Università degli studi di
Milano-Bicocca

Decision Models

Final Project

Santa Gift Matching

Authors:

Borroni Alessandro - 800069 - a.borroni2@campus.unimib.it
Corvaglia Andrea - 802487 - a.corvaglia3@campus.unimib.it
Giugliano Mirko - 800226 - m.giugliano@campus.unimib.it

June 24, 2019


UNIVERSITA' DEGLI STUDI DI MILANO
BICOCCA

Abstract

Questo progetto propone una possibile soluzione a un problema di ottimizzazione combinatoria, più nello specifico si cerca di affrontare la "*Santa Gift Matching Challenge*" di Kaggle, attraverso la realizzazione di due euristiche: un *Local Search* (LS) e un *Greedy* vincolato. Si presenteranno una valutazione delle due metodologie, un confronto tra i risultati delle due euristiche, assieme con la valutazione di un utilizzo combinato delle due. Verranno inoltre affrontate le problematiche relative alla numerosità dei dati e alla presenza di vincoli imposti dal problema. Infine si proporrà un potenziale miglioramento a partire dagli algoritmi ivi implementati. Il codice è stato scritto in *Python* per l'efficacia del pacchetto *Numpy* nella manipolazione di matrici.

1 Introduction

La "*Santa Gift Matching Challenge*", è, come dice il nome stesso, una competizione. Lanciata su Kaggle nel dicembre del 2017, ha come obiettivo quello di dar vita a un algoritmo di matching che sia in grado di massimizzare la felicità, distribuendo ai bambini il regalo che più desiderano, garantendo quindi una sorta di "Natale perfetto". Vince la challenge chi riesce a trovare l'algoritmo di massimizzazione più efficiente. Questo dovrà essere in grado di assegnare un regalo a ciascun bambino, tra un milione di bambini totali, tale per cui la loro felicità complessiva sia massimizzata; il tutto basandosi su una lista ordinata che indica il grado di preferenza dei bambini in merito ai regali (dal più preferito al meno preferito). Contemporaneamente, deve essere massimizzata anche la felicità di Babbo Natale, la quale aumenta più la posizione del bambino ricevente il regalo occupa una posizione alta nella lista dei bambini buoni. Trattasi, quindi, di un classico problema di ottimizzazione, arricchito da vincoli e reso ancora più complicato dalla grandezza dei dataset. La prima difficoltà che si incontra risiede proprio nella struttura della base di dati: bisogna infatti tenere conto che lo 0.5% dei bambini, e cioè le prime cinquemila osservazioni del dataset, sono trigemelli, mentre il successivo 4% del dataset, e cioè quarantacinquemila bambini su un milione, sono gemelli. Ma non solo: sia trigemelli che gemelli dovranno, per volontà dei genitori, ricevere la stessa tipologia di regalo a prescindere dalle loro preferenze.

La funzione obiettivo è rappresentata da una felicità media, normalizzata,

definita come:

$$ANH = ANCH^3 + ANSH^3$$

dove:

- $ANCH$ è l'Average Normalized Child Happiness (o Felicità Media Normalizzata dei Bambini) e si esprime come:

$$ANCH = \frac{1}{n_c} \sum_{i=0}^{n_c-1} \frac{ChildHappiness}{MaxChildHappiness}$$

  - n_c: rappresenta il numero totale dei bambini
  - $ChildHappiness = \begin{cases} 2 \cdot GiftOrder & \text{se } Gift \in ChildWishList \\ -1 & \text{altrimenti} \end{cases}$
  - $MaxChildHappiness = len(ChildWishList) \cdot 2$

- $ANSH$: è l'Average Normalized Santa Happiness (o Felicità Media Normalizzata di Babbo Natale) e si esprime come:

$$ANSH = \frac{1}{n_g} \sum_{i=0}^{n_g-1} \frac{GiftHappiness}{MaxGiftHappiness}$$

  - n_g: rappresenta il numero totale di regali;
  - $GiftHappiness = \begin{cases} 2 \cdot ChildOrder & \text{se } Child \in GiftGoodKidsList \\ -1 & \text{altrimenti} \end{cases}$
  - $MaxGiftHappiness = len(GiftGoodKidsList) \cdot 2$

