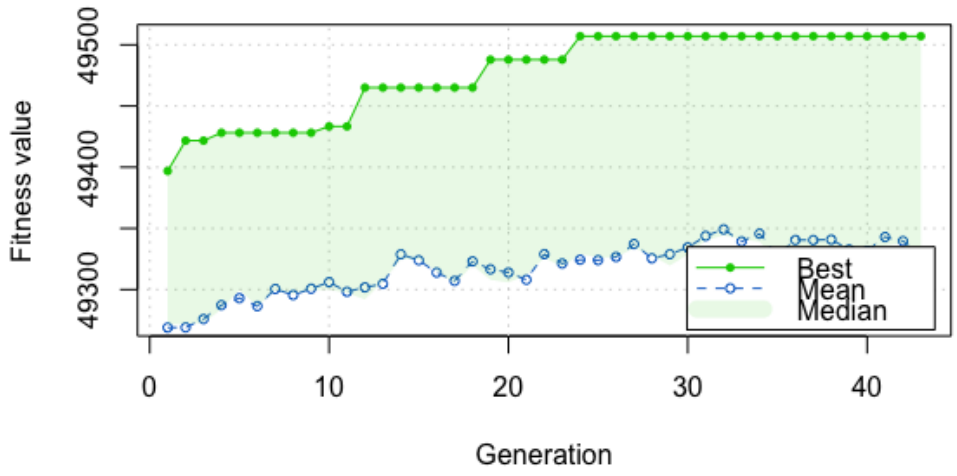

Figure 5: Iterazioni dell'algoritmo genetico

Osservando il riempimento finale dei sacchi [Fig. 6] si nota che la capacità è stata sfruttata per intero nella gran parte dei sacchi a disposizione, con qualche rara eccezione dovuta al raggiungimento del massimo numero di regali contenibili.

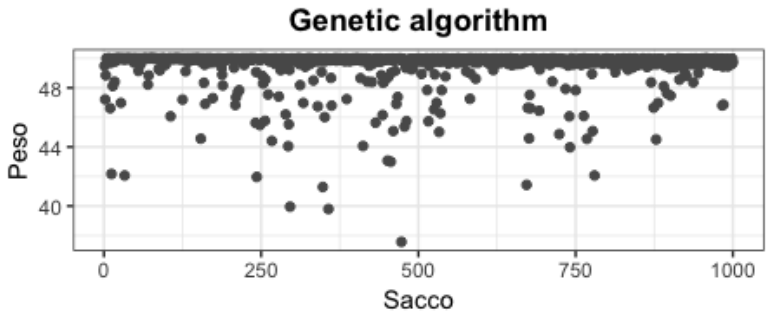

Figure 6: Peso contenuto per sacco

È importante notare che il punteggio ottenuto per l'algoritmo genetico

varia in funzione dei parametri adottati, in particolar modo, si potrebbe migliorare ulteriormente il risultato finale aumentando, alternativamente o congiuntamente, la grandezza della popolazione ed il numero massimo di generazioni. Questa operazione potrebbe far convergere verso un risultato migliore, ma non è stata effettuata in quanto richiederebbe un tempo computazionale eccessivo.

6 Conclusioni

Dati i risultati ottenuti si può affermare che, tramite l'utilizzo dell'algoritmo genetico, il problema proposto è stato risolto conseguendo un risultato prossimo al massimo ottenibile: lo scarto tra la capacità massima dei sacchi ed il punteggio ottenuto, infatti, è inferiore all'1%. Un possibile miglioramento è relativo ai tempi di esecuzione dell'algoritmo genetico che, come accennato in precedenza, dipendono fortemente dai parametri forniti all'algoritmo, i quali possono essere affinati per incrementare l'efficienza senza penalizzare eccessivamente il risultato.

References

[1] S. Martello and P. Toth, "A new algorithm for the 0-1 knapsack problem," *Management Science*, vol. 34, no. 5, pp. 633–644, 1988.

[2] J. Edmonds, "Matroids and the greedy algorithm," *Mathematical programming*, vol. 1, no. 1, pp. 127–136, 1971.

[3] J. R. Koza, "Genetic programming," 1997.

[4] L. Scrucca *et al.*, "Ga: a package for genetic algorithms in r," *Journal of Statistical Software*, vol. 53, no. 4, pp. 1–37, 2013.

[5] Y. He, J. Wang, X. Zhang, H. Li, and X. Liu, "Encoding transformation-based differential evolution algorithm for solving knapsack problem with single continuous variable," *Swarm and Evolutionary Computation*, 2019.