Per la risoluzione del problema ci si è concentrati sull'utilizzo di strategie di ottimizzazione basate su euristiche, consapevoli di sfruttare algoritmi che non garantiscono di ottenere la soluzione ottima, ma comunque in grado di fornire una soluzione buona oltre che ammissibile. In particolare si è posta una maggiore attenzione sul *Local Search* (abbreviato spesso cn LS), il quale consiste in un miglioramento iterativo, che a partire dalla soluzione corrente valuta, in base al vicinato, la mossa da compiere. Definita una strategia di spostamento, o strategia di ricerca, l'insieme di soluzioni che possono essere ottenute mediante una mossa, è chiamato "vicinato" (o intorno) di quella soluzione. A ogni iterazione, la soluzione corrente viene sostituita

dalla soluzione del suo vicinato. In alternativa al primo metodo, ci si è focalizzati anche su un algoritmo di tipo *Greedy* (o algoritmo vorace). Trattasi di euristica costruttiva che consiste appunto nel "costruire" una soluzione localmente ottima invece che modificarne una già esistente. Oltre a essere di facile implementazione e di notevole efficienza computazionale, il Greedy è in grado di aggiungere alla soluzione parziale l'elemento migliore possibile a ogni iterazione. Tuttavia, può capitare che tale algoritmo non garantisca l'ottimalità nè tantomeno l'ammissibilità della soluzione trovata. Infine, si è optato per una combinazione delle due tecniche risolutive nel tentativo di ottenere il miglior score possibile.

2 Datasets

I dataset utilizzabili per questa competizione sono messi a disposizione da Kaggle e sono due. Il primo è relativo ai bambini e alle preferenze in merito al regalo che desiderano rievere a Natale, il secondo, invece, è relativo a Babbo Natale e alle sue preferenze nell'assegnazione di ogni regalo a un bambino. Più precisamente:

a) **child_wishlist_v2.csv**: contiene un milione di osservazioni, in cui a ogni riga corrisponde un codice indentificativo per ogni bambino ($ChildId$) e nelle colonne che seguono la prima è possibile trovare i codici identificativi dei cento regali ($GiftId$) desiderati dal bambino, in ordine di preferenza;

b) **gift_goodkids_v2.csv**: contiene mille osservazioni, in cui a ogni riga corrisponde un codice identificativo di un regalo ($GiftId$) e nelle colonne che seguono la prima è possibile trovare i mille codici identificativi dei bambini ($ChildId$) preferiti, ordinati in base alle preferenze di Babbo Natale.

3 The Methodological Approach

Approcciandosi alla risoluzione del problema, il primo evidente ostacolo che è stato riscontrato concerne le dimensioni del dataset. Visto il numero considerevole di osservazioni, specialmente nel dataset "child_wishlist_v2.csv", i tempi di computazione sarebbero stati notevolmente lunghi e inefficienti.

Per questo motivo, la prima mossa è stata quella di creare un dataset di dimensione ridotta, in modo da permetterci di ottimizzare i tempi di computazione nello sviluppo degli algoritmi. Sfruttando questo dataset è stata per prima cosa sviluppata una funzione in grado di valutare la felicità normalizzata legata al match tra regalo e bambino; questo passaggio è risultato fondamentale in quanto ha posto le basi per la valutazione della funzione obiettivo del progetto. Si è considerata come struttura di una possibile soluzione un vettore di lunghezza pari al numero totale di bambini (e cioè un milione), in cui l'indice rappresenta il codice identificativo del bambino ($ChidId$), mentre il valore registrato è il codice identificativo del gioco assegnatogli ($GiftId$).

Superata questa difficoltà, si sono palesate le complicazioni già citate nella sezione introduttiva: che i gemelli e i trigemelli abbiano lo stesso regalo e che il codice identificativo dei regali sia ripetuto mille volte ciascuno. Per rispettare questi vincoli è stato adoperato un vettore contenente i regali disponibili (mille regali ripetuti mille volte), da cui eliminare i regali estratti a ogni iterazione. Si è optato per una estrazione casuale, con l'accortezza di eliminare il regalo estratto rispettivamente due volte per i gemelli e tre per i trigemelli. In questo modo è stato possibile ottenere una soluzione di partenza accettabile.

Sfruttando come meccanismo di ricerca uno scambio casuale tra gli elementi della sequenza (fornita dal vettore precedentemente definito), e facendo attenzione a scambiare tra loro solo bambini appartenenti alla stessa categoria (i.e. gemelli con gemelli), è stato possibile mantenere la soluzione accettabile lungo tutte le iterazioni. Sulla base di quanto detto, si è ritenuto corretto optare per un algoritmo di Simulated Annealing (SA), con l'intento di trovare una buona soluzione globale. Per valutare la felicità, quindi, è stata creata una matrice con tutte le felicità normalizzate, legate al match tra bambino (sulle righe) e regalo (sulle colonne). In questa, alla sequenza originaria è stata associata una "sequenza delle felicità corrispondenti", in modo tale che eseguendo una somma lungo le posizioni del vettore si ottenesse la bontà della soluzione.

A questo punto si è cercato di applicare lo stesso metodo al dataset di partenza, ma subito ci si è imbattuti in un problema legato alla memoria: la matrice di score (Score Matrix) con un miliardo di valori risultava ingestibile per la RAM. Anche questo problema è stato ovviato utilizzando il formato "float32" invece che il "float64", il quale risulta più gestibile in quanto occupa metà della memoria. Tuttavia, il Simulated Annealing è stato in grado di

produrre miglioramenti dell'ordine di 10^{-19} per iterazione, valori decisamente insignificanti in relazione alla felicità totale, probabilmente a causa della dimensione eccessiva della sequenza.

L'idea dunque è stata quella di veicolare lo scambio attraverso la realizzazione di un'euristica, così che l'algoritmo si muovesse nella direzione di massimo guadagno. In questo modo, quindi, si è deciso di abbandonare la risoluzione mediante Simulated Annealing a favore di un *Local Search* (LS). Il vantaggio di questo cambiamento è stato il poter veicolare i movimenti dell'algoritmo, consistenti nel selezionare il bambino con lo score più negativo, prendere il suo regalo e scambiarlo con quello del bambino che più desidera il regalo inizialmente selezionato, scegliendo tra i bambini con uno score non ottimale. L'algoritmo dà un contributo per iterazione 10^{10} volte migliore rispetto allo scambio casuale (dato dal Simulated Annealing), nonchè sempre positivo.

L'idea successiva è stata quella di usare una euristica costruttiva per dar vita a una nuova sequenza iniziale, in modo da trovarsi fin da subito nelle vicinanze di un massimo locale. In particolare, si è scelto di implementare un *Greedy Algorithm*, mediante il quale è stato possibile assegnare al primo bambino di ogni tris e di ogni duo di gemelli il regalo da lui preferito (rispettando la condizione di uguaglianza di regalo sui trigemelli e sui gemelli) e, successivamente, assegnare, fino a quando i vincoli sulla numerosità dei regali lo hanno consentito, a ogni bambino singolo il miglior regalo disponibile.

In ultima istanza, si è tentata l'implementazione di una combinazione delle due euristiche utilizzate con l'intento di valutare un eventuale possibile miglioramento sulla funzione obiettivo.

4 Results and Evaluation

- **Local Search Algorithm (LS)**

  Il Local Search con punto di partenza randomico ha portato ai seguenti risultati:

  - **Convergenza in 415'000 iterazioni e 4 ore e 32 minuti**;
  - **ANH** = 0.0001854860119340842.

- **Greedy Algorithm**

  - **Convergenza in 8.56 secondi**;

  - **ANH** = 0.00016925660690233772.

- **Modello ibrido**
  - **Convergenza in 0.46 secondi**;
  - **ANH** = 0.00016925991761012374.

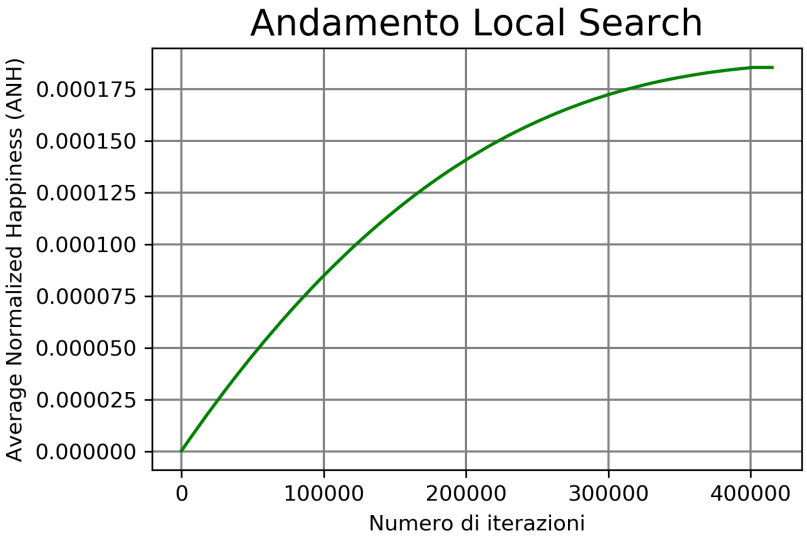

Figure 1: Andamento dello score del Local Search al passare delle iterazioni con soluzione iniziale randomica

5 Discussion

Il *LocalSearch* si è dimostrato essere efficace nella ricerca di un massimo locale, seppure computazionalmente molto costoso. Dal grafico (Figure 1) si può evincere come l'algoritmo, partendo da una sequenza casuale, arrivi a convergenza dopo 415'000 iterazioni circa. La soluzione ottenuta è significativamente migliore rispetto a una casuale.

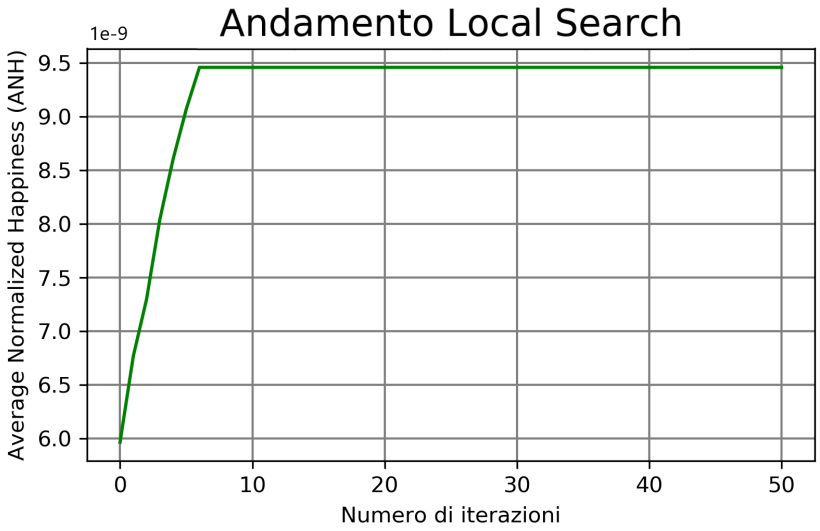

Figure 2: Andamento della variazione dello score del Local Search al passare delle iterazioni con soluzione iniziale da algoritmo greedy.

Per quanto riguarda il *Greedy Algorithm*, sono stati ottenuti risultati significativi in tempi drasticamente più corti (di circa tre ordini di grandezza). Come si vede dai risultati ottenuti con il *Local Search* appena menzionato, il *Greedy* converge a un massimo locale leggermente inferiore.

Molto interessante risulta l'applicazione congiunta delle due euristiche, la quale mostra come il Local Search impieghi solamente sei iterazioni a convergere (Figure 2), evidenziando quindi la tendenza dello stesso algoritmo a rimanere intrappolato in un massimo locale.

5.1 Improvements

Alla luce dei problemi già precedentemente illustrati, possiamo dire che con un dataset minore o con potenza computazionale maggiore, avremmo potuto intraprendere altri percorsi. Il più intuitivo esistente in letteratura è l'*Iterated Local Search* (ILS), il quale impone una perturbazione della soluzione corrente, attraverso grandi cambiamenti randomici, al fine di evitare lo stazionamento in un massimo locale. Tuttavia, dato che per avvicinarsi a un massimo

locale si impiegano circa 400'000 iterazioni, corrispondenti a circa 5 ore di computazione, risulterebbe temporalmente impossibile implementarlo con un significativo numero di perturbazioni.

Vista la lentezza dell'algoritmo di *Local Search*, un miglioramento concreto potrebbe consistere nel fornire allo stesso una sequenza di partenza in grado di ottenere fin da subito un buon score. Come si vede nella sezione dei risultati, una soluzione di partenza eccessivamente vicina a un massimo locale, come quella fornita dal *Greedy*, affossa l'algoritmo nel massimo locale più vicino, non portando a miglioramenti significativi.

Imponendo invece all'algoritmo *Greedy* una componente randomica, per esempio imponendogli una selezione casuale tra i regali più vantaggiosi (con score più alto), si otterrebbe una soluzione efficiente, riducendo il numero di iterazioni del *Local Search*, tuttavia senza finire immediatamente nel massimo locale a causa del *Greedy*. Questo tipo di algoritmo *Greedy* randomizzato prende il nome di *Greedy Randomized Adaptive Search Procedure* (GRASP). Se questo portasse effettivamente a una significativa riduzione dei tempi di convergenza del *Local Search*, aprirebbe le porte all'implementazione di un algoritmo ispirato a un *Iterad Local Search*, in cui ogni volta il punto di partenza è fornito da un *GRASP*.

6 Conclusions

Sulla base del lavoro svolto si giunge alla conclusione che l'algoritmo più efficiente risulta essere il *Greedy*, nonostante la non globalità della soluzione ottimale. L'algoritmo di *Local Search* è sì efficace, ma troppo lento.

Tuttavia, è necessario specificare come lo studio delle due euristiche appena menzionate getti le basi per un futuro sviluppo di una metaeuristica, in stile *Iterated Local Search*, composta da un *GRASP* nella sua fase iniziale e un *Local Search* in quella finale.

References

[1] Emile Aarts and Jan Karel (1997) *"Local Search in Combinatorial Optimization"*, John Wiley & Sons, Inc., New York, 1st ed.